Sidarta Fernandes

OfflineManager - uma biblioteca Android para suporte *offline* em aplicativos

Brasil

2017, Julho

Sidarta Fernandes

OfflineManager - uma biblioteca Android para suporte *offline* em aplicativos

Exame de Dissertação apresentado ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.

Universidade Federal de São Carlos – UFSCar

Departamento de Computação

Programa de Pós-Graduação

Orientador: Daniel Lucrédio

Brasil

2017, Julho

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado da candidata Sidarta Fernandes, realizada em 22/08/2017.

Prof. Dr. Daniel Lucrédio
(UFSCar)

Prof. Dr. Delano Medeiros Beder
(UFSCar)

****

Prof. Dr. Fernando Antonio Mota Trinta
(UFC)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Fernando Antonio Mota Trinta, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Sidarta Fernandes.

Prof. Dr. Daniel Lucrédio
Presidente da Comissão Examinadora
(UFSCar)

Resumo

Com a expansão da computação móvel, na forma da popularização de *smartphones*, *tablets* e também das lojas de aplicativos, e do aumento da interconectividade entre os dispositivos e servidores, diversas questões surgem no cenário da engenharia de software voltado para esse mercado. A medida que grande parte dos aplicativos opera com um servidor de *back-end* na Internet, uma questão que se destaca é a utilização de aplicativos em modo *offline*, pois em diversas ocasiões conexão com a rede não é fator garantido. Essa questão pode ser abordada de diversas formas, entre elas armazenamento das requisições, *caching*, sincronização de dados, dentre outras, e existem soluções no mercado e na academia que tentam auxiliar no tratamento desse problema, porém ainda assim existem dois desafios em aberto que precisam ser tratados: resolver o problema de forma a tratar as diversas camadas de software, incluindo a interface e questões de usabilidade, e possibilitar a resolução em qualquer etapa do ciclo de desenvolvimento sem que seja necessário grande esforço de implementação, facilitando a adaptação e a refatoração. Essa pesquisa de mestrado apresenta a construção de uma biblioteca chamada *OfflineManager*, para a plataforma Android, para auxiliar no tratamento das questões de uso em modo *offline* em aplicativos. A biblioteca foi construída para ser fácil de instalar, utilizar, e também permite a escolha dos modos de tratamento, de forma que o desenvolvedor possa configurar o comportamento para cada chamada ao *back-end* em sua aplicação, em cenário de falta de conexão tanto com a Internet, quanto com o servidor em questão. Além disso, os tratamentos oferecidos se estendem para a camada de interface, na forma de *feedbacks* informativos ou pop-up de confirmação. Para avaliar a biblioteca proposta, foram utilizados 4 metodologias diferentes: avaliação por opinião de especialista, aplicação da biblioteca em um projeto real, experimento com usuários e avaliação heurística. As quatro avaliações tiveram resultados satisfatórios sendo que as duas primeiras avaliaram conceitos técnicos e de implementação, e as duas últimas avaliaram conceitos de usabilidade. Ajudaram também na confecção de um conjunto de protodiretrizes para a escolha dos modos de tratamento, e mostraram que a biblioteca *OfflineManager* atinge o objetivo de fornecer tratamento para uso em modo *offline*, tratando a camada de interface de forma a melhorar a usabilidade, ajudando também no atendimento de algumas heurísticas de usabilidade. Algumas ameaças a validade podem ser listadas para as avaliações propostas, bem como trabalhos futuros que foram identificados durante o processo, baseados nas limitações apresentadas pela solução proposta. Dentre eles, podem ser mencionados tanto melhorias na biblioteca em si, como possibilidade de customização maior dos *feedbacks*, quanto expansão das avaliações realizadas, a fim de aprimorar os processos utilizados, ou firmar diretrizes consolidadas para utilização da biblioteca.

Palavras-chaves: Funcionamento *offline*, Desenvolvimento móvel, Biblioteca Android.

Abstract

With the expansion of mobile computing, caused by the popularization of smartphones, tablets and application stores, and the increase in the interconnectivity between devices and servers, many questions arise in the software engineering scenario. As most applications work with a back-end server over the Internet, one such question involves offline application usage, because in many occasions network connectivity is not available. This question can be solved in many ways, including storage of the requests, caching, data synchronization, among others. There are solutions both in the industry and academy that attempt to solve this problem, however two challenges remain: to solve the problem in a way that covers many software layers, including interface and usability issues; and to support any step in the development cycle without great implementation effort, facilitating adaptation and refactoring of existing applications. This dissertation presents an Android library built to help developers provide offline functionality on their apps, while trying to solving these two challenges. To do that, the library, called *OfflineManager*, was built to be easily installed and utilized, and offers different ways to treat each call to the back-end, both in case of Internet connection missing, or in case of an unavailable server. It also offers feedback messages, and a point of interaction via pop-up. To evaluate the library, four evaluation methods were used: specialist opinion, implantation on a real project, comparative experiment with users, and heuristic evaluation. All four evaluations conducted showed results indication that the library attends its purpose and caters for the two challenges presented, being that the first two evaluations analyzed implementation and development aspects, while the last two analyzed interface and usability aspects. Evaluations showed that the library is easy to install, use and comprehend, that the treatments offered are sufficient and that it improves app usability in an offline scenario. Although, some threats to validity can be listed, indicating that there is room for improvement in the way they were conducted. Finally some improvements can be presented as future work in order to improve the library, expand the experiments taken, and to consolidate guidelines for the library utilization.

Key-words: Offline use, mobile development, Android library.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SDK	*Software Development Kit*
PDA	*Personal Digital Assistant*
sTable	*Simba Table*
sClient	*Simba Client*
CRUD	*Create, Read, Update, Delete*
API	*Application Programming Interface*
REST	*Representational State Transfer*
MCI	*Master-to-Client Interface*
CMI	*Client-to-Master Interface*
HTTP	*HyperText Transfer Protocol*

Sumário

1 Introdução

1.1 Contexto

Um assunto bastante em pauta, tanto no cenário acadêmico quanto no cenário industrial, é o desenvolvimento de aplicativos para dispositivos móveis. Existem diversas questões que esse processo de desenvolvimento de software envolve, quando se considera a evolução da mobilidade do mundo e da computação, com a expansão da computação em nuvem, e dos sistemas não locais e descentralizados (PICCO et al., 2014). Uma questão que se destaca é a questão da utilização dos aplicativos em modo *offline*, e da sincronia da sua utilização com seus servidores correspondentes, na nuvem.

Isso se dá como um problema que vai contra as tendências atuais de tecnologia: enquanto a computação está cada vez mais ágil e acessível, ao alcance da mão, ainda existe a barreira que é possuir uma conexão com a Internet. Esse empecilho pode não ser tão aparente em usuários que vivem em uma grande cidade com cobertura de Internet celular, ou banda larga celular, diversos *hot-spots Wi-Fi* ou com uma boa conexão de banda larga na própria residência. Porém, basta apenas sair um pouco deste ambiente ideal para que as dificuldades apareçam. Em muitos lugares a cobertura de Internet celular pode não ser confiável; ou pode ser cara demais para ser acessível a todos; o usuário pode estar distante de sua residência, e pontos de Internet públicos podem não estar disponíveis. Sahni e Ramamritham (2007), ao proporem uma aplicação que funciona em modo *offline* para ser usada na zona rural, abordam a intermitência ou até falta de conexão nesses cenários não ideais, no caso nas zonas rurais. Por esses e outros motivos, é possível assumir que é difícil garantir uma conexão com a Internet em todos os momentos em que o usuário deseja utilizar uma aplicação. Em resumo, a maioria das aplicações precisam ter o serviço de Internet sempre disponível, mas a rede pode estar fora do ar.

Mesmo com essa questão, porém, existem hoje muitas aplicações já no mercado que resolvem esse problema até certo ponto. Ainda que seja um desafio para a engenharia de software, pois envolve diferentes partes das aplicações tanto no lado do desenvolvimento, quanto no lado de experiência de usuário, é uma questão já considerada nas soluções mais recentes. Isso pode ser observado por exemplo nos tratamentos oferecidos em algumas aplicações usadas em grande escala tais como o aplicativo de comunicação *Whats-App* ou o de mensagens do *Facebook*, que oferecem um certo grau de tratamento de uso *offline*, a fim de não comprometer a experiência de usuário em caso de intermitência ou perda de conexão. Nessas aplicações, mesmo quando uma mensagem é enviada e o usuário que envia encontra-se desconectado da Internet, a mensagem é armazenada e a requisição salva, para que quando o sistema identificar uma nova rede, ela seja enviada e a interação continue de

maneira transparente. Pode ser observado também no aplicativo de documentos Google Drive, que permite o trabalho em documentos no modo *offline*, sincronizando com a nuvem quando houver disponibilidade de rede[1].

As formas como essa questão pode ser abordada se relacionam diretamente com o domínio das aplicações onde o tratamento *offline* será inserido, e também com as tecnologias escolhidas para tratarem desse requisito. Em aplicações de cunho social, como as citadas anteriormente, um mecanismo transparente que armazena a mensagem e envia quando a conexão for detectada parece resolver o requisito de funcionamento *offline*. Já em aplicações bancárias, por exemplo, pode ser desejável que um tratamento mais explícito ou cuidadoso seja feito, a fim de evitar que o usuário cometa erros, por exemplo executando a mesma ação repetidas vezes. Ainda em outros cenários, pode ser que a abordagem seja em mais baixo nível e resolva o problema de uso em modo *offline* com sincronia de dados: a aplicação mantém uma cópia local dos dados e as tarefas são executada neles – quando houver uma conexão estável com a Internet, essa cópia local é sincronizada com o servidor, mantendo a consistência de dados.

Nesta tese, Android foi a plataforma alvo do estudo, e considerando as possibilidades mencionadas no parágrafo anterior, algumas possibilidades de implementação e abordagem para o requisito *offline* são:

- Usar um mecanismo para interceptar as chamadas realizadas ao servidor, e implementar *cache* de dados de forma a armazenar os resultados, de forma a utilizá-los em um segundo momento caso a conexão não esteja disponível;
- Usar classe de serviços do Android, que executam em segundo plano dado que critérios foram atendidos (como presença de conexão), para realizar atualizações ou sincronizações de dados com servidores;
- Usar servidores sandbox (como por exemplo o Firebase) que provêm um banco de dados e um serviço de sincronização já implementado; e
- O aplicativo pode cancelar a ação e avisar ao usuário que naquele instante a rede está indisponível e que ele terá que tentar mais tarde. Como exemplo pode-se citar os próprios navegadores *web*, que normalmente avisam que não é possível prosseguir sem conexão com a Internet, e não permitem nenhuma interação com o conteúdo inicialmente desejado.

Esses são apenas alguns exemplos dos tratamentos possíveis. É possível perceber que nenhum dos mecanismos ou abordagens mencionadas acima tratam a questão da

[1] https://support.google.com/docs/answer/2375012?hl=pt-BR

apresentação e interface para o usuário. Isso faz com que o desenvolvedor, além de arquitetar a estrutura e tecnologia para prover funcionamento *offline* em seu aplicativo, ainda precisará pensar nas questões de usabilidade e apresentação.

Além desses que foram citados, existem outras formas de se resolver o problema, porém há dois pontos que são críticos:

1. Para oferecer suporte ao funcionamento *offline*, ainda que o projeto conceitual seja simples, é necessário um esforço de implementação que passa por todas as camadas de desenvolvimento de software: desde o *back-end*, passando pelas camadas de dados e da implementação das lógicas de negócio da aplicação, até o *front-end* e a camada de apresentação e interface com o usuário. Por esse motivo, questões como usabilidade não podem ser ignoradas. Como é um esforço de implementação que envolve diversas camadas de uma aplicação, o desenvolvedor precisa projetar o sistema considerando esse questionamento desde o início do ciclo de desenvolvimento e da concepção, para poder prover em seu produto uma solução que se estenda em todos os aspectos do software; e,

2. Independente do momento em que o desenvolvedor escolhe aplicar uma solução para esta questão em seu aplicativo, seja como incremento, refatorando e refazendo a aplicação, seja como requisito inicial a ser desenvolvido desde o começo da aplicação, a solução escolhida terá que ser projetada e implementada à mão pelo desenvolvedor. Mesmo que as plataformas possuam ferramentas para facilitar essa tarefa, existe a dificuldade de estender a solução desejada a todos os níveis conceituais do sistema. Isso também significa que se um desenvolvedor quiser prover essa solução em um aplicativo já pronto, ele terá que aplicar muitos recursos e esforço para sua refatoração e reconstrução.

Considerando o atual estado da prática, existem algumas ferramentas que facilitam a aplicação de requisitos de funcionamento *offline* até certo ponto. Especificamente, na plataforma Android, existem diversas bibliotecas para armazenamento de dados e *caching*, e alguns padrões de implementação usando bibliotecas de comunicação HTTP para interceptar e armazenar resultados. Existem também alguns *frameworks* para desenvolvimento multiplataforma de aplicações que tentam resolver esse problema: um exemplo disso é o `Xamarin`[2] – plataforma de desenvolvimento de aplicativos móveis que usa as tecnologias da Microsoft, e fornece um mecanismo para sincronização automática de dados.

Existem também propostas acadêmicas interessantes que serão abordadas posteriormente no capítulo 2. Algumas propondo uma solução completa para a questão de

[2] https://xamarin.com/

uso em modo *offline*, como o trabalho de Perkins et al. (2015), outras propondo alguns mecanismos para auxiliar o tratamento, como o trabalho de Peters et al. (2011).

Contudo, pode-se observar que os esforços se voltam para resolver a questão de funcionamento *offline* das seguintes formas: usando sincronização dos dados (mantendo a consistência entre os dados locais e os armazenados em servidores), espelhamento, e também por meio de *caching* – armazenamento local de dados mais acessados – melhorando assim sua disponibilidade. Essas soluções são bastante interessantes pois oferecem algumas opções diferentes de tratamento quando o desenvolvedor se depara com este tipo de problema em seu ciclo de desenvolvimento.

Mas, como destacado anteriormente, nenhuma das opções existentes, tanto na academia como na indústria, abrange totalmente os pontos críticos mencionados simultaneamente. Principalmente o primeiro ponto crítico, que pode ser considerado o principal, uma vez que o segundo é mais facilmente resolvido e algumas soluções já o consideram. uma vez a Isso porque nenhum dos mecanismos estudados engloba o problema em todas as camadas em que ele se apresenta, que incluem também a interface com o usuário e o gerenciamento das requisições e respostas de interações; e também não abordam o problema de uma maneira que ele possa ser tratado em um momento tardio do ciclo de desenvolvimento, fazendo com que o desenvolvedor precise planejar as adaptações todas no início, se quiser evitar um grande esforço de refatoração em uma etapa posterior do ciclo de vida de seu sistema.

Esse trabalho de mestrado tomou como base esse contexto e essas motivações, para propor a construção de uma ferramenta que facilite o processo de desenvolvimento no cenário de utilização *offline*. A próxima seção mostra em detalhes quais foram os objetivos por trás da construção da biblioteca *OfflineManager*, aqui apresentada.

1.2 Objetivos

O objetivo deste trabalho foi construir uma biblioteca para tentar resolver os pontos críticos citados na seção anterior, permitindo ao desenvolvedor de aplicações inserir em seus projetos certo grau de uso em modo *offline*. A biblioteca construída permite que o desenvolvedor escolha, para cada chamada, qual o modo de tratamento *offline* tanto para o caso de o dispositivo estar sem conexão, quanto para o caso de o servidor se encontrar indisponível. Isso permite que o desenvolvedor aplique a biblioteca de forma granular, pois é possível e comum que o requisito seja implementado de forma diferente para cada chamada ou para cada ponto de realização das requisições, melhorando a aplicabilidade e flexibilidade da biblioteca, no cenário de uma aplicação real. A biblioteca permite também a escolha de métodos diferentes de tratamento, de forma a atender diferentes características de chamadas e contextos de aplicações, que podem ser por exemplo:

- Contexto de aplicação social, ou de conteúdo, ou chamadas que podem ou não ser executadas em segundo plano. Não exige um nível extremo de garantia de conexão;

- Contexto de aplicação ou de chamadas que, apesar de poderem ser executadas em segundo plano, ainda precisariam de certo controle do usuário, pois as requisições possuem dependência de contexto e de momento, ou seja, dependendo do momento ou do contexto, o usuário pode ou não desejar o funcionamento *offline*;

- Contexto de aplicação ou chamada mais robusta, onde garantia de conexão e *feedbacks* imediatos são um requisito principal, ou chamadas dependentes de tempo e momento, que não fariam sentido serem executadas em um segundo momento. Neste contexto, a conexão com a Internet é mandatória e o funcionamento *offline* fica restrito apenas a bons *feedbacks* para o usuário final.

Este trabalho visa também fazer um estudo de usabilidade, com experimento de utilização e comparação, e também com avaliação heurística, a fim de avaliar as melhorias de usabilidade ao aplicar uma biblioteca que trata um requisito em todas as camadas da aplicação, e não somente na camada de dados mas também na camada de interface.

Mais detalhes sobre a biblioteca construída são apresentados no Capítulo 3, onde são explicados as bases tecnológicas para a codificação da biblioteca, e o estudo realizado para decidir sobre a melhor abordagem, em vista das diferentes opções de tratamento possíveis. No Capítulo 4 são apresentadas as diferentes técnicas utilizadas para a avaliação tanto da biblioteca em si, quanto da utilização da mesma sobre um aplicativo real e pré-existente.

A principal contribuição deste trabalho é facilitar para o desenvolvedor de aplicações móveis a inserção de funcionamento em modo *offline* em seus projetos, de maneira simples e fácil, sem se preocupar com detalhes técnicos de implementação, e de forma que o tratamento seja feito nas diferentes camadas do aplicativo: desde a parte de comunicação e armazenamento até a camada de interface – fornecendo *feedbacks* e interações com o usuário, com intuito de melhorar a usabilidade. Dessa forma, o desenvolvedor consegue se concentrar nos outros detalhes do projeto que sejam mais próximos do domínio real da aplicação, e ainda assim fornecer um nível extra de usabilidade, que é o funcionamento em modo *offline* – requisito que pode não ser mandatório, mas agrega bastante valor no cenário atual da computação móvel, que evidencia cada vez mais alta dependência de conectividade com a Internet porém sem garantia de ter conexão sempre ativa.

1.3 Metodologia de pesquisa

Para a realização dessa pesquisa de mestrado, foram seguidos os seguintes passos:

1. Avaliar os possíveis cenários de uso *offline*: passo realizado pelo autor desta dissertação, com base em estudo da literatura e aplicativos reais, e discussões dentro do grupo de pesquisa e da comunidade de desenvolvimento de aplicativos móveis;

2. Analisar os diferentes contextos de aplicações móveis e o requisito de funcionamento *offline* nesses contextos: passo realizado pelo autor desta dissertação, com base em estudo da literatura e pesquisa sobre aplicativos reais e de mercado;

3. Definir a plataforma na qual a abordagem seria estudada e implementada, dentre as mais comuns no cenário atual – iOS e Android: definição feita pelo autor desta dissertação, com base nas características de cada plataforma;

4. Avaliar as diferentes possibilidades de implementação da abordagem: passo realizado pelo autor desta dissertação, com base em discussões dentro do grupo de pesquisa e características da plataforma escolhida. A abordagem escolhida foi desenvolver uma biblioteca Android;

5. Avaliar conceitos de usabilidade no cenário de desenvolvimento de aplicações móveis: passo realizado pelo autor desta dissertação com base em estudo da literatura;

6. Definir o escopo da biblioteca, os modos de tratamento, e os pontos de interação com o usuário e *feedback* que a biblioteca poderia oferecer: passo realizado pelo autor desta dissertação com base em estudo da literatura e discussões dentro do grupo de pesquisa e da comunidade de desenvolvimento de aplicativos móveis;

7. Desenvolver a biblioteca: passo realizado pelo autor desta dissertação;

8. Processo de avaliação por especialista, do ponto de vista técnico e de aplicabilidade: passo realizado pelo autor desta dissertação em conjunto com o grupo de especialistas selecionados, com base em estudos da literatura sobre esse método de avaliação, e obtenção de resultados pela agregação de respostas à avaliação proposta;

9. Processo de aplicação da biblioteca à um projeto real de aplicativo: passo realizado pelo autor desta dissertação, com base em discussões dentro do grupo de pesquisa e nas características do aplicativo em questão;

10. Experimento de avaliação de comparação do aplicativo com a biblioteca e sem a biblioteca: passo realizado pelo autor desta dissertação em conjunto com o grupo de participantes selecionados para o experimento, e resultados obtidos com base em discussões dentro do grupo de pesquisa e agregação das avaliações submetidas;

11. Avaliação de heurísticas que a biblioteca atende: passo realizado pelo autor desta dissertação, com base em estudo da literatura e discussões dentro do grupo de pesquisa;

12. Proposição de melhorias e trabalhos futuros para a biblioteca: passo realizado pelo autor desta dissertação, com base nos passos executados anteriormente e em discussões dentro do grupo de pesquisa e da comunidade de desenvolvimento de aplicativos móveis.

O primeiro passo para o desenvolvimento foi, naturalmente, um estudo dos possíveis cenários onde o uso em modo *offline* se torna interessante. Devido a grande quantidade de aplicativos para dispositivos móveis nas lojas (Play Store e Apple Store), existe uma gama de contextos nos quais eles são utilizados. É compreensível que cada um desses contextos exija requisitos de implementação diferenciados, dentre eles pode estar o requisito de funcionamento *offline*. Assim, foi feita uma análise sobre os domínios mais comuns de aplicação para que a abordagem consiga atender os requisitos desses domínios. Em seguida foi definida a plataforma Android como alvo da implementação devido a familiaridade com a linguagem, e assim, feitas as avaliações técnicas pra definir de fato o que seria a abordagem.

Por fim, a definição foi de construir uma biblioteca: recurso facilmente distribuível a fim de facilitar o processo de instalação e utilização por outros desenvolvedores.

A biblioteca possui um mecanismo interno para armazenamento das chamadas e características de cada chamada, e oferece uma interface simples, na forma de um método, para que o desenvolvedor encapsule as chamadas ao *back-end*. Nesse método, é possível escolher os modos de tratamento tanto em caso de dispositivo *offline* quanto de servidor, e outras variáveis como número de tentativas em caso de servidor *offline* e se vai ser aplicado o modo verboso ou não. O modo verboso significa que todas as possíveis interações da biblioteca para funcionamento *offline* fornecerão *feedbacks* para o usuário.

Depois de ser implementada, a biblioteca passou pelos seguintes processos de avaliação: avaliação por opinião de especialista, aplicação da biblioteca *OfflineManager* a um projeto real, experimento de comparação de usabilidade entre um aplicativo real sem o uso da biblioteca e com a utilização da biblioteca, e avaliação de heurísticas sobre os pontos de interação da biblioteca, para concluir em qual grau ou em quais heurísticas ela melhora (ou piora) a experiência de usuário.

A aplicação real escolhida foi o Inventum[3]: uma aplicação de código livre e disponível no Github[4]. Essa aplicação mostrou-se adequada para uma avaliação inicial pois se aproxima de uma aplicação real no cenário móvel, apesar de não apresentar grande variedade nos tipos de requisições que realiza.

Após as avaliações, observou-se que a biblioteca *OfflineManager* traz as seguintes

[3] https://github.com/qqq3/inventum

[4] Plataforma baseada na web de hospedagem de código e controle de versão, para projetos públicos ou privados. Comumente utilizada para hospedar projetos de código livre

contribuições: facilidade de instalação, compreensão e aplicação, tratamento para diversos cenários e contextos de utilização de aplicativos, pouco consumo de recursos do dispositivo, e aborda tanto a parte de tratamento das chamadas quanto as camadas de interface do aplicativo. Dessa forma, ajuda a resolver os dois pontos críticos destacados na seção 1.1. No entanto, algumas limitações foram observadas, como o alcance em que a biblioteca consegue tratar a interface, pois oferece apenas um ponto de interação de confirmação, e mensagens de *feedback*.

Outras características e tratamentos, que se relacionam mais com o contexto do aplicativo, ainda precisam ser tratados pelo desenvolvedor durante sua implementação. Trabalhos futuros foram propostos sobre as limitações observadas, ou sobre possibilidades de expansão dos estudos realizados nesta dissertação. Por exemplo: melhorar a estrutura interna da biblioteca de armazenamento das chamadas, construir a biblioteca *OfflineManager* para a plataforma iOS, e estudar a possibilidade de utilizar conceitos de desenvolvimento de software dirigido por modelos, para especificar tratamentos mais refinados dos que os oferecidos pela biblioteca, a fim de aumentar o nível de abstração da solução.

1.4 Organização desta dissertação

Este primeiro capítulo apresentou o contexto sobre o qual esse trabalho de mestrado foi desenvolvido, os objetivos desta pesquisa e também a metodologia utilizada durante seu desenvolvimento.

No Capítulo 2 traz-se a revisão bibliográfica, a fim de apresentar tanto o atual estado da prática, quanto o atual estado da arte, sobre o contexto trabalhado no escopo deste trabalho.

No Capítulo 3 é apresentada a implementação da biblioteca, bem como os detalhes do projeto e as decisões sobre suas características, de maneira mais completa e detalhada. Explica também seu mecanismo de funcionamento e suas dependências.

No Capítulo 4 mostra-se quais os processos de avaliação foram utilizados, que foram: opinião de especialista, aplicação em uma aplicação já existente, experimento comparativo de usabilidade e avaliação heurística. Para cada método, apresenta uma introdução sobre o processo avaliativo e em seguida os resultados obtidos com cada um desses processos de avaliação. Argumentações também são feitas sobre esses resultados, que validam a solução proposta neste trabalho de mestrado.

Por fim, no Capítulo 5 apresentam-se as considerações finais, discorrendo sobre a contribuição que este trabalho representa no âmbito da engenharia de software, considerando os resultados obtidos, bem como a contribuição da biblioteca produzida para a

comunidade de desenvolvimento Android, uma vez que ela está disponibilizada e documentada no Github. Apresenta também proposições para trabalhos futuros: possibilidades que foram descobertas durante o desenvolvimento deste projeto.

2 Revisão de Literatura

Este capítulo apresenta uma breve revisão da literatura relacionada a este projeto de pesquisa. Na Seção 2.2 discute-se o *estado-da-prática*: são apresentadas e discutidas técnicas e ferramentas existentes atualmente e que podem ser utilizadas para resolver parte do problema-alvo desta pesquisa. Na Seção 2.3 discute-se o *estado-da-arte*: são apresentados e discutidos trabalhos acadêmicos que possuem proposta relacionada à desta pesquisa, e que constituem avanço ainda não consolidado na indústria.

Os trabalhos acadêmicos aqui apresentados foram encontrados nas plataformas digitais *ACM Digital Library*[1], *Springer*[2] e *IEEE Xplore*[3] após buscas com as palavras chave: *offline, mobile* e *android* e selecionados baseados em sua relevância com o contexto e com a solução proposta neste trabalho. Essa revisão da literatura, realizada de forma semi-sistemática, foi feita durantes os anos 2014 até o início de 2016.

Antes de entrar diretamente nas discussões das tecnologias presentes no mercado e propostas da academia para o tratamento da questão do uso de aplicativos em modo *offline*, é preciso apresentar primeiramente três conceitos que estão bastante presentes, tanto no mercado quanto no cenário acadêmico, e que são o foco de uma grande parte das propostas de tratamento para esta questão. São os conceitos de *cache*, *proxy* e *pre-fetching*, apresentados a seguir.

2.1 Conceitos relacionados

Cache é conceitualmente um componente computacional onde se armazenam dados para que possam ser acessados mais rapidamente, quando forem acessados novamente. No contexto da Internet, pode também ser denominado *Web cache*, e é um componente presente no *browser* ou em aplicações que interagem com a Internet, sendo empregado para armazenar conteúdos resultantes de interações com a Internet para que futuras requisições aos mesmos conteúdos sejam mais facilmente respondidas, pois já foram previamente armazenadas. *Web cache* também pode não ser local e sim estar localizado em servidores de *proxy* para atender a mais de um usuário. Neste caso, o ganho é que os dados mais frequentemente acessados são armazenados em servidores mais próximos aos usuários, diminuindo o tempo de resposta e melhorando o desempenho no geral.

Proxy ou Servidores de *Proxy* são agentes que podem ser computadores ou aplicações, e funcionam como mediadores na comunicação entre clientes e servidores e podem

[1] http://dl.acm.org/
[2] https://link.springer.com/
[3] http://ieeexplore.ieee.org/Xplore/home.jsp

ser utilizados com as mais variadas finalidades. Usualmente, os mais comuns são os *Web Proxies* – sistemas distribuídos pela topologia da *World Wide Web* para tornar mais fácil o acesso a conteúdos, encapsulando as comunicações e também muitas vezes possuindo mecanismos de *caching*, diminuindo as latências entre as requisições e respostas dos diversos clientes e servidores presentes na rede.

Existe também o *pre-fetching*, que de modo geral significa adiantar o processamento de alguma informação, antes dela ser requisitada, com base na possibilidade dela ser necessária posteriormente, para adiantar o processo e melhorar o desempenho. No contexto da Internet, é um conceito bastante utilizado e consiste no pré-processamento de certos conteúdos possíveis de serem requisitados, para que sejam rapidamente servidos quando forem requisitados, diminuindo a latência e consumo de banda.

2.2 Revisão do estado-da-prática

Grandes empresas já proporcionam o suporte para funcionamento com ou sem conectividade em alguns de seus aplicativos, como por exemplo o aplicativo do Facebook ou o aplicativo Google Drive citados no Capítulo 1. Mas isso não é uma preocupação recente: o próprio Google disponibilizou uma tecnologia no passado chamada `Gears`, para a tentativa de prover utilização em modo desconectado para suas aplicações que rodavam somente na Internet. Esse esforço mostra que essa questão já era uma preocupação desde o início da expansão dos *smartphones*, quando o foco ainda era a navegação na Internet e aplicações descentralizadas (aplicações web). Esta tecnologia permitia que essas aplicações fossem mais poderosas, adicionando a capacidade extra aos navegadores de prover arquivos que inicialmente só estariam disponíveis na Internet, mesmo sem conexão. Resumindo seu funcionamento, o `Gears` era uma extensão para os navegadores que armazenava alguns arquivos da Internet localmente, permitindo sua utilização em modo desconectado. O `Gears` funcionou principalmente e quase que exclusivamente para as ferramentas do próprio Google, antes de ser descontinuado em 2010, pois suas capacidades foram diretamente embutidas no próprio padrão Web do HTML5[4].

Essa tecnologia, apesar de sua breve aparição, tem uma participação no cenário da computação, com relação a temática aqui discutida. Isso porque foi uma das primeiras tentativas de prover interação em modo desconectado a aplicativos web, que até então só funcionariam com a presença de conexão. Foi importante também pois a sua ideia central de funcionamento se assemelha à de alguns dos trabalhos científicos atuais com relação a tentativas de resolver esse problema, que são usando *caching* e *proxy*: mecanismos que hoje em dia estão disponíveis diretamente nos padrões do HTML5, mas que precisam se apresentar em soluções refinadas para englobar questões de consistência de dados e

[4] http://gearsblog.blogspot.com.br/

desempenho.

Existem também algumas tecnologias mais recentes disponíveis para que os próprios desenvolvedores possam abordar de forma facilitada o problema de falta de conectividade em seus aplicativos, por exemplo o SDK (*Software Development Kit* ou Kit para desenvolvimento de software) disponível no `Xamarin` – plataforma de desenvolvimento de aplicativos móveis.

2.2.1 Xamarin

O `Xamarin`[5] é uma plataforma completa para desenvolvimento de aplicativos multiplataforma, que permite ao desenvolvedor programar em uma única linguagem, o C#, para depois produzir código nativo para as principais plataformas móveis do mercado – iOS, Android. Além disso, esse *framework* possibilita que o desenvolvedor trate a questão da interação em modo desconectado diretamente com algumas bibliotecas nativas.

Com isso, o aplicativo, que deve ser construído em conjunto com o serviço de nuvem da Microsoft, chamado Azure[6], poderá ser utilizado mesmo sem conexão com a Internet, que os dados manterão uma consistência, mesmo estando distribuídos na nuvem e em outros dispositivos. A arquitetura dessa plataforma então funciona por meio da sincronização de dados: o aplicativo a ser construído deve utilizar as APIs da tabela de sincronização, que são as mesmas APIs das tabelas normais de dados e permitem as operações de criar, ler, atualizar e excluir. Porém, estas falham quando tentam se conectar diretamente ao servidor para realizar as operações, enquanto aquelas, se não houver conexão com a rede, executam as operações de leitura e escrita em um armazenamento local.

Para que essas operações sejam possíveis no armazenamento local, é preciso que este seja inicializado, e consiste na camada de persistência de dados de um aplicativo no dispositivo do cliente, podendo ser qualquer tipo padrão de armazenamento local. O que é importante porém para o mecanismo de sincronização funcionar é o contexto de sincronização.

Um contexto de sincronização é associado a um objeto de cliente móvel. Este por sua vez é um objeto que oferece serviços básicos à aplicação para se vincular ao servidor Azure. Esse contexto rastreia as alterações feitas nas tabelas de sincronização e mantém uma fila que conserva a ordem das operações, para que possam ser repassadas para o servidor em um segundo instante.

Para que a sincronização ocorra, existem dois métodos que devem ser executados explicitamente, e são o *pull* (Extração) e o *push* (Envio). Esses métodos são responsáveis

[5] https://xamarin.com/
[6] https://azure.microsoft.com/

por baixar o conteúdo do servidor e atualizar os dados do armazenamento local; e por enviar as atualizações realizadas localmente para o servidor para sincronizar as operações realizadas, mantendo a consistência de dados. Existem alguns outros mecanismos, mas todos são combinações desses dois métodos, pois são eles os responsáveis pelo processo de sincronia de dados. Existe por exemplo o *push* implícito que acontece quando um *pull* é executado em uma tabela que possui alterações locais não sincronizadas. Neste caso, o *pull* primeiramente enviará as alterações locais ao servidor antes de realizar a extração, para que assim os dados sejam sincronizados no servidor, antes de serem baixados para o contexto local novamente.

Com base nessa arquitetura, a forma como o desenvolvedor implementa isso é a seguinte: o desenvolvedor precisa primeiramente modelar quais partes do sistema terão suporte à interação *offline* porque isso será provido com sincronismo do banco de dados. Por isso, depois de planejar, o desenvolvedor precisa programar o modelo de dados utilizando as tabelas de sincronização e suas APIs para que a sincronização seja possível e gerenciada automaticamente. Isso tudo precisa ser planejado de antemão pelo desenvolvedor pois envolve os tipos de tabela de armazenamento de dados que serão utilizados, e também envolve algoritmos específicos para a execução das operações de sincronia dos dados. Assim, se ele desejar refatorar um sistema para adicionar essa funcionalidade, terá um grande esforço de implementação pois terá que percorrer todo o sistema realizando alterações no modelo de dados e também na lógica de funcionamento, para inserir os mecanismos necessários de contexto, de envio e de extração.

Além disso, ao utilizar essa abordagem, algumas questões ainda precisam de tratamento manual, como por exemplo a camada de interface. Isso significa que a usabilidade da aplicação em modo *offline*, com sincronização, não é tratada pelo *framework*, uma vez que esse apenas trabalha na sincronia dos dados. Isso pode ser um problema quando o uso do aplicativo é refinado, requerendo atenção e confirmação do usuário para que o fluxo ocorra. Quando a interface não é tratada em um cenário de usabilidade em modo *offline*, a experiência pode ser afetada pois o fluxo de interação e resposta pode ser comprometido. Isso ressalta o ponto crítico 1, apresentado no Capítulo 1.

Outro ponto interessante nessa plataforma é que apesar da sincronização funcionar de maneira automática, sua implementação precisa ser bem especificada. Isso pois os processos de atualização e sincronismo dos dados são transparentes ao usuário, mas no código é preciso que os métodos de *pull* e *push*, explicados anteriormente, sejam chamados explicitamente. Esses mecanismos abstraem qualquer complexidade de rastrear dados novos ou não, pois as tabelas de sincronização, em conjunto com o contexto de sincronização, já fazem essas marcações. Mas adicionam o pequeno trabalho para o desenvolvedor de identificar os pontos no código onde esses métodos devem ser executados. Ou seja, caso deseje integrar isso com fluidez no aplicativo, tornando-o transparente ao estado de

conexão, ele deve primeiro identificar as mudanças no estado de conexão do aparelho, para que nessa situação ele execute a sincronização convenientemente.

Esse tipo de solução pode ser comparada com as outras a serem apresentadas no sentido de que esta é uma solução mais prática, que fornece ao desenvolvedor uma solução real à questão do uso em modo desconectado, que já é utilizada por um grande número de desenvolvedores.

2.3 Revisão do estado-da-arte

Antes de apresentar a revisão do estado-da-arte e dos trabalhos relacionados que foram encontrados, é importante contextualizar o cenário da computação móvel nessas últimas duas décadas, que foi o período onde mudou-se o paradigma de interação com dispositivos computacionais e com a própria Internet.

2.3.1 Breve histórico e contextualização

Mobilidade no âmbito da computação é um termo com muitas nuances. Tomando com consideração um artigo publicado no início de 2000, época em que o mundo estava começando a entrar na era móvel, computação móvel era vista pela engenharia de software como a área de estudo onde os componentes computacionais podem mudar de lugar (ROMAN; PICCO; MURPHY, 2000). Ou seja, a definição de computação móvel geralmente era usada para identificar aparelhos computacionais autônomos, geralmente conectados a uma rede de Internet sem fio, e que podem se mover livremente – o usuário pode levar esse aparelho a qualquer lugar que desejar. Essa definição de mobilidade porém, pouco tempo depois, sofreu alguns questionamentos. As evoluções nos aparelhos móveis que ocorreram a partir da data deste artigo citado fizeram o conceito de mobilidade sair do âmbito puramente físico, passando a ter uma conotação lógica importante: além da mobilidade física dos aparelhos, a parte lógica de um sistema também poderia ser móvel, e por questões de projeto o código ou estado do sistema poderia se mover entre as partes distribuídas do mesmo.

Duas décadas depois, muita coisa mudou nesse cenário. A explosão das lojas online de aplicativos, a mudança de interação computacional (de computadores pessoais, para *smartphones* e *tablets*), e a melhoria significativa da capacidade computacional dos dispositivos móveis fizeram com que esses conceitos de mobilidade já não fossem mais novidade no mundo científico. A mobilidade física tornou-se bastante acessível e comum, e alguns dos padrões de projeto relacionados a mobilidade de código tornaram-se lugar-comum nas implementações de sistemas distribuídos; sendo que outros dos padrões caíram em desuso, sendo mencionados apenas pelo seu teor histórico-evolutivo (PICCO et al., 2014).

Picco et al. (2014) fazem essa contextualização da evolução da engenharia de software, com foco na mobilidade, buscando trazer uma breve visão geral desses fenômenos citados acima. O artigo traz, desde o início, uma breve descrição de como era o cenário mobile na época, bem como os desafios identificados. Tais desafios eram, por exemplo:

- A definição da representação de localização e contexto na modelagem de sistemas. São dois conceitos semelhantes, porém com características distintas;
- O desafio dos algoritmos, que precisariam englobar questões de limitação de energia (bateria), mudanças de localização, quedas de conexão, variação de recursos dependendo do aparelho, dentre outras; e
- Mudanças drásticas nas condições de conexão e localização do aparelhos. No início, conexão era rara e custosa, e os aplicativos seguiam o caminho de aplicações locais e pontuais, oferecendo pouca ou quase nenhuma interação com entidades terceiras. No entanto, com a evolução das redes sem fio, os aparelhos conectados, em conjunto com sensores e com GPS, passaram a oferecer um novo grau de integração e colaboração entre sistemas diferentes.

Todos esses desafios derivavam da necessidade em oferecer ao usuário um novo grau de consciência sobre mobilidade, pois a usabilidade das aplicações pode depender diretamente de alguma decisão do próprio usuário. Ainda, as aplicações também precisariam se adaptar às diferentes características de hardware, tamanhos e tipos de tela e infra-estrutura dos aparelhos, o que pode ser problemático se resultar em menor usabilidade.

Picco et al. (2014) ainda apontam quais descobertas ou eventos foram os principais para mudar o rumo que esta tecnologia estava tomando, e estes foram, primeiramente a evidente e grande expansão e diversidade dos dispositivos móveis. Enquanto no início o previsto era a evolução dos aparelhos móveis conhecidos na época como *laptops* e PDAs (*Personal Digital Assistant* ou Assistentes Digitais Pessoais), não era imaginado o grau de penetração que os *smartphones* iriam atingir, e nem o grau com que a diversidade na natureza dos dispositivos influenciaria a mobilidade em si. Com essa expansão explosiva, é evidente que surgiu um grande número de sistemas de software confeccionados especialmente para os *smartphones*, e como consequência, diversas questões de software surgiram desse nicho específico da computação.

Outro fator de grande impacto, diretamente relacionado com o anterior, é o aumento da capacidade computacional desses dispositivos, e junto com isso, os dispositivos de hardware e sensores, que com a grande variedade em que se apresentam nos novos dispositivos, aumentam de maneira significativa o modo como o dispositivo móvel interage com o ambiente e com o próprio usuário. Os autores ressaltam que nesse sentido, o

módulo *wireless* (a placa *Wi-Fi*) foi de fato um dos mais importantes e, apesar de não ter recebido devida atenção quando surgiu, hoje ele é um dos componentes mais importantes e um marco na computação ubíqua, participando de diversas novas tendências, como Internet das coisas, por exemplo. Alguns outros sensores ou componentes que também influenciaram o processo de software foram o giroscópio, GPS e acelerômetro – novidades no início, mas quase itens indispensáveis hoje, devido a todos os softwares e aplicações que necessitam deles.

Por fim, é citado também o fator do impacto causado pelas redes sociais, no cenário da computação móvel. Essa classe de aplicações mudou o paradigma de interações com os dispositivos móveis. Os aparelhos que até então serviam apenas como assistentes nas relações físicas entre usuários passaram a ser o meio principal de interação para a conexão entre usuários de todo o mundo, e também a mais frequente das interações com os dispositivos.

Pode-se observar que alguns dos questionamentos levantados no Capítulo 1 já eram considerados desafios para a computação móvel desde praticamente o seu nascimento. No momento em que se difunde a conexão sem fio para os dispositivos móveis, a fim de permitir uma maior mobilidade e independência computacional, aparece como desafio o fator de que a conexão pode nem sempre estar presente.

Portanto, como o artigo de Picco et al. (2014) mostra, a computação móvel tende a evoluir ainda mais, e ainda existem muitas grandes descobertas por vir neste meio. Isso faz com que os desafios na engenharia de software envolvam cada vez mais questões de usabilidade e de interação com o ambiente e com os usuários, uma vez que a tecnologia portátil está cada vez mais inserida na vida e no cotidiano das pessoas.

2.3.2 Trabalhos relacionados

Feito o levantamento histórico e contextualização, parte-se agora para estudos que tratam diretamente a proposta de trabalhar em modo *offline*. Alguns deles voltam as atenções para a parte conceitual do problema. Dessa forma tentam buscar uma solução resolvendo as questões da consistência de dados. Para isso, utilizam por exemplo sincronismo dos bancos de dados, ou abordam protocolos que facilitam essa sincronização, não tratando assim as requisições e interações diretamente, e sim dando atenção a uma camada mais abaixo – os próprios dados distribuídos. Outro artigo procura abordar o problema por um prisma mais conceitual e social. Para isso, introduz a problemática em um cenário bastante comum no mundo: a intermitência de conexão nas zonas rurais. Nesse caso a solução estudada trabalha um sistema que pode ser usado em modo desconectado, com o objetivo de integrar os habitantes dessas zonas rurais com a Internet, e além de focar nas questões tecnológicas que tornam isso possível, também foca nas questões de usabilidade neste cenário. E por fim, um artigo trabalha o problema como uma questão diretamente

relacionada com interação e navegação web: sua proposta envolve mecanismos de *cache*, *pre-fetching* e *proxy*.

Esses trabalhos são analisados um a um, pois todos mostram contribuições relevantes no cenário da engenharia de software voltada à computação móvel, com foco no uso em modo *offline* e sincronismo de dados. É interessante discutir aspectos de cada proposta que resolvem questões diferentes desta problemática; onde eles se assemelham e onde se diferem da proposta a ser estabelecida aqui; e quais das questões críticas do cenário mobile, introduzidas no Capítulo 1, eles conseguem atender ou ainda deixam a desejar.

2.3.3 Perkins et al. (2015)

O primeiro artigo a ser discutido é o artigo de Perkins et al. (2015), que propõe a construção do serviço de sincronização `Simba`. Mas, antes de apresentar essa proposta, o artigo faz um estudo bastante interessante como motivação.

Perkins et al. (2015) primeiramente estruturam um estudo com diversos aplicativos populares para *smartphones*, sob o prisma da sincronização e consistência de dados. Eles configuram cada aplicativo para rodar em dois aparelhos diferentes, ambos rodando o sistema operacional Android e ambos com o mesmo usuário logado no aplicativo a ser testado. Nesses aparelhos depois foram executadas diversas ações a fim de simular cenários reais de utilização com conflitos de interação e de utilização de dados compartilhados. Para a análise, foram estabelecidos dois conceitos:

1. Para analisar questões de granularidade, foi analisado o esquema local de dependência entre dados tabulares e dados de objetos dos aplicativos; e

2. Para analisar questões de consistência, foi observado sob o ponto de vista de um usuário quais aplicativos ampliam o conceito de *server-only* (sincronização simplesmente com o servidor) para *server and client* (aplicativos que propagam tanto para o servidor quanto para outras instâncias do aplicativo). Essa análise foi feita sob o ponto de vista do usuário, com ações iniciadas por ele.

Esses dois conceitos possibilitaram a identificação e classificação do comportamento dos aplicativos estudados de acordo com a seguintes características:

- Necessidades de consistência diversificadas: aplicativos onde questões de consistência são tratadas em níveis diferentes, em diferentes partes do aplicativo;

- Semântica de sincronia alheia a consistência: alguns aplicativos simplesmente impõem uma política de escrita das últimas interações, sem considerar questões de

consistência. Funciona para apenas acrescentar informações, mas não gerencia nenhum tipo de conflito e torna o sistema inconsistente;

- Suporte *offline* limitado: aplicativos que já exibiam erros com uso *offline*, ou simplesmente não permitiam interação quando *offline*.
- Propagação de erro inadequada: aplicativos que quando entram em uma situação de conflito de dados, exibem comportamentos aleatórios e até interrupções; e
- Atomicidade violada de dados granulares: aplicativos que precisam guardar dados interdependentes, estruturados ou não, mas a notação de sincronia só permite dado tabular ou de arquivo, gerando inconsistência e violação da atomicidade.

Com base em todas essas observações desse experimento com aplicativos já existentes, e as formas de tratamento da consistência de dados, o artigo apresenta dois principais pontos em que os aplicativos falham em tratar:

1. Granularidade dos dados inadequada pois os aplicativos só tratam dados em tabelas e em objetos, mas nenhuma representação engloba os dois (e por isso falta uma representação boa quando as interações envolvem os dois tipos); e
2. Falta de um tratamento de consistência configurável, pois os aplicativos forçam a escolha de uma mesma semântica de consistência para todos os dados do aplicativo, não permitindo tratamentos diferentes em porções específicas do sistema.

Assim, Perkins et al. (2015) propõem um novo serviço de sincronização de dados baseado na nuvem. Junto com esse novo serviço, o artigo apresenta a criação de uma nova abstração de dados para a utilização com esse serviço chamada `sTable` (*Simba Table*). A `sTable` faz com que todos os dados armazenados nela sejam sincronizados de maneira transparente com a nuvem e com outros dispositivos. Para isso, oferece um esquema de armazenamento de dados interdependentes, sejam eles dados tabulares ou objetos, e permite ao desenvolvedor especificar qual o esquema de consistência de dados utilizar, dentre uma coleção de opções.

É importante porém mencionar que o `Simba` não é um simples serviço que pode ser facilmente configurado em qualquer aplicativo. Para entender o seu funcionamento, será explicado a seguir todos os componentes necessários para que este serviço seja possível.

Primeiramente, e diretamente em contato com os aplicativos móveis, existe o que é chamado de `Simba SDK`, que é simplesmente uma ligação entre o aplicativo que vai usar a API do `Simba`, e o *Simba Client* (ou `sClient`). O *Simba Client* é um serviço que roda em plano de fundo no dispositivo e oferece um armazenamento local confiável para os dados locais de aplicações. Também fornece a interface com os *gateways* do *Simba Cloud*, que é

a parte do servidor da arquitetura do `Simba`, na forma de um protocolo de sincronismo, e é o responsável por gerenciar o sincronismo no lado do dispositivo e do cliente.

No *Simba Cloud* existem os *gateways* que gerenciam as conexões com os diversos *Simba Clients*. Nele também são armazenados os próprios dados, gerenciados de acordo com os esquemas de consistência, e separados em nós para que esta solução seja escalável tanto em questão de aumento do tráfego quanto no aumento de dados. É este componente que engloba todo o serviço no lado da nuvem e do servidor.

Além desses componentes existe ainda o *Simba Protocol*, que é o protocolo de comunicação entre o `sClient` e o `sCloud`. Este protocolo opera de modo específico para gerenciar as ações de sincronização de dados entre os componentes, e é especialmente confeccionado de acordo com a política de consistência que está sendo usada pelo aplicativo.

Por fim, depois de explicada toda a arquitetura do `Simba`, o artigo propõe métodos de avaliação da solução e da abstração criada, e conclui falando um pouco sobre o desempenho e sobre a usabilidade para desenvolvimento com este serviço. As questões propostas para a avaliação desta solução são as seguintes:

- Leveza do protocolo de sincronização;
- Desempenho do `sCloud` em operações CRUD (*Create, Read, Update, Delete* ou Criação, Leitura, Atualização e Exclusão);
- Escalabilidade do `sCloud` relativo ao número de clientes e tabelas;
- *Trade-off* entre consistência e latência; e
- Facilidade em implementar uma aplicação usando o `Simba`.

Todas essas questões foram respondidas com testes da solução em alguns casos específicos de estresse e de escalabilidade, tanto de dados quanto de usuários. Em resumo, a solução se mostrou bastante robusta quanto a todos os quesitos de performance, como o artigo comprova em seus resultados. O que é interessante, porém, no caso deste trabalho, é analisar o último tópico, que se refere a facilidade em escrever uma aplicação usando o `Simba`.

Perkins et al. (2015) apresentam duas avaliações neste sentido: primeiro, eles portam um aplicativo simples para que este passe a se beneficiar de múltiplos modelos de consistência usando o `Simba`. É um aplicativo que usa o serviço do Dropbox[7] para manter e sincronizar dois arquivos. Os arquivos ativos usam o modelo robusto e são sincronizados sempre que há algum tipo de alteração. Já os itens arquivados, por não serem frequentes,

[7] https://www.dropbox.com/

usam o modelo eventual de consistência – as mudanças não são propagadas instantaneamente por não serem operações críticas no sistema.

Os autores afirmam que modificar o aplicativo para usar o `Simba` foi um procedimento simples e com um bom ganho pois tirou qualquer necessidade de o próprio usuário iniciar operações de sincronismo – todas são feitas automaticamente de acordo com o modelo escolhido.

Já o segundo caso utilizado nesta avaliação é um aplicativo que apresenta inconsistência e a ideia foi consertá-lo com o `Simba`. Neste item houve um pouco mais de dificuldade. A aplicação utilizada é um sistema para armazenar dados e senhas de um usuário. A inconsistência estava presente com a atualização concorrente de dados de usuário. Foram tentadas duas abordagens: a primeira envolvia transformar todo o banco de dados da aplicação em um objeto de uma `sTable`, e isso significava mudar o servidor de armazenamento do Dropbox para nuvem do `Simba`. Mas esta solução não foi considerada pois não tratava alterações granulares e individuais. Outra solução tentada foi separar cada conta de usuário da aplicação em uma linha na tabela de abstração (`sTable`), pois dessa forma alterações seriam tratadas de maneira individual, para cada usuário.

Nestes dois casos de avaliação, o autor afirma que o processo de sincronismo automático e em plano de fundo foi muito vantajoso para essas aplicações, e que o processo de portar uma aplicação para usar o `Simba` é bastante simples e direto.

Analisando de um modo geral este trabalho publicado por Perkins et al. (2015), algumas passagens merecem destaque: a questão levantada de que é importante permitir que sejam escolhidos níveis diferentes para tratar a sincronia dos dados é bastante relevante e é um tópico mencionado no Capítulo 1, ou seja, é um tópico ainda em aberto neste cenário pois influencia bastante não só o funcionamento em plano de fundo das soluções, mas também afeta questões de usabilidade e interação, uma vez que dados e ações diferentes em uma aplicação podem ter níveis diferentes de importância naquele sistema. Saber identificar quais são as mais importantes, ou como moldar a solução para tratar esses níveis diferentes é algo bastante importante para que a solução seja genérica e possível de se usar pelos desenvolvedores, em diferentes aplicações.

Outro ponto importante de ressaltar é que o artigo não apresenta como contribuição apenas a nova abstração de dados proposta, mas sim a API inteira do `Simba` que funciona como um serviço completo – com partes no dispositivo e partes na nuvem – para a adição de sincronismo e consistência de dados em aplicações que podem ser usadas em modo *offline*. E isso leva a outra ressalva: esta solução não foi pensada só para aplicações que possuem uso em modo *offline*, e nem possui como motivação essa questão. A motivação da proposta é manter a consistência de dados em aplicações distribuídas. Dentro dessa motivação, como algumas dessas aplicações podem ser usadas mesmo sem conexão com a Internet, e isso pode gerar conflitos e inconsistência, então a solução se estende.

Porém o artigo não aborda por exemplo questões específicas relacionadas ao trabalho em modo desconectado como por exemplo questões de usabilidade para notificações quando o aparelho estiver sem conexão, dentre outras, mencionadas no Capítulo 1.

2.3.4 Do et al. (2014)

Outro artigo interessante (DO et al., 2014) trata sobre a mesma questão da interação *offline* mas em um cenário mais específico, que é o das redes sociais. É um caso específico bastante interessante, pois hoje em dia é comum a utilização de dispositivos móveis para o acesso a conteúdos sociais. Assim, é justificável uma proposta que trate de maneira eficiente essas interações, sem que seja necessária conexão com a rede o tempo todo, visando oferecer uma melhor experiência de usuário.

O artigo, quando foi escrito, mostra que o principal problema das redes sociais para aparelhos móveis, como por exemplo o aplicativo do Facebook, é que eles esperam sempre estar conectados com a rede. Caso não haja conexão, o aplicativo simplesmente informa que a conexão foi perdida, e qualquer interação social do usuário fica indisponível. O aplicativo baseia seu fluxo em enviar as requisições de atualizações quando é iniciado ou sob demanda do usuário, sempre considerando e dependendo de conexão com a Internet. Partindo desse ponto, o autor argumenta então que a possibilidade de interação em modo *offline* seria de fato um grande ganho para a experiência dos usuários nesta classe de aplicação. O alto grau de mobilidade, presente nessa classe de aplicação, pode resultar em momentos de falta de conexão, e assim impedir os usuários de interagirem com a aplicação e receberem atualizações sobre seus amigos ou familiares. Se houvesse um suporte de interação em modo *offline*, o usuário poderia continuar interagindo socialmente, de maneira transparente, sem sofrer uma experiência ruim de usabilidade, independente de sua localização ou qualquer outro fator externo que influencia o estado de conexão de seu aparelho.

Para isso, os autores implementam uma arquitetura baseada em *proxy/broker*[8], onde a ideia é armazenar os dados em um *proxy*, salvando dessa forma somente os conteúdos com maior probabilidade de serem vistos pelo usuário.

Um diferencial é que o próprio sistema determina a relevância do conteúdo, por usuário, com base nas atualizações fornecidas pelas redes sociais, para poder decidir quais conteúdos baixar. Importante mencionar que isso é um ganho com relação às atualizações que essas redes sociais fornecem por padrão, pois enquanto elas apenas emitem notificações para ações que envolvem o próprio usuário (como solicitações de amizade, por exemplo), este sistema consegue avaliar relevância de conteúdo que envolva sua rede de amigos ou outros conteúdos que o usuário queira ver, que podem não ser uma interação direta.

[8] *Broker* é uma estrutura intermediária em um sistema distribuído que funciona como mediador na comunicação entre clientes e servidores

Com esse conhecimento, Do et al. (2014) desenvolveram um algoritmo de escalonamento para o download de conteúdo nos dispositivos. Sobre esse algoritmo, foi desenvolvido um aplicativo para Android que navega pelo Facebook de maneira *offline*, para testar o desempenho da solução proposta. Os autores concluíram que a solução atende aos requisitos de navegação *offline*, e que a implementação é eficiente em termos de consumo de bateria.

O trabalho de Do et al. (2014), apesar de suas contribuições, não resolve nenhum dos dois desafios apresentados no Capítulo 1. O primeiro desafio não é atendido porque o autor trabalha diretamente em um cenário bastante específico, que é o dos aplicativos de redes sociais, e assim o modelo de tratamento já foi previamente escolhido para melhor atender a este cenário. A solução já foi construída e moldada diretamente para este cenário, e assim não oferece opções diferentes de tratamento. Também não é uma solução genérica que pode ser utilizada em outros cenários, pois os algoritmos como os de *pre-fetching* são amarrados diretamente às características do conteúdo de aplicativos sociais, por exemplo: notificações e atualizações de amigos, fotos, etc, ou seja, conteúdos sociais. O segundo desafio não é tratado pois o trabalho não aborda nenhuma técnica que permita a refatoração de aplicativos para que passem a utilizar a solução proposta. A estrutura de *broker* e *proxy* proposta que analisa e armazena os conteúdos relevantes é de certa forma genérica e independente de aplicação, pois funciona separadamente, mas os algoritmos de relevância funcionam para conteúdo de redes sociais. Além disso é necessário que exista uma aplicação especialmente implementada de forma que possa consumir esses dados pré-processados dessa estrutura e prover o uso das redes sociais em modo *offline*, aplicação esta que será a interface com o usuário. Portanto, caso o engenheiro de software queira empregar esta solução em seu aplicativo, terá que trabalhar manualmente em todas essas adaptações.

2.3.5 Sahni e Ramamritham (2007)

Existe também um trabalho interessante de ser mencionado que apesar de não apresentar uma solução direta e genérica para a interação em modo *offline*, discute o problema em um nível que explora mais a questão da acessibilidade dos dados, considerando que a Internet não é algo presente em todo o território habitado. O trabalho de Sahni e Ramamritham (2007) começa discutindo sobre a importância do conhecimento para a evolução das civilizações, e como é importante a questão da acessibilidade de conhecimento e conteúdo para populações em zonas rurais. É pensando nisso e com essa parte da população em mente que o artigo mostra a construção de um portal chamado `aAQUA` – *Almost All Questions Answered* – que permite a adição de conteúdo a fim de construir e enriquecer sua base de dados para disponibilizar informação e conhecimento.

Neste portal, além do desafio do conteúdo em si, existe o problema da conexão

com a Internet. É nesse desafio que o artigo se mostra interessante para o contexto desta pesquisa. O cenário rural é um lugar onde a conexão com a Internet muitas vezes é precária, intermitente, isso quando ela existe. Assim, disponibilizar o conteúdo para esses lugares por meio de uma plataforma web se apresenta como um grande desafio. Os autores discutem também uma questão interessante que é a experiência de usuário considerando essas condições de conexão. Se um usuário fosse até um quiosque com terminais para acessar o sistema pode ser que ele enfrente muitos atrasos e lentidão na conexão e utilização do sistema, ou pode ser até que ele não consiga interagir naquele momento, devido à perda de rede. Então o artigo mostra a solução proposta, chamada de `offline aAQUA`, que é o portal porém com mecanismos que permitem sua utilização em modo *offline*, permitindo a utilização da plataforma de maneira transparente independente do estado da conexão.

Depois dessa introdução, o artigo discute sobre as alternativas de navegação em modo *offline* existentes na época. A primeira apresentada é o *caching* das páginas ou do conteúdo por um mecanismo chamado de *fish search*: consiste basicamente em baixar e armazenar uma página, e depois recursivamente baixar e armazenar todo o conteúdo para que os links dessa página encaminham. Este método não é muito adequado para todos os cenários pois é uma abordagem que acaba por baixar muito conteúdo replicado, desperdiçando banda e armazenamento.

Um segundo mecanismo citado é a Replicação Homogênea da Base de Dados, que consiste em disponibilizar uma cópia da base de dados, que se mantem sincronizada com a base principal sempre que houver Internet disponível, em um servidor mais próximo ao cliente que irá acessar os dados. Apesar do possível ganho de performance pois a réplica próxima ao cliente diminuiria o atraso, é uma solução que não se viabiliza para o cenário rural pois requer infraestrutura dedicada e de alta performance.

Outro mecanismo discutido é um que se chama *Value-Based Cache*, e funciona de forma similar ao primeiro citado. Porém, a fim de evitar desperdício de recursos com conteúdos que já foram armazenados, esta solução propõe quebrar o conteúdo em blocos de dados, e acessos aos mesmos blocos de dados simplesmente armazenam indicadores para a instância armazenada daquele dado. O autor afirma que é uma opção eficiente, mas que necessitaria de um *proxy* para cada cliente para armazenar esses dados de referência.

Um último método apresentado é a possibilidade de usar um leitor de RSS ou um *News Reader* – leitor genérico de notícias. Mas não se aplicam ao escopo do `aAQUA` pois o portal requer que todo o conteúdo da plataforma, inclusive pesquisas e buscas, esteja disponível, e não somente atualizações ou novas postagens.

Apresentadas algumas tecnologias possíveis, os autores fazem uma avaliação sobre as características de cada uma, a fim de agrupar e identificar os fatores-chave necessários para o funcionamento ideal do portal. Eles apresentam então a arquitetura utilizada para o funcionamento do `offline aAQUA`, que se baseia no que eles chamam de Sincroniza-

ção de Dados Heterogênea. O funcionamento, de forma resumida, consiste em armazenar, do lado do cliente, apenas meta-dados relacionados aos conteúdos, em um banco de dados simplificado; e o conteúdo das postagens é armazenado em um repositório de *cache* separado, isso para não sobrecarregar o banco. Enquanto isso no servidor os dados são armazenados por completo, além dos metadados e outras informações necessárias. Essa estruturação permite atualizações inteligentes, com resolução de conflitos, para a sincronia dos dados nos dois lados, permitindo que o cliente forneça a interface e a interação mesmo em modo *offline*. A sincronização acontece apenas sobre os dados que foram modificados, reduzindo assim a carga e a redundância, e o banco de dados do lado do cliente evita que dados comuns tenham que ser enviados mais de uma vez.

Quanto à performance desta solução empregada no `offline aAQUA`, os autores apresentam os resultados de testes realizados com 594 visitas ao portal, ao longo de um período de 5 dias, analisando logs tanto do cliente quanto do servidor, para avaliar os aspectos de performance, consumo de banda, tempo de resposta e tempo necessário de conexão. Os resultados são apresentados em uma tabela e a conclusão sobre eles é que a arquitetura escolhida mostra um desempenho melhor do que as outras tecnologias possíveis, apresentadas anteriormente no artigo. Além disso, o fato de apresentar uma boa performance com relação a tempos de resposta e atrasos, o portal também mostrou uma melhoria significativa na experiência de usuário, atingindo sua proposta de estar disponível para o usuário de maneira transparente, independentemente do estado da conexão com a Internet.

Analisando esse trabalho sob os aspectos discutidos e apresentados no Capítulo 1, pode-se dizer que ele não apresenta ideias que possam resolver os dois desafios levantados. Isso se dá porque este artigo não busca apresentar uma ferramenta ou alguma solução genérica para a utilização de sistemas em modo *offline*. Ele propõe a construção de uma plataforma de conteúdo, a ser disponibilizado em uma página web, motivado pelas questões de falta de acessibilidade no campo. Porém, apesar de parecer um trabalho que se distancia da ideia central, ele de fato agrega conhecimento, porque Sahni e Ramamritham (2007) mesmo visando uma plataforma diferente, se deparam com o próprio desafio de prover funcionalidade em modo *offline* ao seu sistema; consequentemente, seu trabalho mostra estudos, algoritmos e possibilidades interessantes de solução, e também evidencia a questão da usabilidade, bastante mencionada anteriormente.

Dito isso, esse artigo mostrado não apresenta soluções para os dois desafios, mas serve para evidenciar o segundo deles ao se atentar a experiência do usuário do campo, quando este fosse utilizar o `aAQUA` para a navegação e aquisição de conhecimento.

2.3.6 Peters et al. (2011)

Por fim existe também um outro artigo interessante de ser apresentado e discutido aqui, é o artigo de Peters et al. (2011) que propõe um modelo de replicação de dados baseado em um protocolo REST (*Representational State Transfer* ou Transferência de Estado Representacional). O REST é um estilo arquitetural que abstrai os componentes distribuídos da Internet para enfatizar as questões de escalabilidade de interações, interfaces genéricas, distribuição independente de componentes e componentes de mediação para a diminuição de latência na comunicação. Ele fornece um conjunto bem definido de regras para o design de componentes que integrarão um sistema distribuído, resultando em uma arquitetura mais fácil de se manter e de melhor performance (FIELDING; TAYLOR, 2000)

Peters et al. (2011) procuram tratar o problema se voltando apenas para a questão da consistência de dados e para o processo de sincronização. Os autores afirmam que protocolos de sincronização desenvolvidos para os sistemas tradicionais não funcionam em um cenário móvel, devido às características da natureza dos aplicativos. Por isso, algumas propriedades precisam ser consideradas, como por exemplo o fato de os clientes não estarem sempre conectados e por isso a necessidade de um critério menos restritivo de garantia dos dados.

Ao mesmo tempo, os autores justificam a escolha de trabalharem com o protocolo REST, pois é uma tecnologia bastante popular tanto para a construção de aplicações web quando para aplicativos móveis, que podem facilmente consumir recursos nessa tecnologia.

Dessa forma, os autores apresentam a proposta de criar um protocolo otimizado, que chamam de *Client-Centric Replication Protocol* - `CCR-Replication` (ou Protocolo de Replicação Centrado no Cliente), que funciona para sincronizar dados de bancos relacionais sem mecanismos de trava e que transmite para o servidor e para outras instâncias da aplicação apenas os dados alterados, a fim de reduzir o consumo de banda. É construído sobre a interface REST e passível de ser integrado com as aplicações que trabalham com ela sem muito esforço de implementação. O módulo principal de sincronização fica do lado do cliente, facilitando essa integração com a interface REST, mas também possibilitando que os conflitos sejam gerenciados de forma independente por cada cliente. Resumindo as características mostradas acima, a proposta se baseia em: usar a interface REST, transmitir somente dados que foram alterados, centrar a resolução de conflitos no lado do cliente e trabalhar em bancos sem mecanismos de trava

Explicando um pouco melhor a arquitetura da solução apresentada por Peters et al. (2011), o mecanismo funciona para uma estrutura com um servidor e um número arbitrário de clientes. Do lado do servidor, está presente a camada do banco de dados e também uma aplicação web responsável pelo gerenciamento e comunicação com os

clientes, que opera sobre a interface REST. A sincronização entre o servidor e os clientes é feita por dois métodos: o `MCI` (*Master-to-Client Interface* ou Interface Mestre-Cliente), que é a requisição que o servidor envia para o cliente solicitando as atualizações de dados e disparando o processamento, se necessário, dos dados e dos conflitos no cliente; e o `CMI` (*Client-to-Master Interface* ou Interface Cliente-Mestre) que é o envio das atualizações para o servidor.

Para o gerenciamento dos dados a serem sincronizados, a estrutura de dados de negócio do aplicativo no cliente precisa ser alterada para adicionar uma estrutura de logs, que armazenam as últimas alterações que necessitam de sincronização. Também é adicionado um atributo no modelo de dados, chamado de chave remota, que serve para mapear os dados dos clientes com os dados referentes no servidor. Além disso, os dados novos e as atualizações são mantidos com marcadores de tempo, enviados pelo servidor em cada requisição `MCI`, para que com comparações com os marcadores anteriores, seja possível identificar quais dados devem ser sincronizados.

Com essa estrutura, os aplicativos podem ser utilizados de maneira transparente, inclusive em modo desconectado e as sincronizações são realizadas por esses dois métodos apresentados. Como a natureza da interface REST é baseada em requisições singulares e simples, é necessário uma requisição para cada classe de modelo a ser sincronizada. E a sincronização funciona de maneira que o servidor primeiro envia uma requisição `MCI` para o cliente, solicitando os dados novos e atualizações resultantes de processamento, e o cliente responde com a requisição `CMI`, contendo as atualizações de sincronização que devem ser refletidas no servidor.

Neste processo de sincronização existem alguns cenários onde podem ocorrer conflitos. Os autores explicam como esses cenários podem ocorrer e como eles podem ser tratados. O primeiro é o cenário de exclusão de dado, onde ambas as partes deletam um dado. Este cenário pode ser desconsiderado pois ele é simplesmente ignorado pelo sistema: o dado será apagado de qualquer forma. Os outros cenários possíveis são os de exclusão em um lado e atualização do outro (lado significando cliente ou servidor); e atualização em ambos. O tratamento nesses dois casos depende do momento em que o conflito é detectado: caso seja desde o começo do processo, o conflito pode ser detectado pelo cliente quando este recebe a requisição do servidor e ela contém informação sobre dados que mudaram no servidor, mas não mudaram no log do cliente. Neste caso o cliente pode resolver localmente os conflitos, como é proposto desde o início, e se necessário atualizar os dados no servidor. E no caso de o conflito ter sido detectado pelo servidor ao receber as atualizações do cliente (no `CMI`), uma troca extra de mensagens é necessária para informar o conflito ao cliente para que ele possa executar as estratégias de resolução (PETERS et al., 2011).

Como pode ser observado, os autores tratam somente de um cenário envolvido na

questão de utilização de aplicativos em modo *offline*. Eles propõem um mecanismo mais eficiente para o tratamento do processo de sincronismo entre os dados de aplicativos e seus respectivos servidores, e abordam isso usando uma tecnologia bastante comum no mercado que é a interface REST de comunicação.

Apesar de suas contribuições, o trabalho de Peters et al. (2011) também não atende aos pontos críticos apresentados no Capítulo 1. Por ser uma solução específica e voltada diretamente para a sincronização e replicação de dados, ela não oferece opções diferentes de tratamento ao engenheiro de software, envolvendo diferentes tipos de abordagem ou possibilitando a utilização em diferentes tipos de aplicações. Principalmente, não oferece um tratamento que cubra as diversas camadas de software de uma aplicação. Ela apenas atua na camada de dados, por trabalhar com replicação, e de comunicação, por ser baseada na proposta de um novo protocolo de comunicação. Outras camadas do aplicativo como por exemplo a interface não são abordadas. Também não resolve o segundo ponto crítico pois é uma solução que para ser implantada precisa de alterações, ainda que mínimas, no aplicativo e no servidor, por exemplo adição das estruturas de *log* e de marcações de tempo, e a proposta não aborda nenhum mecanismo que facilite ou automatize esse processo, nem em caso de implantação em um aplicativo já pronto. Assim, esforço de implementação sempre será necessário, tanto em um momento inicial, desde o planejamento, quando em uma etapa tardia do desenvolvimento, em uma refatoração.

2.3.7 Considerações finais

Sumarizando os conceitos aprendidos com a revisão da bibliografia, uma comparação pode ser vista na tabela 1.

Trabalho ou Solução	Descrição
`Xamarin`	Solução de mercado. Se apresenta na forma de uma plataforma completa para desenvolvimento mobile multiplataforma. Nessa plataforma existe o tratamento para utilização em modo offline. Tratamento é feito por meio de estruturas de dados específicas propostas e sincronização de dados, que devem estar na nuvem da Microsoft – chamada *Azure*
Perkins et al. (2015)	Propõe a construção do serviço de sincronização `Simba`, baseado na nuvem. Apresenta nova abstração de dados (`sTable`), além de uma estrutura completa que deve ser usada, desde um módulo cliente que reside no dispositivo móvel, até um módulo de servidor ou *cloud*. Para o funcionamento *offline*, o aplicativo deve ser adaptado para essa estrutura e utilizar as APIs e protocolos definidos.
Do et al. (2014)	Propõe uma estrutura de *proxy/broker* com algoritmo de *prefetching* para baixar previamente (quando há conexão) dados mais relevantes e com mais chances de serem consumidos pelo usuário. Foi feito especificamente para o contexto de redes sociais.
Sahni e Ramamritham (2007)	Apresenta a construção de um portal para dispobilização de informação para populações rurais com difícil acesso a internet. Por causa do cenário rural, o portal apresenta o problema de uso em modo *offline*. A proposta é a utilização de um sistema de sincronização baseado em cache, com atualizações inteligentes dos dados, para manutenção da consistência de dados entre o portal e o servidor. Assim, as informações ficam mais atualizadas e disponíveis mesmo quando não houver conexão.
Peters et al. (2011)	Apresenta um protocolo REST otimizado, voltado para a questão da consistência de dados sincronizados. A ideia central é transmitir somente os dados modificados, reduzindo o consumo de banda, e mantendo todas as instâncias (aplicações e servidores) sincronizadas.

Tabela 1 – Tabela sumarizando a solução de mercado e os trabalhos acadêmicos apresentados neste capítulo

3 OfflineManager - uma biblioteca Android para suporte *offline* em aplicativos

Como já citado anteriormente, a proposta deste trabalho foi fornecer um mecanismo rápido e em alto nível para que um desenvolvedor possa implantar certo grau de funcionamento *offline* em sua aplicação, sem ter que se preocupar com detalhes mais técnicos e de baixo nível relacionados a esse domínio. Isso permite que ele se concentre mais no domínio de sua aplicação, mas ainda assim forneça uma boa usabilidade quando o celular do usuário estiver sem conexão ativa com a Internet. A ideia central do projeto, foi que essa abordagem oferecida passasse por todas as camadas da aplicação, sendo de certa forma um tratamento completo, e não apenas em uma das camadas do projeto de uma aplicação. Isso significa que a proposta apresentada nesta dissertação trata não somente dos detalhes técnicos de funcionamento em um cenário de falta de conexão, mas trata também as questões de interação com o usuário e mensagens de retorno. Este capítulo explica de forma detalhada os passos e estudos executados para as decisões do projeto, bem como as motivações para a construção da biblioteca desenvolvida. Explica, também, como a biblioteca funciona, os tipos de tratamento oferecidos e também as estruturas adotadas para que ela funcione da forma esperada. É apresentado o modo como a biblioteca deve ser aplicada em um projeto, por meio de exemplos de código, que mostram essas estruturas e o método de tratamento.

3.1 Arquitetura da Solução

A biblioteca *OfflineManager* não foi implementada sobre nenhum padrão de projeto específico. Porém, funciona sobre uma arquitetura bem definida. Essa arquitetura geral será apresentada nesta seção, e suas partes serão explicadas em detalhes nas próximas seções.

O funcionamento básico da biblioteca requer apenas que o desenvolvedor substitua a chamada HTTP realizada em sua aplicação, no caso feita por uma biblioteca de comunicação por exemplo o Retrofit, por uma chamada ao método `treatedCall(...)`. A Figura 1 mostra como a aplicação executa uma ação com chamada ao servidor em um cenário normal, usando apenas o Retrofit e antes da aplicação da biblioteca *OfflineManager*.

Como pode ser visto na figura 1, o primeiro evento (evento 1) é uma ação do usuário no aplicativo, que dispare uma chamada da internet requisitando algum dado. Essa ação pode ser, por exemplo, um clique no aplicativo que requisite uma lista de registros do servidor, para ser mostrada no aplicativo. Essa requisição é realizada no

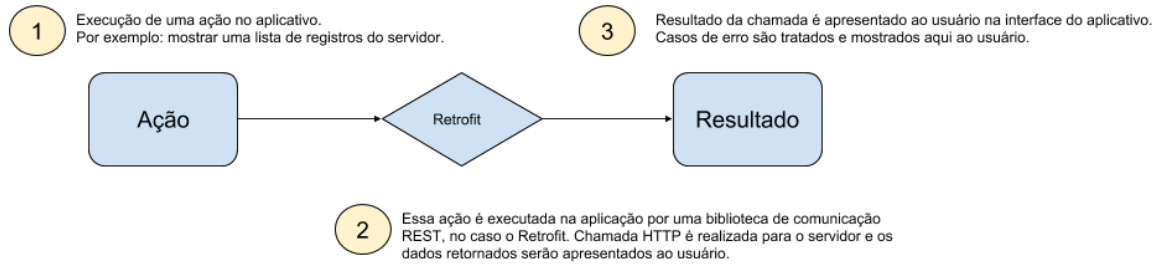


Figura 1 – Fluxo simples de uma ação com chamada HTTP a um servidor

aplicativo por meio da biblioteca Retrofit (evento 2), que encapsula a comunicação com o servidor para realizar a chamada HTTP, retornando os dados requisitados caso haja conexão com a internet no momento da chamada. Por fim (evento 3), os dados retornados são tratados e apresentados no aplicativo. Caso haja erro nesse fluxo, por falta de conexão com a internet por exemplo, os dados não estariam disponíveis e a mensagem de erro seria apresentada nesta etapa. O usuário, então, teria que realizar o fluxo novamente para uma nova tentativa de obter os dados desejados.

Já a Figura 2 mostra o mesmo fluxo com a aplicação da biblioteca *OfflineManager*, ilustrando de forma simplificada como é o processo de utilização da biblioteca para o tratamento *offline* em uma chamada HTTP do aplicativo Android.

Na figura 2, a requisição ao Retrofit é encapsulada pelo método `treatedCall`, presente na biblioteca *OfflineManager*. Para o mesmo fluxo de ação onde o usuário primeiramente realiza alguma ação de forma a requisitar um dado da internet, o segundo evento neste caso é a chamada que é feita pelo método `treatedCall`. Neste método estão configurados os parâmetros de tratamento para cenários de falta de conexão com a internet ou com o servidor. Como pode-se ver na figura 2, é feito o tratamento para cenário de Device Offline antes da chamada, e tratamento para cenário de Servidor Offline depois da chamada – ambas realizando o armazenamento da chamada para tentativas futuras. Dessa forma, com a aplicação da biblioteca *OfflineManager* em uma chamada, os erros por falta de conexão com a internet ou com o servidor são tratados durante a chamada e novas tentativas são realizadas automaticamente caso esse tenha sido o tratamento escolhido. Portanto, o resultado final pode ser apresentado ao usuário mesmo em cenários de falta de conexão.

Por fim, a arquitetura da biblioteca implementada pode ser vista de forma simplificada na Figura 3.

`OfflineManager` é a classe que deve ser inicializada para a utilização da biblioteca. Nela está disponível o método `treatedCall`, que deve ser usado para fazer o tratamento das chamadas HTTP. De acordo com os parâmetros passados para este método, ele ar-

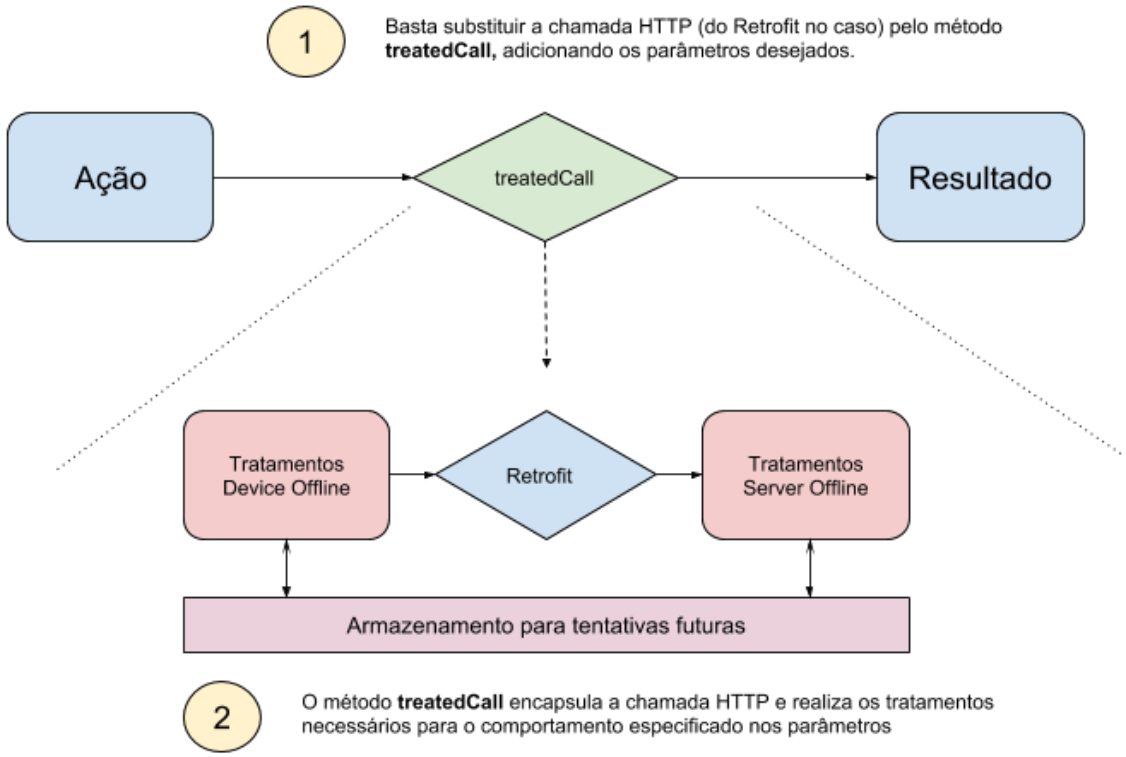


Figura 2 – Fluxo de uma ação com chamada HTTP a um servidor tratada pelo método `treatedCall`

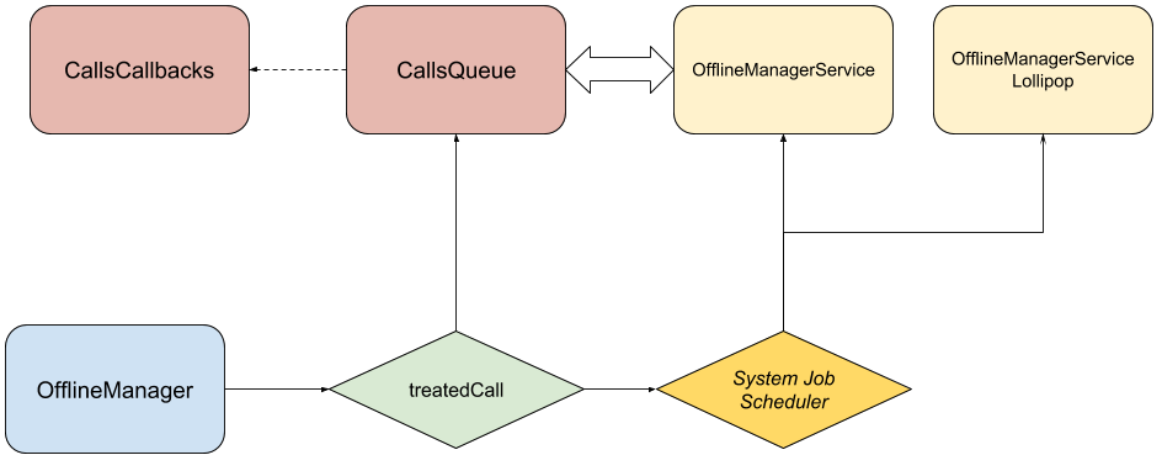


Figura 3 – Diagrama simplificado das classes da biblioteca *OfflineManager*

mazena as chamadas em um `Hash-Map` presente na classe CallsQueue. Esse *Hash-Map* é composto por objetos da classe `CallsCallbacks` – classe que representa uma chamada HTTP, seus aspectos e contexto. De acordo com o tratamento especificado, um serviço (*Job*) é escalonado no sistema para ser executado em um segundo momento. Esse serviço é definido pelas classes `OfflineManagerService` e `OfflineManagerServiceLollipop` que

executam as chamadas armazenadas no momento especificado e adequado.

Essas classes serão explicadas de forma mais detalhada nas próximas seções deste capítulo, e o funcionamento geral da biblioteca e do método `treatedCall` serão melhores detalhados.

3.2 Biblioteca Android

A abordagem adotada para implementação da solução consiste em uma simples biblioteca, que expõe um método para encapsular as chamadas de conexão com a Internet. Seu funcionamento interno seria transparente ao usuário, que apenas precisaria definir os modos de tratamento na assinatura do método, bem como outras características relativas àquele ponto específico de interação com a Internet, para cada chamada a um *back-end* em sua aplicação. Essa foi a abordagem escolhida pois é a abordagem mais simples de ser aplicada e oferece todos os recursos necessários para tratar o problema da forma desejada e proposta neste trabalho de mestrado.

3.3 Plataforma Android

Para o desenvolvimento do projeto, existiam duas reais possibilidades de plataforma para a construção do protótipo para avaliação da proposta. A plataforma Android, e a plataforma iOS, que são as duas plataformas mais utilizadas no cenário de aplicações móveis, no momento da produção deste trabalho de mestrado.

No escopo desse projeto, foi escolhida a plataforma Android por ser um ambiente aberto de desenvolvimento – tanto a IDE de desenvolvimento, quanto os simuladores para teste, são de fácil instalação em qualquer computador. Em contrapartida, para o desenvolvimento na plataforma iOS, seria necessário um computador próprio do fabricante Apple. Isso sem contar outras características menores como por exemplo a facilidade, no Android, em instalar aplicações desenvolvidas em celulares que rodam Android, para poder testar de fato em um dispositivo móvel.

Além disso, esse fator da facilidade em desenvolver, testar e distribuir projetos na plataforma Android, junto com o fato de que, no momento da escrita desta dissertação, a plataforma Android já é a que domina a maior parte do mercado de dispositivos móveis[1], tornam clara a escolha da plataforma para o desenvolvimento do projeto aqui apresentado. Maior experiência com a linguagem Java, que é a linguagem utilizada para o desenvolvimento de aplicativos Android, também foi um fator. Existe porém, uma grande motivação para a construção de uma biblioteca semelhante para a plataforma iOS, uma vez que os

[1] Segundo duas fontes populares de estudo de mercado: IDC (http://www.idc.com/promo/smartphone-market-share/os) e Gartner (http://www.gartner.com/newsroom/id/3609817)

conceitos utilizados como base para esse projeto foram validados e viabilizados, portanto é possível assumir que estender esse projeto também para a plataforma da Apple é algo de valor para a comunidade de desenvolvimento bem como para a comunidade científica.

3.4 Retrofit

Para fornecer usabilidade em modo *offline*, de forma a resolver os pontos críticos citados no Capítulo 1, é preciso que a biblioteca trate de alguma forma a comunicação da aplicação com o servidor, ou com a Internet. Como a maioria das aplicações existentes na plataforma mobile funciona de forma a se comunicar por chamadas HTTP, faz sentido que a biblioteca aqui construída operasse em cima dessas chamadas.

No contexto da plataforma Android, existem algumas bibliotecas para facilitar ao desenvolvedor aplicar comunicação HTTP, para que ele não precise se preocupar com questões de baixo nível e implementar, desde o início, todas as características dessa comunicação. Bibliotecas com esse propósito são também conhecidas como "clientes HTTP". Dentre as diversas bibliotecas existentes de cliente HTTP, uma que é bastante popular é a biblioteca Retrofit[2].

O Retrofit funciona de forma que o desenvolvedor, para implementar a comunicação HTTP, cria uma classe de interface para cada *back-end* a ser chamado, e define uma URL de base para esse *back-end*. Também nessa classe de interface, o desenvolvedor utiliza o Retrofit para definir métodos, que representam as chamadas HTTP para cada funcionalidade desejada, bem como os cabeçalhos (*headers*) dessas chamadas ou dados a serem enviados. Dessa forma, esses métodos são utilizados para a criação de um objeto do tipo `Call`.

Esse objeto `Call`, é um objeto de chamada da biblioteca Retrofit, que pode ser executada de duas maneiras: assíncrona, por meio do método `enqueue` ou síncrona e bloqueante, por meio do método `execute.`

Como o objetivo da biblioteca *OfflineManager* é ser simples e e fácil de ser aplicada e utilizada, é natural que ela fosse construída sobre algum cliente HTTP, assim o desenvolvedor consegue com facilidade tanto implementar a comunicação desejada de seu aplicativo com o respectivo *back-end*, quanto já oferecer funcionalidades em modo *offline*. Para facilitar sua aceitação, foi tomada a decisão de utilizar o Retrofit devido a sua popularidade, pois do ponto de vista técnico não haveria diferença ou dificuldades em implementar sobre outros clientes HTTP.

[2] http://square.github.io/retrofit/

3.5 Componente usado para os *feedbacks*

Para os retornos de fedback do modo verboso, nos deparamos com duas possibilidades diferentes, na plataforma Android: `Toast`[3] e `Snackbar`[4].

O primeiro, chamado `Toast`, é um componente simples que retorna mensagens para o usuário em uma aplicação na forma de um pequeno pop-up na região inferior da tela. Eles desaparecem automaticamente depois de um tempo, e usualmente são utilizados por padrão para mensagens referentes a ações do sistema. Dadas as características desse primeiro mecanismo, no contexto da biblioteca *OfflineManager* alguns pontos são de interesse:

1. Avisos emitidos pelo `Toast` são independentes da tela em primeiro plano da aplicação. Ou seja, a mensagem será exibida independente de qual tela o aplicativo estiver, ou até mesmo se o aplicativo estiver em segundo plano, inclusive com outras aplicações rodando em primeiro plano;

2. Para a mensagem de `Toast` ser construída e exibida, ela não precisa de uma tela ou componente visual da aplicação como referência. Isso permite mais flexibilidade nos pontos em que ele pode ser construído e mostrado. No contexto da biblioteca *OfflineManager*, é possível construir e mostrar mensagens de `Toast` mesmo em classes de serviço, que rodam em *threads* secundárias do sistema.

O segundo, chamado `Snackbar`, é também um componente simples de retorno de mensagens, padrão da plataforma Android. Ele também aparece no final da tela, e também desaparece automaticamente após um curto período de tempo. Porém, se difere do `Toast` no sentido de que o `Snackbar` oferece um layout que indica um pouco mais de dependência do contexto da tela do aplicativo em questão. Por esse motivo, ele é mais utilizado para mensagens de *feedback* cujo contexto seja mais próximo do que o aplicativo mostra no momento. É importante mencionar também que mensagens de `Snackbar` são exibidas apenas na tela da aplicação em que foram chamados. Portanto se a aplicação estiver em outra tela, e a tela em que o `Snackbar` foi chamado continuar na pilha de telas, mas em segundo plano, ele será mostrado somente quando sua tela vier a primeiro plano. O mesmo acontece para o caso da aplicação como um todo ter ido para segundo plano. Assim, os pontos a serem ressaltados do mecanismo `Snackbar` são:

1. Oferece um *feedback* mais contextual, e pela experiência tradicional da plataforma Android, mais amigável no contexto de uma aplicação;

2. Não traz a conotação de mensagens relacionadas a ações do sistema;

[3] https://developer.android.com/guide/topics/ui/notifiers/toasts.html
[4] https://developer.android.com/reference/android/support/design/widget/Snackbar.html

3. É dependente do contexto e da tela da aplicação em questão, portanto não será exibido a menos que a tela em que foi chamado volte para o primeiro plano da aplicação; e

4. Para sua construção, é necessário associá-lo a uma tela de referência. Isso pode causar exceções em tempo de execução, caso seja construído em *threads* secundárias e chamado em uma tela que já foi destruída, ou que não possui mais referência na aplicação. Por esse motivo, não é um mecanismo viável para mensagens de *feedback* construídas nas classes de serviço, quando as chamadas são executadas em segundo plano.

Feitas as considerações sobre os possíveis mecanismos de *feedback* citados acima, a biblioteca utiliza uma combinação deles. Para mensagens que serão retornadas imediatamente ao usuário, no contexto da tela em exibição da aplicação, é utilizado o `Snackbar`. Para mensagens de *feecback* a serem retornadas após a completude de alguma execução feita em segundo plano, é utilizado o `Toast`.

3.6 Método `treatedCall` para tratamento das chamadas

Para possibilitar o tratamento oferecido pela biblioteca *OfflineManager*, a mesma expõe um método para utilização do desenvolvedor. Este método chama-se `treatedCall` e é utilizado para cada chamada do Retrofit onde pretende-se adicionar comportamento para utilização em modo *offline*. Os parâmetros desse método, cuja assinatura é apresentada na Listagem 3.1, são detalhados a seguir.

```
1  public <T> void treatedCall(
2      final Call<T> objCall,
3      final DeviceOfflineTreatment deviceOfflineTreatment,
4      final ServerOfflineTreatment serverOffTreatment,
5      final int retries,
6      final boolean verbose,
7      final CustomCallbackSuccess onSuccess,
8      final CustomCallbackFail onFail)
9      {
10         //...
11     }
```

Listagem 3.1 – Listagem que apresenta a assinatura do método `treatedCall`

```
public <T> void treatedCall( final Call<T> objCall,
                             final DeviceOfflineTreatment deviceOfflineTreatment,
                             final ServerOfflineTreatment serverOffTreatment,
                             final int retries,
                             final boolean verbose,
                             final CustomCallbackSuccess onSuccess,
                             final CustomCallbackFail onFail )
```

Figura 4 – Parâmetro destacado é onde deve ser passada a chamada do Retrofit a ser tratada

3.6.1 Chamada do Retrofit a ser tratada

O primeiro parâmetro a ser passado ao método `treatedCall`, destacado na Figura 4, é uma chamada do Retrofit, do tipo `Call`, onde deseja-se adicionar os tratamentos para funcionamento *offline*. O tratamento oferecido pela biblioteca ocorre de maneira refinada para cada chamada específica do Retrofit, portanto é por meio desse parâmetro onde essa chamada é definida.

Essa chamada representa uma chamada HTTP contendo o endereço do *back-end*, variáveis de cabeçalho dessa chamada HTTP e se houver, dados a serem enviados, como foi explicado na seção 3.4.

3.6.2 Modos de tratamento para dispositivo *offline*

```
public <T> void treatedCall( final Call<T> objCall,
                             final DeviceOfflineTreatment deviceOfflineTreatment,
                             final ServerOfflineTreatment serverOffTreatment,
                             final int retries,
                             final boolean verbose,
                             final CustomCallbackSuccess onSuccess,
                             final CustomCallbackFail onFail )
```

Figura 5 – Parâmetro destacado define o modo de tratamento para cenário de dispositivo *offline*

O segundo parâmetro da chamada `treatedCall`, representado na Figura 5, especifica qual será o tratamento da chamada para o caso do dispositivo estar sem conexão com a Internet. Para esse parâmetro, o desenvolvedor pode escolher um entre os três tratamentos disponibilizados, cada um com suas características específicas e cenários para o qual foi pensada.

3.6.2.1 Modo *Enforce*

O modo *Enforce* (obrigatório), quando atribuído ao tratamento de uma chamada, funciona de forma a não permitir nenhuma execução caso o dispositivo esteja sem conexão com a Internet. Ele trata os casos de chamadas que são sensíveis ao contexto e ao momento,

e que não devem permitir nenhum tipo de uso em modo *offline*. Apesar da biblioteca *OfflineManager* existir para permitir certo grau de funcionalidade mesmo com dispositivo sem conexão, alguns cenários não devem permitir tal flexibilidade. Um exemplo disso são aplicações bancárias: algumas chamadas de movimentação ou transações, necessárias para a execução de outras funções, só poderão ser executadas com a presença de conexão, tanto por questões de segurança e garantia de execução, quanto por sensibilidade ao momento. Um usuário provavelmente não irá pagar uma conta no aplicativo, se uma transferência feita antes não obtiver sucesso.

3.6.2.2 Modo *Flexible*

O modo *Flexible* (flexível) foi pensado para chamadas que apresentam possibilidade de serem executadas em segundo plano, com tratamento de funcionamento *offline*, mas ainda representam chamadas mais importantes no contexto do aplicativo. Isso faz com que seja interessante oferecer controle sobre o tratamento *offline* dessas chamadas ao usuário. Pode ser que dependendo do contexto de utilização e dos dados que se referem àquela chamada, o usuário deseje ou não prosseguir com o tratamento *offline* daquela ação. Assim, interações tratadas com o modo *Flexible*, caso o dispositivo esteja sem conexão com a Internet, apresentam um pop-up de confirmação para o usuário, cujo objetivo é: informar ao usuário que o dispositivo está sem conexão, e questionar se ele deseja que a ação seja armazenada e executada em um segundo momento automaticamente, quando houver conexão disponível. Um exemplo prático de chamadas possíveis de serem tratadas com o modo *Flexible* são preenchimento de formulários ou consultas de dados na aplicação. Dependendo do contexto do usuário, ele pode desejar que o formulário seja enviado em um segundo momento, automaticamente, caso seu dispositivo esteja desconectado. Porém, ele pode escolher não prosseguir com a submissão automática, se achar que pode mudar de ideia.

Apesar do exemplo citado onde o modo *Flexible* pode ser utilizado, é possível perceber que não existe um contexto específico para a utilização desse modo de tratamento. Na verdade ele é apenas uma versão de tratamento para uso em modo *offline* que oferece interação com o usuário e mais *feedbacks*. Assim, cabe ao desenvolvedor escolhê-lo para tratar chamadas, dependendo da experiência de usuário desejada para seu aplicativo.

3.6.2.3 Modo *Transparent*

Por fim, o modo *Transparent* (transparente) se opõe ao tratamento oferecido pelo modo *Flexible* no sentido de que foi pensado para se utilizar em chamadas onde deseja-se uma execução silenciosa e transparente. Na maioria das aplicações, existem diversos pontos de interação com o servidor que não são mandatórios para a execução de fluxos da aplicação. Para tratar essas chamadas, é necessário que haja um mecanismo que aja

de maneira silenciosa.

Assim, o modo *Transparent* armazena a chamada em que foi especificado, e a executa em segundo plano de maneira automática e transparente ao usuário. Sendo transparente, ele não requisita nenhum tipo extra de interação. Porém o desenvolvedor ainda pode optar por ativar o modo *verbose*, que também é um parâmetro e será detalhado posteriormente, para essa chamada. Se o fizer, mesmo sendo um tratamento transparente, *feedbacks* serão oferecidos na completude das chamadas armazenadas. Assim é possível configurar de maneira refinada a experiência de usuário desejada.

Um outro exemplo prático da utilização do modo *Transparent* são aplicações sociais, ou de troca de mensagens: o usuário pode enviar uma postagem, foto ou mensagem, a qualquer momento e sem se preocupar com o estado de conectividade. De forma transparente, a aplicação deve tratar o envio desses dados caso não houver conexão no momento do envio. Para o usuário, é indiferente qual o momento exato em que o envio ocorre.

3.6.3 Modos de tratamento para servidor indisponível

```
public <T> void treatedCall( final Call<T> objCall,
                             final DeviceOfflineTreatment deviceOfflineTreatment,
                             final ServerOfflineTreatment serverOffTreatment,
                             final int retries,
                             final boolean verbose,
                             final CustomCallbackSuccess onSuccess,
                             final CustomCallbackFail onFail )
```

Figura 6 – Parâmetro destacado define o modo de tratamento para cenário de servidor indisponível

O terceiro parâmetro da chamada, ilustrado na Figura 6, trata-se do modo como a chamada será tratada em caso de servidor indisponível. A biblioteca trata o caso de servidor indisponível de formas semelhantes ao caso de dispositivo *offline*, porém em conjunto com o quarto parâmetro, que é o número de tentativas. A ideia central é que a biblioteca aplique o tratamento escolhido dentre os três tipos disponíveis, e realize tentativas de acordo com o número especificado pelo desenvolvedor. Isso para possibilitar que, nesse meio tempo, o servidor volte a ficar disponível, cenário onde umas das novas tentativas em segundo plano será bem sucedida, sem que o usuário tenha que realizar a ação novamente. Os três tipos de tratamento disponibilizados foram construídos de acordo com diferentes tipos de contexto em mente, e também envolvem níveis de interação com o usuário diferentes.

3.6.3.1 Modo *NoAction*

O modo *NoAction* (sem ação) trata-se de um modo mais seguro, da mesma forma que o *Enforce*, para dispositivo *offline*. É um modo que deve ser utilizado para chamadas

onde o contexto e o momento da realização da chamada é importante e não interessa para a lógica da aplicação, neste momento, armazenar a chamada para realizar novas tentativas caso o servidor estiver indisponível. Dessa forma, o modo *NoAction* não realizará nenhum tipo de tratamento ou novas tentativas. O mesmo exemplo pode ser utilizado aqui: aplicações bancárias, ou que necessitam de maior segurança ou garantia de dados, utilizarão esse tratamento nas chamadas mais sensíveis da aplicação.

3.6.3.2 Modo Flexible

Já o modo *Flexible* (flexível) trata-se, como o próprio nome já diz, de uma opção mais flexível para uma chamada na aplicação. Caso o servidor estiver indisponível, ao invés de a aplicação retornar erro e impedir o usuário de continuar, esse modo oferece uma interação que requisita confirmação do usuário para saber se ele deseja armazenar essa chamada para ser tentada novamente. Esse ponto de interação é importante para fornecer controle ao usuário sobre o tratamento em modo *offline*, pois entende-se que algumas chamadas podem ou não ser tratadas de acordo com o contexto de utilização no momento da indisponibilidade do servidor. Assim, haverão momentos onde o usuário aceitará essa confirmação, tratando a chamada realizada, e momentos em que para ele não é interessante que o tratamento ocorra. É importante ressaltar que a principal diferença entre o modo flexível e o modo transparente é esse grau mais elevado de interação com o usuário. Ambos os modos tratam a chamada de forma a armazená-la e realizar novas tentativas depois do tempo de intervalo definido e de acordo com o número de tentativas restantes. Porém somente no modo flexível tem-se o controle sobre o tratamento passado para o usuário. Esse ponto de interação pode ser valioso em diversos contextos de aplicação, fazendo com que a aplicação apresente melhor usabilidade. Exemplos de aplicação que justifiquem esse modo são inúmeras, pois entende-se que em qualquer tela onde haja um carregamento de dados importante, ou até alguma chamada que envie dados para um servidor, é interessante o uso do modo flexível, desde que o pop-up de confirmação não seja um fator que piore a usabilidade.

3.6.3.3 Modo Transparent

Por fim o modo *Transparent* (transparente) se opõe do modo flexível para tratar os casos restantes de interação. Com este modo de tratamento, são abordados o restante dos casos, onde entende-se que não exista necessidade de interação ou confirmação do usuário. Na verdade, trata os casos onde esse tipo de interação acaba por interromper o fluxo natural da aplicação e piorar a usabilidade. Por isso, é importante que a biblioteca ofereça um modo de tratamento onde as ações ocorram de forma transparente ao usuário. Esse modo deve ser utilizado para chamadas secundárias nas telas da aplicação, e que o usuário não se preocupa de fato com o tratamento que está sendo feito. Ele apenas ocorre em segundo plano, possivelmente disponibilizando as informações em um segundo

momento, caso novas tentativas tenham ocorrido com sucesso. Os exemplos de contexto para a utilização desse modo estão diretamente relacionados com o contexto da chamada e da aplicação. Da mesma forma que o tratamento transparente para quando o dispositivo está *offline*, aqui é possível citar aplicações sociais: o usuário pode realizar interações na forma de publicações ou envio de mensagens, independente do estado de disponibilidade do servidor, sem precisar se preocupar com o que está acontecendo no fluxo da aplicação. Assim, a biblioteca trata as chamadas realizando novas tentativas. Caso tenha sucesso, a interação será transparente ao usuário, que continuará utilizando a aplicação em um segundo momento. Caso haja falha, o usuário será avisado que a ação não foi possível.

3.6.4 Número de tentativas

```
public <T> void treatedCall( final Call<T> objCall,
                             final DeviceOfflineTreatment deviceOfflineTreatment,
                             final ServerOfflineTreatment serverOffTreatment,
                             final int retries,
                             final boolean verbose,
                             final CustomCallbackSuccess onSuccess,
                             final CustomCallbackFail onFail )
```

Figura 7 – Parâmetro destacado é onde se define o número de tentativas a serem executadas para tratamento de servidor indisponível

O quarto parâmetro a ser passado no método `treatedCall`, destacado na Figura 7, é um número inteiro que representa o número de tentativas a serem executadas caso o servidor esteja indisponível. Na primeira chamada da requisição encapsulada são feitas as verificações de conexão com a Internet. Caso haja conexão, a chamada é realizada, e pelo retorno, é avaliado se há conexão com o servidor. Se não houver, é aplicado o tratamento escolhido pelo desenvolvedor para servidor indisponível (terceiro parâmetro). Se o modo for qualquer um diferente de *NoAction*, que não executa nenhum tratamento, a chamada é armazenada e será tentada novamente em segundo plano. Essas novas tentativas são realizadas por uma classe de serviço, e para essas novas tentativas não há necessidade uma nova verificação de conexão com a Internet pois, além do tempo de intervalo definido para suas execuções, existe a condição do serviço somente ser disparado caso haja conexão com a Internet.

Dessa forma, esse parâmetro representa o número de novas tentativas a serem executadas por essa classe de serviço em segundo plano, caso o servidor continue indisponível. Se durante essas novas tentativas o servidor voltar a ficar disponível, uma das tentativas vai retornar uma chamada com sucesso e a biblioteca continua com o comportamento esperado, chamando os *callbacks* de retorno, e cancelando possíveis novas tentativas. O intervalo de tempo entre as tentativas é definido no construtor da classe `OfflineManager`

e é definido de modo geral para uma instância do gerenciador, como explicado na seção 3.7.

3.6.5 Modo verboso

```
public <T> void treatedCall( final Call<T> objCall,
                             final DeviceOfflineTreatment deviceOfflineTreatment,
                             final ServerOfflineTreatment serverOffTreatment,
                             final int retries,
                             final boolean verbose,
                             final CustomCallbackSuccess onSuccess,
                             final CustomCallbackFail onFail )
```

Figura 8 – Parâmetro destacado é onde se define o número de tentativas a serem executadas para tratamento de servidor indisponível

Uma das heurísticas de usabilidade diz que o usuário deve sempre saber o que está acontecendo na aplicação, e sempre ser informado sobre estados ou ações (INOSTROZA et al., 2013). O modo *verbose* (verboso), destacado na Figura 8, é o quinto parâmetro do método `treatedCall`. Foi pensado para esse cenário de sempre enviar mensagens de *feedback* para o usuário para ele sempre estar ciente das ações que estão sendo executadas, mesmo que em segundo plano, pela biblioteca. Ele pode ser definido como verdadeiro ou falso pois o desenvolvedor pode entender que alguns dos tratamentos aplicados não precisam desses *feedbacks* contantes, em um cenário que a ação não é de impacto e deve acontecer de forma transparente.

Importante mencionar que o modo verboso retorna apenas mensagens de *feedback* sobre ações que ocorreram ou que poderão ocorrer. Ele não interage de forma alguma requisitando ações do usuário. Para isso, existe o modo *Flexible* para tratamento de dispositivo sem internet ou servidor indisponível, explicado anteriormente. O modo *Flexible* que traz esse contexto de uma confirmação do usuário antes que alguma ação para funcionamento *offline* seja executada. Portanto, é possível existir uma combinação do modo flexível, que interage com o usuário por meio de confirmação, e do modo verboso, que retorna mensagens de *feedback* sempre informando quais ações foram tomadas e também informando no momento que ações são completadas.

Todas as mensagens de retorno para o usuário, incluindo as mensagens mostradas quando o modo verboso é verdadeiro, podem ser sobrescritas pelo desenvolvedor, durante a construção do aplicativo. A biblioteca *OfflineManager* já vem com um conjunto padrão de mensagens em seu arquivo `strings.xml` (arquivo onde são definidos os textos mostrados em um aplicativo), todas em inglês, que basicamente dizem o que aconteceu e em qual momento. É necessário, porém, que o desenvolvedor sobrescreva essas mensagens pré-definidas tanto para fins de internacionalização, bem como para aderir os *feedbacks* da biblioteca no contexto de sua aplicação.

3.6.6 Funções de callback

```
public <T> void treatedCall( final Call<T> objCall,
                             final DeviceOfflineTreatment deviceOfflineTreatment,
                             final ServerOfflineTreatment serverOffTreatment,
                             final int retries,
                             final boolean verbose,
                             final CustomCallbackSuccess onSuccess,
                             final CustomCallbackFail onFail )
```

Figura 9 – Parâmetro destacado é onde se define o número de tentativas a serem executadas para tratamento de servidor indisponível

Por fim, os últimos dois parâmetros necessários do método `treatedCall`, ilustrados na Figura 9, são dois métodos de *callback*. Isso se dá porque, por padrão, a biblioteca do Retrofit oferece dois *callbacks* para o retorno da execução de uma chamada ao *back-end*: um para o caso de a chamada ter conseguido se conectar ao *back-end* e executar a chamada, e outro para o caso de falha neste processo. Note-se que o retorno para o primeiro caso não significa de fato que o retorno obteve sucesso – pode ser que o *back-end* tenha retornado algum código HTTP de falha, como por exemplo erro 401[5], ou seja, não houve problema na conexão, mas não foi uma operação válida no servidor. Por isso, dentro deste *callback* o desenvolvedor ainda precisa verificar se o retorno obteve sucesso com um código HTTP de sucesso, o que é encapsulado pela verificação do Retrofit `isSucessful()`. Dados esses detalhes, é de suma importância que o desenvolvedor, mesmo encapsulando a chamada ao *back-end* com o método oferecido pelo *OfflineManager*, que trata as questões de funcionamento *offline*, consiga ainda tratar o retorno da sua chamada, executando suas regras e lógicas de negócio. Portanto, a biblioteca *OfflineManager* oferece esses dois *callbacks*, que são chamados pelos retornos das chamadas ao Retrofit, que é executada internamente. Assim, o desenvolvedor consegue tratar essa interface da mesma forma como trataria os *callbacks* da chamada diretamente executada pelo Retrofit.

3.7 Funcionamento e utilização da biblioteca

Foram apresentadas as decisões de projeto para a construção da abordagem, que foi decidido ser uma biblioteca, construída na plataforma Android, e utilizando a biblioteca Retrofit como cliente HTTP. Foi mostrado também o método que é exposto para sua utilização, de forma a encapsular as chamadas do Retrofit e fornecer tratamento de uso em modo *offline*, baseados nos parâmetros utilizados. Nesta seção são apresentadas as características de implementação da biblioteca e as estruturas internas utilizadas.

[5] Erro 401 - Não autorizado: erro retornado pela chamada HTTP quando a chamada e os dados enviados pelo cliente estão corretos, porém o servidor não conseguiu autenticá-lo. Isso ocorre porque a autenticação não foi fornecida, ou porque não passou nos testes de autenticação realizados pelo servidor.

A biblioteca *OfflineManager* é composta por cinco classes. A primeira delas e a mais importante é a classe chamada `OfflineManager`, pois é onde é definido o método `treatedCall`, que é o método exposto para os desenvolvedores. É o método que encapsula as chamadas e aplica os tratamentos definidos, dentre as possibilidades de tratamento explicadas na seção 3.6.

Para sua utilização, o desenvolvedor deve inicializar essa classe passando como parâmetros para o construtor uma *view* de referência e o contexto atual da atividade onde a chamada será tratada. Atividades são componentes de uma aplicação que fornecem uma interface (tela) para o usuário interagir [6].

Esses parâmetros são necessários pois são utilizados para que as mensagens de *feedback* possam ser criadas e exibidas pela biblioteca. No construtor, o desenvolvedor pode também opcionalmente sobrescrever o intervalo de tempo definido para novas tentativas, no caso dos tratamentos para quando o servidor estiver indisponível. Esse valor é definido de forma geral no próprio construtor pois, na maioria dos casos, a aplicação utiliza conexão com apenas um *back-end*, e o intervalo de novas tentativas esta relacionado com o servidor da aplicação, pois serve para definir em quanto tempo a biblioteca deve tentar executar a chamada novamente, até que o servidor esteja disponível novamente.

Dessa forma, o tempo de intervalo se relaciona ao contexto do servidor utilizado, e suas características ou até estatísticas de inatividade, e não ao de chamadas específicas. Também, para a utilização, a classe `OfflineManager` deve ser inicializada a cada atividade onde são feitas chamadas, portanto é possível que em partes diferentes da aplicação, sejam definidos tempos diferentes de intervalo para novas tentativas no caso do servidor estar indisponível.

A Listagem 3.2 mostra a inicialização da biblioteca. Na linha 2, a variável `mManager` representa um objeto do tipo `OfflineManager` declarado no escopo da atividade. Ainda na linha 2, é passado o contexto da atividade, por meio do `this`, e uma *view* de referência: `mGetQuoteBtn`, que neste caso representa um botão.[7] Na linha 3 é definido um intervalo de 3 segundos para novas tentativas, e na linha 4 é chamado o método `build()` para efetivar a inicialização.

```
1    //inicializacão offlineManager
2    mManager = new OfflineManager.Builder(this, mGetQuoteBtn)
3        .maxTimeoutSeconds(3)
4        .build();
```

Listagem 3.2 – Trecho de código referente a inicialização da biblioteca *OfflineManager*

[6] https://developer.android.com/guide/components/activities.html?hl=pt-br

[7] Componentes visuais em Android, por exemplo botões, herdam de um componente genérico *view*. Por isso é possível passar um botão como *view* de referência.

A classe `OfflineManager` é onde é definido o método `treatedCall`. Este é o método que os desenvolvedores utilizarão em suas aplicações para encapsular as chamadas do Retrofit. Assim, o primeiro parâmetro é uma chamada já construída do Retrofit. Essa chamada nada mais é do que um objeto que representa uma chamada HTTP contendo o endereço do *back-end*, variáveis de cabeçalho dessa chamada HTTP e se houver, um corpo de dados a ser enviado. A Listagem 3.3 mostra como essa chamada é criada.

```
1    //criação de uma chamada no retrofit
2    Call getQuoteCall = mRetrofitApi.getQuote();
```

Listagem 3.3 – Trecho de código referente a criação de uma chamada do Retrofit

Na linha 2, define-se a variável da chamada, denominada `getQuoteCall`, e em seguida utiliza-se um método definido na API do Retrofit para atribuir o comportamento dessa chamada. De forma mais detalhada, `mRetrofitApi` na linha 2 representa uma interface Retrofit definida pelo desenvolvedor, já inicializada previamente. Dentro dessa interface, o desenvolvedor cria métodos que são associados com as chamadas HTTP a serem realizadas. Neste caso, o método `getQuote()` executado pela API atribui a chamada HTTP desejada para a `Call getQuoteCall`.

```
1    //usando o enqueue diretamente
2    getQuoteCall.enqueue(new Callback<QuoteResult>() {
3        @Override
4        public void onResponse(Call<QuoteResult> call, Response<
   QuoteResult> response) {
5            //tratar onResponse ...
6            if(response.isSuccessful()){
7                //tratar sucesso ...
8            } else {
9                //tratar falha ...
10           }
11       }
12
13       @Override
14       public void onFailure(Call<QuoteResult> call, Throwable t
   ) {
15           //tratar onFailure ...
16       }
17   });
```

Listagem 3.4 – Trecho de código referente a execução assíncrona de uma chamada diretamente pela API do Retrofit

A Listagem 3.4 mostra a execução direta dessa chamada, sem o tratamento da biblioteca *OfflineManager*. Para executar a chamada basta-se utilizar o método `enqueue` (linha 2), assíncrono, que escalona essa chamada para ser realizada outra *thread*, retornando dois *callbacks*: `onResponse` (linha 4) que é executado se a chamada ter conseguido se conectar ao *back-end*, ou `onFailure` (linha 14) que é executado caso uma conexão não tenha sido possível. Dessa forma os possíveis retornos de sucesso dessa chamada devem ser tratados pelo desenvolvedor por meio dos códigos de retorno HTTP, e a lógica da aplicação implementada dentro dessas condições. É possível que o desenvolvedor queira utilizar um mecanismo síncrono para executar essa chamada na *thread* principal. Para isso ele poderia utilizar o método `execute` no lugar do `enqueue`, na chamada. Porém essa última forma não é muito utilizada por ser bloqueante.

```
1    //chamada tratada pela biblioteca OfflineManager
2    mManager.treatedCall(
3            getQuoteCall,
4            OfflineManager.DeviceOfflineTreatment.Flexible,
5            OfflineManager.ServerOfflineTreatment.Flexible,
6            5,
7            true,
8            new OfflineManager.CustomCallbackSuccess() {
9                @Override
10               public void responseCallback(Call call, Response
     response) {
11                   //tratar onResponse ...
12                   if(response.isSuccessful()){
13                       //sucesso ...
14                   } else {
15                      //retorno HTTP de falha ...
16                   }
17               }
18           },
19           new OfflineManager.CustomCallbackFail() {
20               @Override
21               public void failCallback(Call call, Throwable t)
     {
22                   //tratar onFailure ...
23               }
24           }
25    );
```

Listagem 3.5 – Trecho de código referente a execução de uma chamada tratada pela biblioteca *OfflineManager* pelo método `treatedCall`

Em comparação com a execução direta da chamada pelo Retrofit, o método `treatedCall` executa essa mesma chamada, da forma mostrada na Listagem 3.5. Pode-se ver nessa listagem que o desenvolvedor chama o método `treatedCall` (linha 2) em uma instancia da biblioteca *OfflineManager* inicializada como `mManager` (linha 2). Em seguida, passa a chamada do Retrofit `getQuoteCall` (linha 3), o modo flexível de tratamento para dispositivo *offline* (linha 4), o modo flexível de tratamento para servidor indisponível (linha 5), o número de tentativas (linha 6), o modo verboso como verdadeiro (linha 7) e por fim, na linha 8 implementa o *callback* para sucesso de conexão com o *back-end*, e na linha 19 implementa o *callback* para falha na chamada. Esses dois *callbacks* representam os mesmos retornados pelo Retrofit: `onResponse` e `onFailure` respectivamente.

Outras duas classes que compõe a biblioteca são as classes de serviço que implementam os serviços a serem executados em segundo plano, quando certas condições forem atingidas, como detecção de conexão ou intervalo de tempo esgotado. Essas classes de serviço que são responsáveis por recuperar a chamada armazenada e a executá-las novamente quando a Internet estiver disponível, ou quando o intervalo definido chegar ao fim, na forma de uma nova tentativa. Essas duas classes de serviço, na verdade representam a mesma funcionalidade. Porém, duas classes são necessárias pois uma é utilizada quando a versão do Android utilizada no dispositivo da aplicação é a versão 5, e outra é utilizada quando a versão é a 6 ou maior.

Essas duas classes distintas para versões diferentes do Android são necessárias pois o modo como o Android escalona as classes de serviço sofreu mudanças da versão 5 para a versão 6 e posteriores, alterando a forma como essas classes de serviço são implementadas. É por um motivo relacionado que a utilização da biblioteca *OfflineManager* exige versão 5 ou superior da aplicação que a utilizará. Versões anteriores à versão 5 não implementam os mesmos mecanismos de serviços e escalonamentos de sistema, impossibilitando a implementação e utilização dessas classes de serviço para execução das chamadas em segundo plano de forma adequada ao funcionamento da biblioteca *OfflineManager* (por exemplo, visando otimização da utilização de recursos como bateria e processamento). Dessa forma, para a aplicação da biblioteca *OfflineManager*, é necessário que o projeto utilize a versão 5 ou superior do Android.

```
1     class CallsCallbacks {
2
3         private Call call;
4         private OfflineManager.CustomCallbackSuccess
    callbackSuccess;
5         private OfflineManager.CustomCallbackFail callbackFail;
6         private Context context;
7         private View view;
```

```
8         private OfflineManager.DeviceOfflineTreatment
    deviceOfflineTreatment;
9         private OfflineManager.ServerOfflineTreatment
    serverOfflineTreatment;
10        private boolean verbose;
11        private int retries;
12
13        //...
```

Listagem 3.6 – Trecho de código referente a definição da classe `CallsCallbacks`

Existe também a classe `CallsCallbacks`, cuja definição é mostrada na listagem 3.6: essa entidade representa uma estrutura para armazenar dados referentes à chamada que foi armazenada. Isso para que quando a chamada for executada novamente pela classe de serviço, em segundo plano, seja possível recuperar os dados: chamada a ser executada (linha 3), *callbacks* de sucesso e falha (linhas 4 e 5), o contexto da aplicação (linha 6), a *view* de referência (linha 7), os modos de tratamento para caso de dispositivo *offline* e de servidor indisponível (linhas 8 e 9), o modo verboso (linha 10) e o número de tentativas para àquela chamada (linha 11).

Todos esses dados formam um objeto da classe `CallsCallbacks`. Cada chamada tratada pelo método `treatedCall` produz um objeto com esses dados, para que eles possam ser recuperados quando a chamada for executada em um segundo momento. Esses objetos são armazenados em uma estrutura de `hash-map`, definida na classe `CallsQueue`. Esse *hash-map* armazena como chave o identificador do serviço que irá executar posteriormente a chamada armazenada, e armazena para esse identificador um objeto `CallsCallbacks`, que guarda todos os dados referentes a essa chamada.

Com essas classes, o fluxo em que o tratamento de uma chamada segue pode ser definido pela Figura 10, que apresenta a primeira parte do tratamento.

Como pode ser observado na Figura 10, quando uma chamada ao `treatCall` é feita, primeiro é feito uma verificação ao estado da conexão, para que os tratamentos para o caso de dispositivo *offline* sejam tratados. Pode-se observar que para o modo *Flexible*, existe um passo adicional que é o pop-up de interação para a confirmação do usuário. Observa-se também o modo *Enforce*, que em falta de conexão, encerra a chamada.

As chamadas tratadas e armazenadas pelos modos de tratamento de dispositivo *offline*, são realizadas novamente em um segundo momento quando a Internet estiver disponível. Neste caso, como pode ser visto, a chamada é executada e uma verificação ocorre para saber se o servidor está disponível. Caso esteja, a chamada é executada, e se o modo verboso for utilizado, a mensagem M4 é apresentada. Caso não esteja disponível, a chamada passa para o tratamento em caso de servidor indisponível.

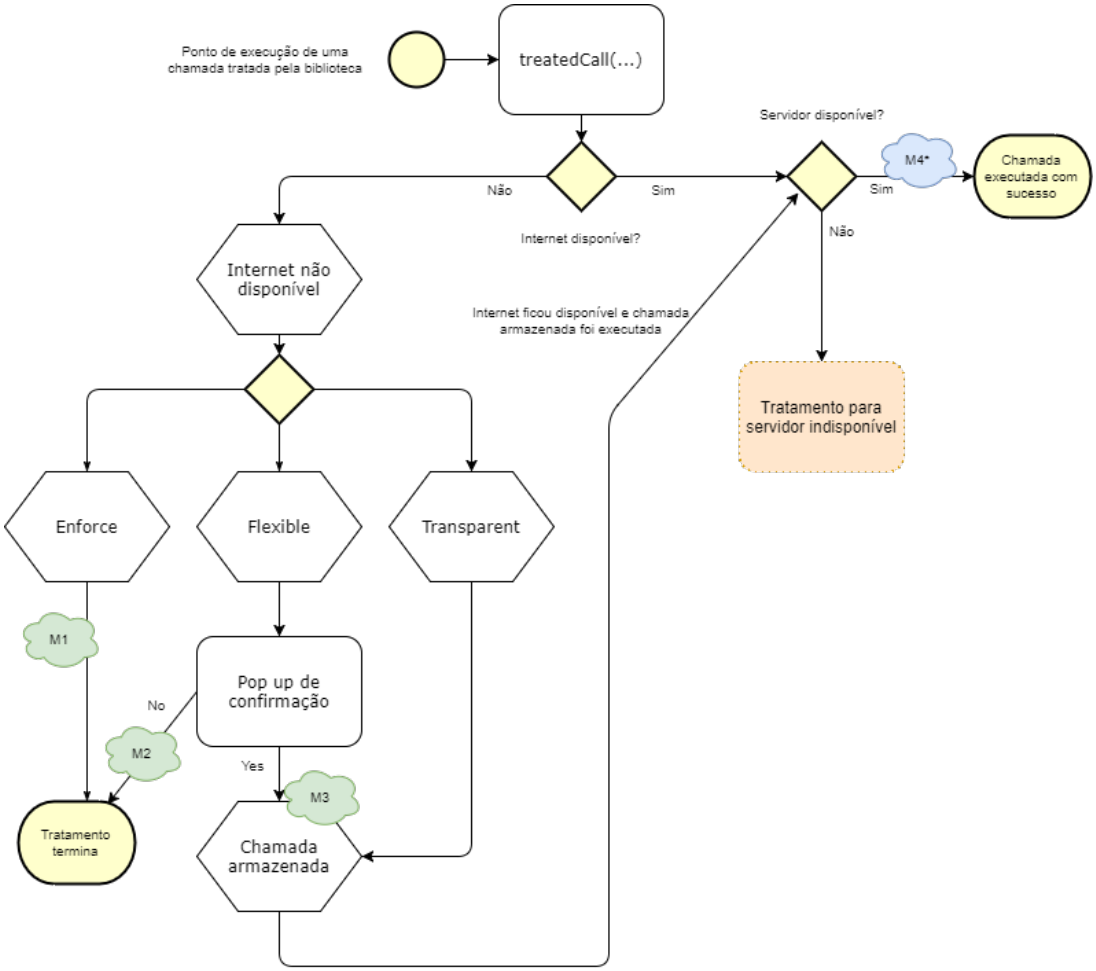


Figura 10 – Fluxograma que apresenta a primeira parte do tratamento de uma chamada, referente a conexão com a Internet no dispositivo.

Os marcadores M1, M2, M3 e M4 representam os pontos onde mensagens de *feedback* são enviadas ao usuário, e são:

- M1: Conexão com a Internet não disponível. Tentar novamente mais tarde.
- M2: Tratamento não realizado.
- M3: Tratamento realizado, ação será realizada assim que a internet estiver disponível.
- M4: Conexão disponível, ação realizada.

É importante ressaltar que a mensagem M1 é enviada independente do modo verboso ativado ou não. As mensagens M2, M3 e M4 são enviadas somente se o modo verboso estiver ativado. Sendo que a mensagem M4, só é emitida se houve tratamento e o sucesso da chamada aconteceu em segundo plano. Se a chamada for executada e o fluxo tiver sucesso pois o dispositivo possuía conexão com a internet, ela não é exibida.

Como mostrado, a segunda etapa de tratamento verifica a disponibilidade do servidor. Essa etapa acontece independente se a chamada foi tratada por falta de conexão com a Internet, ou se foi realizada normalmente com a presença de conexão.

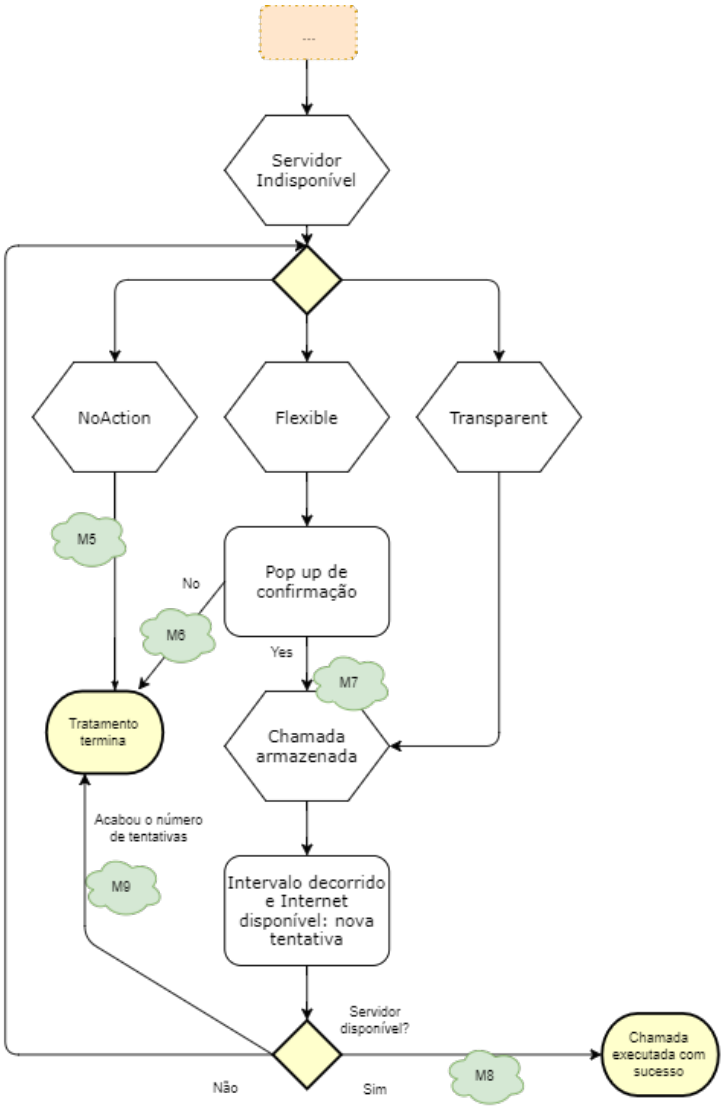


Figura 11 – Fluxograma que apresenta a segunda parte do tratamento de uma chamada, referente a disponibilidade ou não do servidor de *back-end*.

Nesse momento, caso haja falha ao conectar com o servidor, os tratamentos para o caso de servidor *offline* são aplicados. Como pode ser observado, o fluxograma apresentado na Figura 11 mostra os tratamentos possíveis para caso de servidor indisponível. Mostra também, algumas etapas adicionais desse tratamento, como a parte do tempo de intervalo para uma nova tentativa. Quando esse intervalo termina, uma nova tentativa é executada, e a verificação é feita novamente.

Dessa forma, se o servidor estiver disponível, a chamada realizada será bem sucedida. Caso contrário, existe uma verificação sobre o número de tentativas remanescentes. Caso o número de tentativas acabe, o tratamento termina.

Na Figura 11, pode-se observar também os pontos onde *feedbacks* são retornados para o usuário. Os marcadores M5, M6, M7, M8 e M9 representam as mensagens retornadas para os respectivos fluxos, e essas são mostradas apenas caso o modo verboso seja utilizado. As mensagens são:

- M5: Servidor não disponível. Tentar novamente mais tarde.
- M6: Tratamento não realizado.
- M7: Tratamento realizado. Depois de algum tempo essa ação será tentada novamente.
- M8: Conexão disponível, ação realizada.
- M9: Última tentativa realizada e o servidor continua indisponível. Tentar novamente mais tarde.

Outro ponto importante é que as mensagens disponibilizadas na biblioteca estão em inglês, para que a biblioteca seja compreensível para um público maior de desenvolvedores. Porém, nesta dissertação elas são descritas em sua forma traduzida. O texto para as mensagens é genérico, e como pode-se observar, não se relaciona a nenhum contexto de aplicação. Isso porque a adaptação das mensagens para textos adequados fica por conta do desenvolvedor – as mensagens e textos introduzidos pela biblioteca seguem o padrão de internacionalização do Android.

Quando a biblioteca trata uma chamada, entende-se que essa chamada pode se tornar indisponível por dois motivos: o dispositivo do usuário encontra-se sem conexão com a Internet, impedindo a realização de comunicação com o servidor e com a rede, ou o servidor utilizado pela aplicação pode estar indisponível. Por isso, a biblioteca foi pensada para que, quando o desenvolvedor encapsular uma chamada, ela seja tratada de acordo com essas duas características. Isso simplifica sua utilização, pois o desenvolvedor não precisa tratar essas duas situações de erro separadamente, uma vez que do ponto de vista do usuário, esses erros resultam em uma mesma experiência, impedindo a continuidade de uso da aplicação.

3.8 Considerações finais

Neste capítulo foi mostrado como a biblioteca *OfflineManager* foi construída. Primeiramente foram apresentadas algumas decisões sobre a plataforma escolhida para o desenvolvimento, a plataforma Android, e sobre como seria implementada a abordagem: na forma de uma biblioteca. Foi introduzida a biblioteca Retrofit como cliente HTTP que é uma dependência da a biblioteca *OfflineManager*, e por fim discutiu-se os mecanismos

de *feedback* utilizados, dentre as opções disponíveis na plataforma Android. Neste capítulo foi apresentado de forma mais detalhada o método que a biblioteca *OfflineManager* expõe para uso dos desenvolvedores, chamado `treatedCall`. Esse método é o que encapsula as chamadas do Retrofit, e de acordo com os parâmetros mostrados, o desenvolvedor pode escolher o comportamento para aquela chamada:

- Tratamento para quando o dispositivo estiver sem conexão com a internet
- Tratamento para quando o servidor encontra-se indisponível
- Número de novas tentativas a serem executadas enquanto o servidor encontra-se indisponível
- Modo verboso que define a exibição de *feedbacks* para o usuário
- Métodos de *callback* para que o desenvolvedor possa implementar a lógica da aplicação no retorno da chamada tratada. Da mesma forma como faria com os métodos de *callback* oferecidos pelo Retrofit.

No próximo capítulo são apresentados os métodos utilizados para avaliar a biblioteca *OfflineManager*, tanto do ponto de vista do desenvolvedor que irá aplicá-la, quanto do ponto de vista do usuário da aplicação final.

É importante adicionar que a biblioteca *OfflineManager* foi disponibilizada no GitHub[8] de maneira livre, para que desenvolvedores da comunidade Android possam utilizá-la em seus projetos. A plataforma do GitHub permite também que desenvolvedores contribuam com o projeto, sugerindo alterações, melhorias e correções, por meio do chamado *pull-request*[9]. Desde a publicação da biblioteca, da versão apresentada aqui nesta dissertação, já houve um contribuidor que propôs melhoria no processo de permissões da biblioteca, no `AndroidManifest.xml`, e o seu *pull-request* foi aceito e incorporado à biblioteca *OfflineManager*. Também, na época da escrita dessa dissertação, um desenvolvedor já entrou em contato pois está em processo de iniciar a utilização da biblioteca *OfflineManager* em seu projeto de aplicativo.

[8] https://github.com/Sidarta/OfflineManager

[9] Processo da plataforma GitHub onde um contribuidor sugere uma mudança para melhoria, alteração ou correção, no projeto, e os responsáveis pelo projeto podem aprovar essa mudança de forma a incorporá-la ao código

4 Avaliação

O objetivo desta pesquisa foi o de tentar superar os dois pontos críticos elencados no Capítulo 1:

1. Oferecer apoio à solução do problema em todas as camadas de software: desde o *back-end* até o *front-end*; e

2. Reduzir o esforço de implementação do requisito de funcionamento *offline* tanto em desenvolvimentos iniciais como em sistemas já desenvolvidos.

Para avaliar o atendimento a este objetivo, portanto, a avaliação realizada buscou não apenas validar a implementação, mas sim examinar diversos aspectos pertinentes ao escopo que a biblioteca pretende abordar. Assim, foram feitas avaliações do ponto de vista do desenvolvedor, visando identificar as vantagens e desvantagens que a biblioteca traz para o desenvolvimento em termos de alcance de todas as camadas e redução de esforço de implementação. Foi também feita uma avaliação do ponto de vista do usuário final do aplicativo, visando identificar se os requisitos de uso de aplicativos em situações de indisponibilidade de conexão são atendidos mediante o emprego da biblioteca.

Assim, foram propostos para este trabalho de mestrado quatro diferentes avaliações para o projeto, cada uma com o objetivo de avaliar um aspecto diferente da biblioteca. As **duas primeiras avaliações** dizem respeito aos quesitos técnicos de implementação, e pertencem ao escopo pertinente ao processo de desenvolvimento de uma aplicação, e como a biblioteca *OfflineManager* pode ser aplicada neste processo. Essas avaliações foram feitas em uma fase inicial de desenvolvimento, sendo pensada desde o começo em conjunto com as outras especificações e requisitos de projeto, e também em uma fase posterior de desenvolvimento, no qual o projeto já existe e o requisito de funcionamento *offline* surge, demandando que possíveis refatorações de código sejam feitas a fim de atender a esse requisito. Já as **duas últimas avaliações** se referem às questões de usabilidade e experiência de usuário final que a biblioteca proporciona sendo utilizada em uma aplicação. Este aspecto foi bastante discutido na proposta deste trabalho de mestrado, pois é de suma importância que o tratamento oferecido pela biblioteca percorra também a camada de interface, deixando o aplicativo em que foi utilizada com a mesma usabilidade, sem perda de qualidade pela adição de comportamento em cenários de uso *offline*; ou até com melhor usabilidade, permitindo de forma controlada ou transparente utilizações que antes não eram possíveis em casos de intermitência na conexão com a Internet ou com o servidor.

A seguir são apresentados detalhes das quatro avaliações conduzidas e os resultados obtidos.

4.1 Opinião de especialista

O primeiro processo de avaliação foi uma avaliação por especialista, e consiste no processo de elencar um conjunto de especialistas do domínio em questão (aplicativos móveis), que podem atuar tanto na área acadêmica quanto na área de desenvolvimento (em empresa), possuindo também experiências heterogêneas a fim de avaliar a biblioteca por óticas diferentes, para realizar um estudo sobre a a biblioteca desenvolvida, e por fim responder a um questionário de avaliação, que percorre diferentes aspectos e conceitos de desenvolvimento – no final um estudo é feito sobre o conjunto dessas avaliações dos especialistas, a fim de produzir uma avaliação final por consenso sobre a biblioteca a eles apresentada.

Opinião de especialista consiste em um processo científico frequentemente empregado para interpretar dados, prever comportamentos de sistema, ou avaliar incertezas (COOKE, 1991). Ele é bastante utilizado, principalmente para avaliar análises probabilísticas ou processos de tomada de decisão, à medida que conhecimento sobre os diversos campos de estudo envolvidos nesses contextos são incompletos, e dados experimentais ou estatísticos não são facilmente encontrados e empregados (LI; SMIDTS, 2003). É um processo também bastante empregado no cenário atual da engenharia de software, apesar de existirem ainda algumas controvérsias, como a possibilidade de dados informais poderem ser influenciados por opiniões pessoais (KITCHENHAM et al., 2007) ou o grau elevado de subjetividade empregado, demonstrado em estudos prévios (LI; SMIDTS, 2003).

Esse processo de avaliação por opinião de especialista, empregado para avaliar a biblioteca construída neste trabalho de mestrado, se baseia na avaliação realizada no trabalho de Garcia, Almeida e Meira (2011) e no trabalho de Li e Smidts (2003) que sugere os seguintes passos:

1. Definição do Problema a ser avaliado: o contexto do problema, e o problema em si devem ser claramente articulados e definidos;

2. Seleção dos especialistas: um número razoável de especialistas deve ser selecionado, baseado em um conjunto de critérios. Esse conjunto de critérios deve incluir: credibilidade, conhecimento e confiabilidade dos especialistas;

3. Treinamento dos especialistas: etapa importante do processo de familiarização dos especialistas com os objetivos da elicitação, bem como com os detalhes do problema em questão. O objetivo é que haja um entendimento comum do problema entre os especialistas, dado que responderão a um mesmo questionário;

4. Elicitação de opiniões: este passo realiza as perguntas certas, e garante condições que induzam a um exercício de elicitação;

5. Agregação de opinião: a ideia central desse passo é alcançar uma agregação de opiniões ou um consenso sobre o qual seja possível tomar alguma decisão;

6. Tomada de decisão: este último passo consiste na tomada de decisão baseada na opinião agregada ou no consenso.

Além disso, alguns outros aspectos como o número de especialistas necessários para o estudo, a identificação e calibração de possíveis influências, e a técnica de agregação de opinião utilizada, precisam ser esclarecidas (LI; SMIDTS, 2003), conforme discussão que se segue.

4.1.1 Número de especialistas

A priori, se um especialista for perfeito (conhecimento infinito sobre o tópico do estudo e nunca erra), então o número de especialistas necessário será um. Porém, como existe a chance de um especialista real cometer um erro, é seguro a utilização de mais de um especialista. Por isso, existe uma tendência a buscar segurança nos números (GARCIA; ALMEIDA; MEIRA, 2011) – aumentando a quantidade de especialistas na medida do possível.

No entanto, foi mostrado que a existência de dependência entre os especialistas reduz a incerteza[1] sobre os dados, aproximando-se de um valor-limite mínimo, fazendo desnecessário o uso de muitos especialistas. As principais dependências existentes são: similaridades no treinamento, educação e experiência (LI; SMIDTS, 2003).

É importante ainda ressaltar a diferença entre o papel da opinião de especialista em um estudo e o papel de uma amostra utilizada em um cálculo estatístico. O processo de elicitação por opinião de especialista assume uma avaliação sobre o tema em questão que considera a experiência de mundo real do especialista, e mais especificamente, dos valores a serem estudados. Assim, se houver credibilidade nos especialistas, um já bastaria para prover certo grau de avaliação sobre o valor real avaliado ou elicitado. E ainda, dado que haja dependência entre os especialistas, aumentar o número de opinantes não ajudará no sentido de melhorar a precisão ou credibilidade do estudo sobre o valor avaliado. Em contrapartida, amostras em estudos estatísticos representam instâncias de fenômenos probabilísticos, portanto, aqui sim, o aumento do número de amostras infere melhoria na precisão da descrição (GARCIA; ALMEIDA; MEIRA, 2011).

[1] incerteza aqui significa a diferença entre a análise de um especialista e o valor real de um dado. Essa diferença é uma variável estocástica.

4.1.2 Seleção dos especialistas

Idealmente é de se esperar que os especialistas sejam selecionados de acordo com o maior grau de precisão nos julgamentos sobre o contexto, mas como não existe um processo formal sobre o qual isso pode ser alcançado (LI; SMIDTS, 2003), alguns pontos são definidos como guia para a seleção dos especialistas (ORTIZ et al., 1991). São eles:

1. Especialistas devem ter demonstrado conhecimento por meio de publicações, experiência de trabalho, e consultorias gerenciando estudos relacionados a área;
2. Especialistas devem apresentar e representar vasta variedade de experiências, como por exemplo, obtidas em universidades, consultorias, laboratórios, entre outros.
3. Especialistas devem representar uma vasta perspectiva sobre o problema, tão quanto for possível;
4. Especialistas devem estar dispostos a serem elicitados de acordo com o método escolhido para o estudo.

Utilizando esses pontos como parâmetro, a escolha dos especialistas foi adaptada para o contexto da biblioteca aqui apresentada, no sentido de aproximar essas características do cenário atual da computação móvel, mais especificamente, de desenvolvimento de aplicativos para dispositivos móveis. Portanto, foram selecionados 15 especialistas considerando suas experiências diversas, e disponibilidade. Neste caso, todos os especialistas trabalham majoritariamente na indústria, e não na área acadêmica. Isso se dá pois a aplicação da biblioteca, do ponto de vista de desenvolvimento, se aproxima mais de um aspecto prático e de mercado. Por isso, essa experiência se mostrou mais válida como critério de seleção. Outro aspecto importante é que o item 1 citado anteriormente foi subdividido em alguns critérios mais específicos para o contexto deste trabalho. O conceito de experiência foi descrito sobre os seguintes pré requisitos para a seleção dos especialistas:

1. Mais de um ano de experiência com a tecnologia Java ou Android, combinadas;
2. Pelo menos algum contato com projetos que utilizaram a arquitetura REST para comunicação;
3. Pelo menos alguma experiência com a biblioteca Retrofit para comunicação REST na plataforma Android.

4.1.3 Identificação e calibração de possíveis influências

Tversky e Kahneman (1975) separam as possíveis influências ou desvios em duas possíveis categorias: desvio por localização e desvio por superconfiança. O primeiro significa a sistemática super ou supra estimativa da variável quantidade, e o segundo se

refere a uma tendência em afirmar uma gama artificialmente menor de conhecimento do que o conhecimento atual do especialista. O conhecimento e posteriormente compensação desses possíveis desvios é conhecido como calibração dos especialistas. Li e Smidts (2003) sugerem alguns mecanismos para calibração, por exemplo: para desvios de superconfiança, encorajar os especialistas a encontrarem argumentos que contrariem suas opiniões iniciais. No contexto desse trabalho porém, a calibração de desvios foi tratada no sentido de as perguntas do questionário oferecerem, posteriormente, espaço para explicações ou sugestões dos especialistas. Além disso, um acompanhamento foi fornecido e os especialistas foram encorajados a requisitarem explicações sobre qualquer ponto passível de dúvida, e por fim, nenhum indício de desvio foi encontrado durante o processo ou nas respostas ao questionário (GARCIA; ALMEIDA; MEIRA, 2011).

4.1.4 Técnica de agregação utilizada

Segundo Li e Smidts (2003), diferentes métodos de agregação vem sendo utilizados, e eles variam do mais simples como uma simples médias dos resultados obtidos, até mais complexos como o método de agregação Bayesian (CHHIBBER; APOSTOLAKIS; OKRENT, 1992).

Para a avaliação da biblioteca, neste estudo, foi utilizado um mecanismo simples de média das avaliações individuais de cada especialista. Isso justificado pela análise das respostas submetidas, que indicam que nenhum desvio ou influências foram introduzidas no estudo. Dessa forma, considera-se que todos os especialistas possuem igual valor e peso na produção final do resultado baseado na agregação das elicitações, assim como considerado no estudo de Garcia, Almeida e Meira (2011).

4.1.5 Processo de avaliação por especialista da Biblioteca

O processo de avaliação por especialista seguiu os seguintes passos, mencionados na seção 4.1: na primeira etapa o problema e o escopo da biblioteca foram definidos – o problema sendo a questão da utilização de aplicações em um cenário de uso *offline*; e escopo da biblioteca sendo uma tentativa de solução para esse problema, de forma a atender os pontos críticos citados no capítulo 1. Com isso, o questionário de avaliação foi elaborado. Na segunda etapa, os especialistas foram selecionados, como explicado anteriormente. Na terceira etapa foi feito o processo de treinamento dos especialistas, e consistiu em explicações a fim de familiarizar os especialistas com os objetivos a serem atingidos com a construção da biblioteca, as camadas que ela pretende tratar dentro do projeto de uma aplicação.

Ainda para fins de treinamento, foi disponibilizado um documento explicando de maneira simplificada como instalar e utilizar a biblioteca em um projeto, e ainda foi

disponibilizado um vídeo com um tutorial de instalação e utilização da biblioteca. A quarta etapa consistiu no envio do questionário para os especialistas e a quinta etapa, por fim, consistiu na coleta dos dados obtidos pelas respostas fornecidas, e a posterior caracterização da avaliação da biblioteca *OfflineManager*, que aborda aspectos como: facilidade de compreensão e familiarização com a biblioteca, facilidade de instalação e utilização pela primeira vez em uma chamada, suficiência dos tratamentos oferecidos e das mensagens de *feedback* disponibilizadas, e também viabilidade de utilização da biblioteca, aplicabilidade e o quão ela, por fim, ajuda no tratamento da problemática de tratamento para utilização em modo *offline*.

Mais especificamente sobre o questionário utilizado para avaliação dos especialistas, ele foi elaborado em abril de 2017, baseado em um estudo extensivo sobre as tecnologias atuais de desenvolvimento para aplicações Android. Devido a motivação de a biblioteca ser uma ferramenta que contribua de forma prática com a comunidade de desenvolvimento Android, o questionário abordou principalmente questões técnicas e de implementação, voltadas ao processo de aplicação e utilização da biblioteca. Abordou também a frequência com que os especialistas se deparam com problemas relacionados a utilização em modo *offline*, ou frequência com que precisam implementar requisitos de uso em modo *offline*, durante o desenvolvimento de aplicações móveis. Esse questionário foi revisado duas vezes ao longo de dois meses, a fim de ser refinado para obter os dados desejados, e para essas revisões foram consideradas opiniões de dois especialistas que trabalham na área de desenvolvimento móvel, e um pesquisador da área de engenharia de software. Esses três indivíduos que contribuíram para a versão final do questionário não participaram do processo de avaliação.

A versão final do questionário utilizado para essa avaliação contém 23 questões, pode ser encontrada no apêndice A, e foi enviada por e-mail para o conjunto inicial de especialistas selecionados, para responderem de forma digital. Do conjunto inicial, 7 responderam dentro do prazo de um mês que lhes foi dado para a submissão das respostas. Desses 7 especialistas que responderam a tempo e consequentemente compõem os resultados desta pesquisa, um deles é um desenvolvedor de aplicativos para Android estrangeiro que foi colaborador do projeto da biblioteca Retrofit, e os outros 6 são desenvolvedores de aplicativos de empresas diferentes da região do interior de São Paulo. A seguir apresenta-se o perfil de cada especialista que participou desta avaliação:

Especialista A: Trabalha em empresa e caracteriza sua posição atual como *Especialista de tecnologia*. Possui mais do que 4 anos de experiência com Java e mais do que 4 anos de experiência com Android. Além disso, trabalha diariamente com a arquitetura REST e possui entre 1 e 2 anos de experiência com a biblioteca Retrofit, onde já trabalhou em mais de 3 projetos grandes que a utiliza;

Especialista B: Trabalha em empresa e se caracteriza como *Desenvolvedor de aplciativos android*. Possui 2 a 3 anos de experiência com Java e 1 a 2 anos de experiência com Android. Trabalha frequentemente com a arquitetura REST e tem 1 a 2 anos de experiência com a biblioteca Retrofit, onde participou de um projeto grande que a utiliza;

Especialista C: Trabalha em empresa e se descreve coom *Desenvolvedor Android*. Possui de 1 a 2 anos de experiência com Java e até um ano de experiência com Android. Além disso, possui contato médio com a arquitetura REST e até um ano de experiência com a biblioteca Retrofit, trabalhando em até 3 projetos grandes que a utiliza;

Especialista D: Trabalha em empresa e se caracteriza também como *Desenvolvedor Android*. Possui de 3 a 4 anos de experiência com Java e de 3 a 4 anos de experiência com Android. Trabalha com a arquitetura REST diariamente e possui entre 1 e 2 anos de experiência com a biblioteca Retrofit, onde já participou de mais do que 3 projetos grandes que a utiliza;

Especialista E: Trabalha em empresa e se caracteriza também como *Desenvolvedor Android*. Possui de 1 a 2 anos de experiência com Java e até um ano de experiência com Android. Além disso, trabalha frequentemente com a arquitetura REST e possui até um ano de experiência com a biblioteca Retrofit, participando de até 3 projetos pequenos ou médios que a utiliza;

Especialista F: Trabalha em empresa e também se caracteriza como *Desenvolvedor Android*. Possui entre 2 e 3 anos de experiência com Java e 1 a 2 de experiência com Android. Trabalha diariamente com a arquitetura REST e possui entre 1 e 2 anos de experiência com a biblioteca Retrofit, onde já participou de até 3 projetos pequenos ou médios que a utiliza;

Especialista G - Contribuidor do projeto Retrofit: Trabalha em empresa e se caracteriza como *Desenvolvedor de aplicativos móveis*. Possui de 3 a 4 anos de experiência com Java e de 2 a 3 anos de experiência com Android. Trabalha frequentemente com a arquitetura REST e possui de 2 a 3 anos de experiência com a biblioteca Retrofit, onde participou de até 3 projetos grandes que a utiliza. Além disso, é um contribuidor do projeto do Retrofit;

4.1.5.1 Resultados obtidos

As 7 primeiras questões do questionário apenas avaliavam as experiências do especialista em questão, com as diversas tecnologias presentes na biblioteca *OfflineManager*. Duas dessas questões perguntavam a área de atuação principal do participante, e a descrição da posição atual de trabalho, caso trabalhem. O objetivo era assegurar que os

especialistas de fato pertenciam ao grupo-alvo esperado, de desenvolvedores de aplicações móveis. Dos 7 participantes, todos trabalham em empresas e 6 deles na área de desenvolvimento de aplicativos móveis. Um dos especialistas (especialista A) identificou sua posição atual como especialista de tecnologia.

Dessas 7 primeiras perguntas, as outras 5 questionam sobre as experiências dos especialista com as tecnologias relacionadas ao projeto da biblioteca, para garantir que as condições mencionadas anteriormente fossem atingidas.

Essas questões mostraram que a seleção dos especialistas foi válida: Combinando a experiência Java com a experiência Android dos participantes, todos possuem mais de um ano de experiência, sendo que cinco possuem mais de 3 anos de experiência combinada. Além disso, todos possuem pelo menos contato médio com a arquitetura REST: entendem o conceito mesmo tendo utilizado poucas vezes. Também, todos os especialistas tem alguma experiência com a biblioteca Retrofit e todos trabalharam em pelo menos um projeto que a utiliza.

Conhecendo um pouco sobre os especialistas participantes, e tendo validado suas experiências dentro dos pré-requisitos definidos, o questionário aborda sobre o cenário de utilização de aplicações em modo *offline*. Quando perguntados sobre já terem se deparado com uma situação onde precisaram fornecer usabilidade em modo *offline* em aplicações que estavam trabalhando, apenas o especialista G disse não ter se deparado com tal situação. Os outros 6 já tiveram essa experiência pelo menos uma vez, como mostra o gráfico da Figura 12.

A próxima pergunta a essa foi uma questão aberta para que os especialistas explicassem de forma sucinta como resolveram esse requisito de funcionamento *offline*, no contexto onde precisaram abordar esse problema. Dois dos especialistas (especialistas C e F) relataram que utilizaram o Firebase como *back-end* de suas aplicações para resolver o problema por meio de sincronismo, como citado no capítulo 1. Outros dois especialistas (especialistas B e E) relataram que, diante dos cenários de uso *offline*, simplesmente oferecem *feedbacks* sobre a falta de conexão, informando para o usuário tentar novamente, e restringem funcionalidades do aplicativo quando o dispositivo encontra-se sem conexão. O especialista A relatou que geralmente, diante desse requisito, utiliza um interceptador das chamadas feitas pelo Retrofit, e junto com uma biblioteca de armazenamento de dados ou *cache*, oferece persistência de dados em modo *offline*. Dois dos especialistas (especialistas D e G) não especificaram os modos como tratam esse requisito.

Partindo para a avaliação da biblioteca *OfflineManager* em si, o questionário indaga por meio de quatro questões fechadas sobre facilidade ou dificuldade para: instalação da biblioteca, compreensão da biblioteca, inicialização e aplicação em uma primeira chamada, e encontrar uma chamada onde a biblioteca possa ser aplicada. Sobre a instalação, o especialista D achou trivial, os especialistas A, B, E F e G acharam fácil, e o especia-

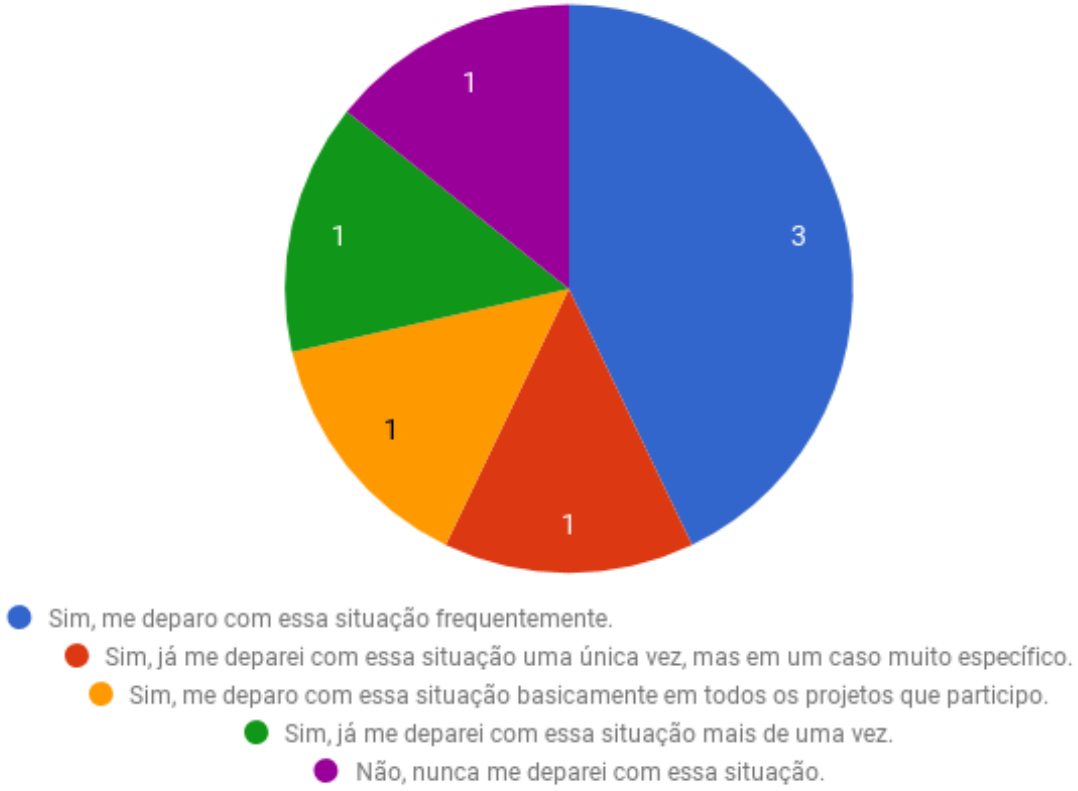
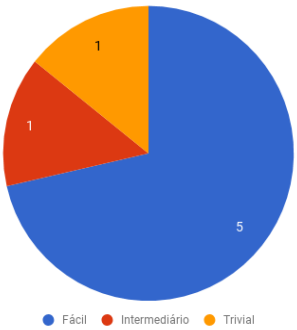


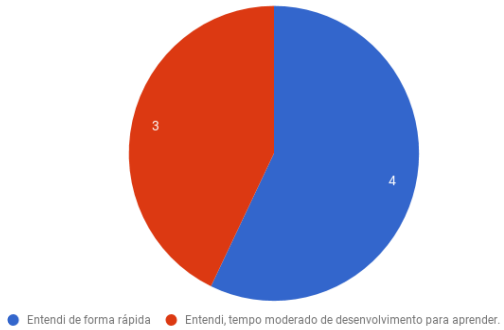
Figura 12 – Distribuição dos resultados da questão do questionário referente à experiência prévia com suporte *offline*.

lista C achou a dificuldade moderada. Sobre compreensão, os especialistas A, B, D e E entenderam de forma rápida, e os especialistas C, F e G entenderam em tempo moderado. Todos os especialistas acharam fácil inicializar a biblioteca e aplicá-la em uma chamada. Por fim o especialista E entendeu como trivial a forma de achar uma chamada possível de se aplicar a biblioteca. Os especialistas A, B, C, D e G avaliaram como fácil e apenas o especialista F avaliou como dificuldade moderada. A Figura 13 mostra a distribuição das respostas para essas perguntas de forma gráfica.

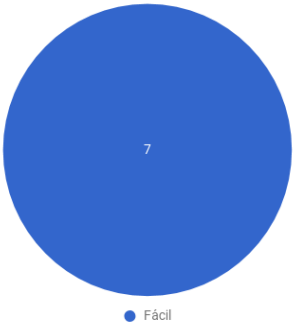
Essa parte do questionário mostra, de forma bem direta, que um dos objetivos propostos na construção da biblioteca foi atingido com sucesso. Para instalar, encontrar uma chamada possível de ser tratada, inicializar e tratar uma chamada, 5 especialistas avaliaram como fácil. E sobre a compreensão da biblioteca, mais da metade avaliou como fácil compreensão. Sobre esse último ponto, porém, é possível atribuir a dificuldade moderada de compreensão que foi relatada por 3 especialistas, à documentação. O aspecto de compreensão é facilmente melhorado com revisões da documentação disponibilizada, bem como a tradução para o português, uma vez que ela foi disponibilizada apenas em inglês.



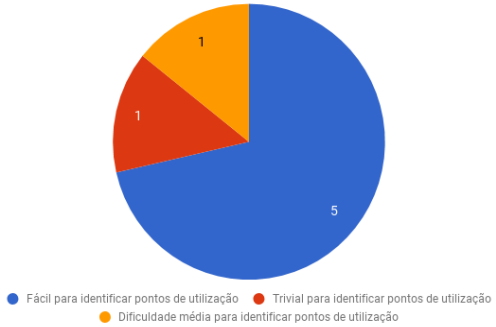
(a) Dificuldade para instalar a Biblioteca



(b) Dificuldade para entender a Biblioteca



(c) Dificuldade em inicializar e aplicar a biblioteca a uma primeira chamada



(d) Dificuldade para encontrar chamada onde a Biblioteca pode ser usada

Figura 13 – Distribuição dos resultados das questões do questionário referente à: (a) Dificuldade em instalar a biblioteca, (b) Dificuldade para entender a Biblioteca, (c) Dificuldade em inicializar e aplicar a biblioteca a uma primeira chamada, e (d) Dificuldade para encontrar chamada onde a Biblioteca pode ser usada

Posteriormente, os especialistas foram questionados sobre a suficiência dos modos disponíveis de tratamento para cenários de dispositivo *offline*, que são: *Enforce*, *Flexible* e *Transparent*, conforme discutido no Capítulo 3. Foram questionados também sobre a suficiência dos tratamentos oferecidos para cenários onde o servidor está indisponível, que são também três: *NoAction*, *Flexible* e *Transparent*. E foram questionados sobre as mensagens de *feedback* oferecidas para quando o modo verboso for marcado como verdadeiro, se são suficientes. Essas três questões são fechadas, mas oferecem um meio de o especialista responder algo diferente, caso entenda que sua opinião não se enquadra nas opções apresentadas.

Para o cenário de dispositivo sem conexão, os especialistas A, E, F e G entenderam que os modos disponibilizados são suficientes. Dois especialistas (especialistas B e D) entenderam que pelo menos um modo de tratamento é desnecessário, sendo que o especialista D explicou que, em seu ponto de vista, o usuário não deve tomar o tipo de decisão oferecido pelo modo flexível, tornando esse modo de tratamento desnecessário. O especialista C respondeu que na verdade faltam de 1 a 2 novos tipos de tratamento.

Quando pedido para o especialista C especificar quais seriam esses tratamentos

faltantes, ele respondeu que em seu ponto de vista, poderia ainda existir um modo de tratamento que persistisse as chamadas em uma estrutura mais duradoura, como um banco de dados local. Justifica que essa persistência seria útil para que as chamadas pudessem ser armazenadas e tentadas novamente, dentro de um espaço maior de tempo, e mesmo se a aplicação for encerrada ou iniciada novamente.

Esse resultado mostra que, em suma, a maioria dos especialistas concorda que os métodos oferecidos são suficientes para tratar os diferentes cenários onde pode haver interação em modo *offline*. Contudo, é interessante observar que, para um dos especialistas (especialista D), o tipo de usabilidade oferecido no modo flexível não faz parte do contexto de decisões que um usuário deve tomar durante o uso de uma aplicação móvel, tornando esse modo de tratamento desnecessário.

Para o cenário de servidor indisponível, os especialistas avaliaram da mesma maneira. Os especialistas A, E, F e G entendem que os tratamentos oferecidos são suficientes, o especialista C opina que faltam de 1 a 2 novos tratamentos, e os especialistas B e D avaliam que pelo menos um modo de tratamento é desnecessário, sendo que o especialista D acredita que o modo flexível não seja válido. Essa semelhança das avaliações pode ser facilmente explicada pelo fato de que, apesar de tratarem condições diferentes de uso em modo *offline* (servidor indisponível e aparelho sem Internet), a experiência final alcançada é a mesma e deve ser o mais transparente possível ao usuário. Assim, as características dos aplicativos e das chamadas onde cada modo será utilizado são as mesmas tanto para esses dois cenários de tratamento.

O especialista C, ao opinar que faltam de 1 a 2 novos tratamentos, aqui se refere também ao sugerido novo tratamento com estrutura mais duradoura de armazenamento das chamadas. A justificativa é a mesma, seria o mesmo tratamento que sugeriu anteriormente para cenário de dispositivo *offline*.

Já sobre as mensagens de *feedback* oferecidas pelo modo verboso, o questionamento é sobre se elas são suficientes. Diante disso, a grande maioria de 6 especialistas (especialistas A, B, C, D, E e G) respondeu que as mensagens são adequadas, sendo que o especialista F respondeu que existem de uma a duas mensagens desnecessárias.

Quando questionado sobre quais mensagens são desnecessárias, o especialista F disse que considera desnecessárias mensagens de *feedback* que aparecem quando o usuário já saiu da aplicação. Justifica que essas mensagens podem confundir o usuário se ele estiver utilizando outra aplicação. Completa que, em sua visão, somente as mensagens emitidas logo após alguma ação do usuário são necessárias.

As próximas três questões abordam: experiência de usuário, aplicabilidade da biblioteca *OfflineManager* e se existe algum problema na utilização da biblioteca. Com relação à experiência de usuário, os especialistas A, B, C e D responderam que a aplica-

ção da biblioteca não altera a experiência de usuário de um aplicativo. Por outro lado, os especialistas E, F e G responderam que a experiência de usuário de um aplicativo fica boa, depois da aplicação da biblioteca, ou seja, sua utilização melhora a experiência oferecida pelo aplicativo. É importante ressaltar que essa questão de usabilidade foi respondida pelos especialistas de acordo com suas visões de desenvolvedores, sem que eles de fato tenham aplicado e testado a experiência de uso oferecida pela biblioteca extensivamente. É por isso que na avaliação de comparação, explicada de forma mais completa posteriormente nesta dissertação, observamos um resultado diferente: ao comparar duas aplicações em cenário de uso *offline*, uma delas utilizando a biblioteca, todos os participantes do experimento entenderam que a experiência de usuário é de fato melhorada na que utiliza a biblioteca.

Sobre aplicabilidade, os especialistas A, B, F e G entendem que a biblioteca é aplicável, porém com alguma melhoria. Os especialistas C e E entendem que ela é de fato aplicável da forma como é, e o especialista D entende que ela é aplicável apenas para alguns cenários específicos. Em seguida, todos os especialistas responderam que não encontraram problemas evidentes, decorrentes da utilização da biblioteca *OfflineManager*.

Isso indica que do ponto de vista técnico, a aplicação da biblioteca não fere nenhum conceito de desenvolvimento de aplicativos móveis, e que também não insere nenhum problema de código decorrente de sua aplicação em um projeto real. Como nenhum especialista respondeu positivamente para a existência de problemas na utilização da biblioteca, não foi descrito nenhum problema na próxima questão, que indagava sobre os problemas encontrados.

Outra pergunta aberta que segue, pede para os especialistas listarem melhorias possíveis de serem feitas, e para esta apenas quatro especialistas fizeram sugestões: o especialista A afirma “Acredito que essa seja uma iniciativa muito interessante, como melhoria eu não deixaria como responsabilidade da biblioteca a exibição de componentes visuais (como pop-ups). Fornecer um *callback* já seria suficiente”. Ele acredita portanto que a interação com a camada de interface não é necessária, e que a biblioteca pode somente tratar as chamadas de forma a oferecer os *callbacks*.

O especialista C afirma que não está claro se a chamada ainda será executada caso o aplicativo for finalizado. Sua resposta estaria melhor encaixada se listada na questão sobre problemas encontrados. Mas pode-se concluir que ele sugere deixar isso mais claro na documentação da biblioteca.

O especialista D afirma que, apesar de interessante, os modos flexíveis de tratamento não são úteis para o usuário, portanto poderiam ser removidos das opções de tratamento. Sugere também que além de definir o número de tentativas, seja possível definir o tempo do intervalo para novas execuções. Por fim, diz que em sua opinião, para que a biblioteca *OfflineManager* seja aplicável em um projeto real, ela necessita de mais

funcionalidades. Nesta resposta é importante ressaltar que, apesar de sugerido, definir o tempo de intervalo para novas tentativas já é possível, por meio do construtor da classe `OfflineManager`. Assim, a conclusão é deixar essa ferramenta mais clara na documentação da biblioteca. O especialista D é o mesmo que respondeu sobre os modos flexíveis não serem necessários, nas questões sobre os modos de tratamento.

Quando questionado sobre quais funcionalidade a mais a biblioteca necessita para ser aplicável, o especialista D especificou algumas características a mais para a classe de serviço que executa as chamadas em segundo plano. Mostrou algumas características que ela poderia ter, tais como:

- Critério para cancelamento automático do serviço que iria executar a chamada, também chamado de *back-off criteria* ou critério de parada;
- Conhecimento, no serviço, sobre estado da bateria – se está conectada a uma fonte de energia e carregando, ou não. Isso para que serviços só sejam executados quando o dispositivo estiver ligado a uma fonte de energia, por exemplo.
- Persistência do serviço, mesmo através de reinicializações do dispositivo.

A última resposta é do especialista F, que sugere apenas a possibilidade de as mensagens de *feedback* serem sobrescritas de maneira programática, na inicialização da classe *OfflineManager*.

As duas últimas perguntas feitas no questionário oferecem uma resposta em escala de zero a dez, para que os especialistas atribuam um número-valor como forma de resposta.

A primeira questiona: "Em uma escala de 0 a 10, quanto você acha que a biblioteca *OfflineManager* pode ajudar nesses requisitos de funcionamento *offline*?". O especialista F atribuiu o valor 6, os especialistas D e G atribuíram o valor 7, os especialistas A, C e E atribuíram o valor 8 e o especialista B atribuiu o valor 9. Essas respostas indicam claramente que, com pesos diferentes, todos os especialistas acreditam que a biblioteca *OfflineManager* é uma ferramenta útil para o tratamento de utilização em modo *offline*, mas que poderia melhorar.

Finalmente, a última questão indaga: "Em uma escala de 0 a 10, quanto você acha viável aplicar a biblioteca *OfflineManager* em um projeto real?". Para essa, o especialista D atribuiu o valor 4, mostrando que não acha a biblioteca viável de ser aplicada em um projeto real. Sua opinião sobre a viabilidade foi também esclarecida quando indicou que a biblioteca para ser viável precisa oferecer mais funcionalidades, como por exemplo as listadas anteriormente. Quanto aos outros, no entanto, os especialistas B, C, E e F indicaram um valor 8, o especialista A indicou o valor 9, e o especialista G o valor 10. Isso mostra que a maioria dos especialistas concorda sobre a viabilidade da aplicação da biblioteca em um projeto real.

4.1.6 Discussão

Retomando o objetivo desta pesquisa e os dois pontos críticos citados no início deste capítulo, observa-se que os resultados indicam que pelo menos o primeiro ponto crítico (apoio à solução em todas as camadas do *software*) pode ser amenizado com o uso da biblioteca *OfflineManager*. Isso porque a questão de tratar o problema nas diferentes camadas, mesmo não sendo abordada explicitamente, foi avaliada por meio de alguns indícios: no geral os especialistas avaliaram os diferentes tratamentos oferecidos como suficientes, sendo que eles abordam tanto a camada de serviço, quanto a camada de *back-end* e de interface. A maioria (6 especialistas) também avaliou que as mensagens de *feedback* são adequadas. Além disso, 3 dos 7 especialistas entenderam que a aplicação da biblioteca melhora a experiência de usuário em uma aplicação.

Esses indícios, em conjunto com os indicadores que mostram que a biblioteca *OfflineManager* ajuda na solução para uso em modo *offline*, mostram que o problema em questão (utilização em modo *offline*) é resolvido pela biblioteca e portanto engloba todas as camadas do sistema.

O fato de a biblioteca *OfflineManager* ajudar na solução para o problema de uso em modo *offline* é explicitado pela penúltima questão, onde os especialistas responderam atribuindo pelo menos o valor 6 (numa escala de 0 a 10) quando questionados sobre o quanto a biblioteca ajuda para resolver esse problema.

Com relação ao segundo ponto crítico (redução do esforço de implementação), pode-se concluir pelos resultados que ele foi atendido pelo menos parcialmente. Isso porque nas questões referentes às dificuldades de utilização da biblioteca (instalar, compreender, aplicar a uma chamada, e encontrar uma chamada), as respostas indicam que de fato a biblioteca é facilmente utilizável. Outros indícios são os valores atribuídos para avaliação da aplicabilidade da biblioteca, e pela falta de problemas relatados em utilizar a biblioteca. Porém, não houve uma avaliação explícita que mostra facilidade maior em se utilizar a biblioteca *OfflineManager* no lugar de outras possíveis soluções. Não houve também nenhuma resposta que considerou uma aplicação a um sistema já pronto.

4.2 Aplicação em um projeto real: Inventum

Ainda no âmbito de desenvolvimento, a segunda avaliação foi feita pelo autor desta dissertação e desenvolvedor da biblioteca *OfflineManager*. Essa avaliação seguiu um caminho mais prático. Foi um processo de aplicação da biblioteca desenvolvida em um projeto real de aplicativo Android, de médio porte. Portanto, seu foco foi avaliar a facilidade de uso da biblioteca em um sistema já existente, conforme o segundo ponto crítico mencionado no início deste capítulo.

Nesse processo, avaliações diferentes foram feitas sobre a biblioteca e dizem respeito tanto à refatoração de código necessária, que pôde ser facilmente metrificada, quanto outras características simples de desenvolvimento, também facilmente metrificadas, como por exemplo: facilidade para adição e inicialização da biblioteca no projeto, número de mudanças de código necessárias para a aplicação da biblioteca em uma chamada ao servidor, etc. Em contrapartida, outras observações avaliativas também são bastante importantes e dizem respeito a dificuldades encontradas no processo de refatoração, pois alguns fluxos da aplicação foram projetados para funcionar sem considerar utilização *offline* e estavam muito amarrados à arquitetura do projeto. Nessas situações, apesar de a biblioteca ter oferecido um modo fácil de inserção de comportamento *offline*, alguns ajustes sobre a lógica e fluxos do aplicativo se fizeram necessários para que a experiência de usuário continuasse fluida.

A fim de caracterizar melhor esse processo de avaliação, alguns passos foram definidos anteriormente com base no processo de refatoração de uma aplicação já existente, a fim de parametrizar melhor os resultados obtidos. A implantação e análise seguiu os seguintes passos:

1. Definição da aplicação a ser refatorada;
2. Estudo da aplicação como um todo, e análise sob o prisma do requisito de funcionamento *offline*;
3. Estudo mais específico sobre os possíveis pontos de aplicação e interação *offline*;
4. Refatoração da aplicação para adicionar o requisito de funcionamento *offline* por meio da biblioteca *OfflineManager*;
5. Validação e pequenas correções sobre a refatoração realizada, a fim de ajustar e refinar o requisito implantado;
6. Avaliação do processo de implantação.

A seguir, esses passos são discutidos de forma mais detalhada.

4.2.1 Escolha da aplicação

O primeiro requisito e o mais importante para a escolha do projeto foi que ele deveria ser de código aberto, para que se pudesse ter acesso a todo o código da aplicação, a fim de realizar quaisquer mudanças necessárias.

O segundo critério, de caráter exclusivo, foi a necessidade de o projeto utilizar a biblioteca do Retrofit. É importante mencionar que o aplicativo a ser estudado já deveria fazer uso da biblioteca Retrofit antes da avaliação, pois como foi explicado no Capítulo

3, Retrofit é um pré-requisito para a utilização da biblioteca *OfflineManager*. Portanto, as observações aqui feitas dizem respeito exclusivamente às vantagens e desvantagens da biblioteca *OfflineManager*, e não ao uso do Retrofit.

Outros critérios foram importantes também para a seleção do projeto, dentre eles a atualidade do projeto, com a finalidade de selecionar um projeto que não fosse tão antigo ou depreciado do ponto de vista de tecnologias ou arquiteturas utilizadas, considerando que diversas versões da plataforma Android foram lançadas nos últimos anos. Além disso, a biblioteca *OfflineManager* suporta apenas versão 5 do Android ou superior, como apresentado no Capítulo 3.

Outra característica considerada foi o tamanho do projeto, que pode ser metrificado pela quantidade de classes, no ponto de vista técnico, e pela quantidade de funcionalidades que oferece, do ponto de vista de utilização e requisitos. Essa característica é importante pois aproxima o processo aqui realizado de um cenário real de utilização da biblioteca. Por fim, o último critério, apesar de mais subjetivo, é também igualmente importante, é o grau em que o requisito de funcionamento *offline* se aplica ao projeto em questão, ou seja, o quanto é interessante para a aplicação escolhida oferecer tratamento para utilização em cenários *offline*.

Dados os critérios e características consideradas, fez-se uma busca no GitHub, por ser uma popular plataforma de compartilhamento e disponibilização de projetos em código livre. A busca foi realizada percorrendo extensivamente os projetos disponíveis, utilizando as palavras chave *Android* e *Retrofit*, dado o pré-requisito da aplicação ser em Android e utilizar a biblioteca Retrofit. O maior desafio dessa busca deu-se pelo fato de que grande parte dos projetos disponibilizados em Android são bibliotecas, dificultando a seleção de projetos de aplicações completas.

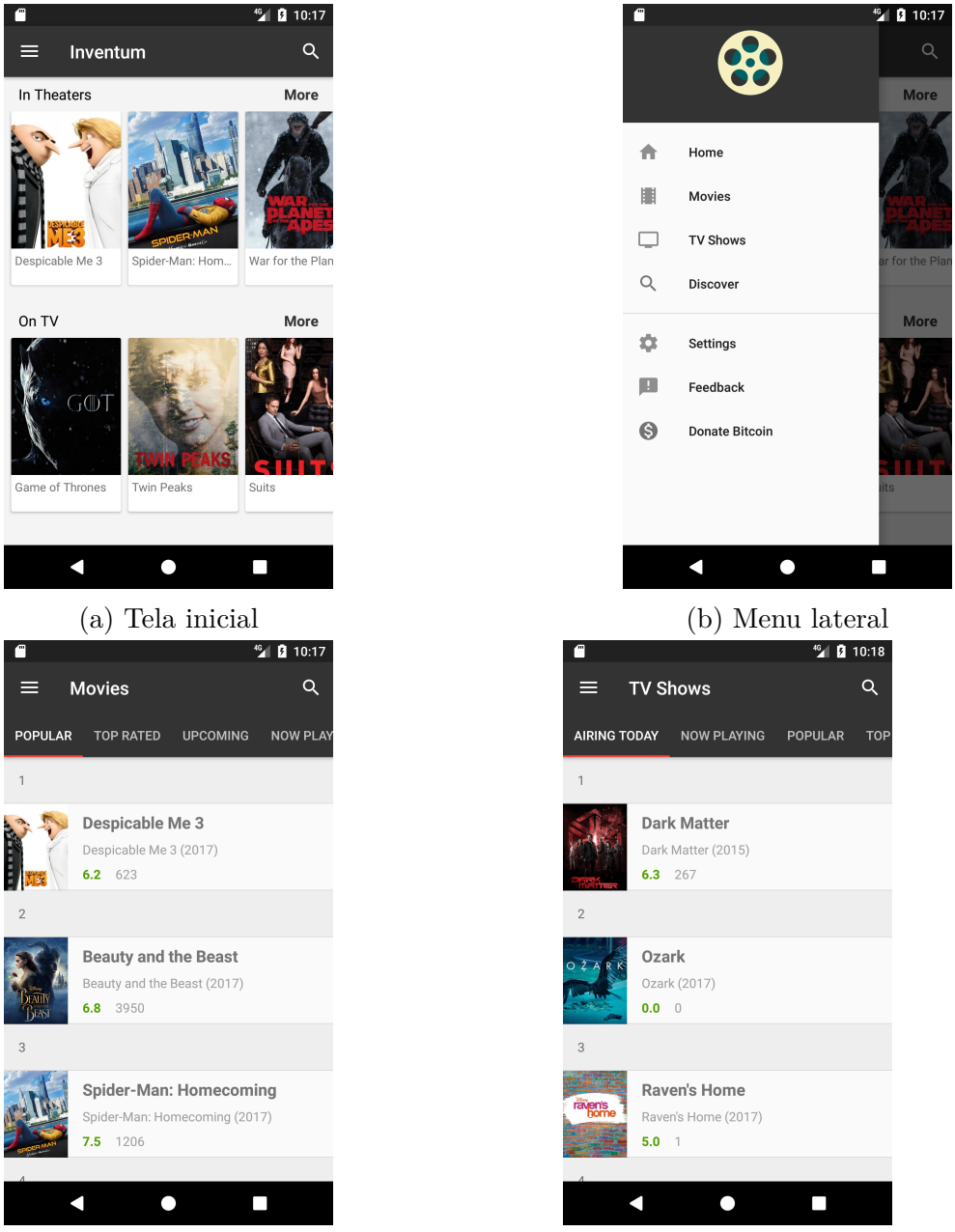
Dos projetos de aplicativos completos que foram encontrados, parte não se utilizava da biblioteca Retrofit, e parte eram projetos pequenos ou apenas aplicativos construídos na forma de tutorial, ou seja, aplicativos construídas com a finalidade de detalhar ou explicar algum conceito ou tecnologia sobre a plataforma Android. Por esses motivos, poucos candidatos foram encontrados.

Dentre três aplicações candidatas, a que mais se adequou nos critérios e características consideradas foi a aplicação chamada Inventum[2]. Trata-se de um aplicativo Android que utiliza arquitetura REST, em conjunto com o Retrofit, e faz requisições no *back-end* do TMDb – uma plataforma que, em conjunto com um banco de dados sobre filmes, séries de TV e atores, disponibiliza esses dados por meio de uma API aberta[3].

Do ponto de vista de funcionalidades, esse aplicativo não requer que o usuário faça cadastro ou *login*, e apresenta na tela inicial listas de filmes e séries de TV que estão em alta

[2] https://github.com/qqq3/inventum
[3] https://www.themoviedb.org/

no momento. Pelo menu lateral, o usuário consegue navegar para seções de filmes, onde filmes podem ser listados de acordo com quatro características: mais populares, mais bem avaliados, "estão por vir" e "passando atualmente". Também no menu é possível navegar para uma tela semelhante, porém para séries de TV, que ordena a listagem de acordo com as características: "passando hoje", séries atuais, populares, e mais bem avaliadas. A Figura 14 mostra a tela inicial do aplicativo, as opções disponíveis no menu lateral e as duas listagens mencionadas: de filmes e de séries de TV.



(a) Tela inicial

(b) Menu lateral

(c) Listagem de filmes

(d) Listagem de séries de TV

Figura 14 – Telas do aplicativo Inventum que mostram a tela inicial, as opções do menu lateral, e as telas de listagem de filme e de séries de TV

Tanto nessa duas telas de listagens de filmes e séries de TV, quanto na tela inicial,

mostradas na Figura 14, o usuário pode pressionar o ícone no canto superior direito para realizar uma busca rápida, onde ao digitar qualquer texto, os resultados vão sendo populados abaixo. No menu lateral, por fim, o usuário consegue ir para uma tela chamada "Discovery", que nada mais é do que uma busca avançada: é possível preencher alguns dados e realizar uma busca por resultados filtrados de acordo com esses dados.

As últimas três opções listadas no menu lateral do aplicativo: *Settings* (configurações), *Feedback* (Feedback sobre o aplicativo) e *Donate Bitcoin* (Doar bitcoins), mostrada na tela apresentada na Figura 14, não foram consideradas na refatoração apresentada nesta seção. A funcionalidade *Settings* não interage com a internet, e as funcionalidades *Feedback* e *Donate Bitcoin* abrem outra aplicação do dispositivo (se houver) para serem executadas.

O aplicativo Inventum também é um projeto atual: última atualização, indicada em sua página no GitHub, foi em 28 de janeiro de 2017. Além disso, é um projeto que pode ser considerado de médio porte, pois oferece uma quantidade razoável de funcionalidades: desde funcionalidades grandes como as listas de filmes e séries, detalhes dos filmes ou séries, pesquisa e busca avançada; até requisitos não-funcionais como por exemplo armazenamento em *cache*. Além disso, é composto por 57 classes e 5 interfaces e possui 23 arquivos de *layout* (arquivos que definem as interfaces das telas da aplicação). Dessas classes:

- 13 são *activities* (atividades),
- 6 são *fragments* – fragmentos: semelhante a uma atividade, porém define comportamento de apenas uma parte da tela ou de algum componente em específico,
- 5 são *adapters* – classes que definem o comportamento de listas e seus itens,
- 15 são classes de modelo e representam entidades utilizadas na aplicação, como por exemplo a entidade *Movie* (Filme), ou a entidade *Person* (Pessoa).

O projeto conta, na época de sua utilização para esta dissertação, com 36 *commits* na plataforma de versionamento do GitHub, e com 3 contribuidores, desde a sua criação em 8 de Janeiro de 2017, definida pelo *commit* inicial do projeto[4].

Por fim, devido ao seu funcionamento, é um projeto passível de aplicação de funcionalidades *offline*, uma vez que o tratamento em *cache* é feito apenas para a tela de detalhes de filmes e não cobre nenhum tipo de tratamento para o caso de o servidor estar *offline*; e as demais telas do sistema não podem ser tratadas com *cache* por questões de requisito ou de funcionamento, ou seja, caso o dispositivo perca a conexão com a Internet, a maioria das funcionalidades e telas ficarão indisponíveis. Do ponto de vista de usuário, o

[4] https://github.com/qqq3/inventum/commit/675bb32579533d54cf025a38b938e5bd177db6b5

requisito de funcionamento *offline* é interessante pois mesmo sem conexão, algumas ações ainda poderiam ser realizadas, *feedbacks* poderiam ser melhorados, e no geral a usabilidade poderia continuar boa, mesmo com o dispositivo sem conexão, ou com o servidor indisponível.

4.2.2 Estudo da aplicação e requisito de funcionamento *offline*

Depois de selecionado o projeto para a implantação da biblioteca, foi feito um estudo sobre o aplicativo em questão para a compreensão de suas funcionalidades, e também uma análise sobre o código do projeto para o entendimento de sua implementação.

Em seguida, foi analisada a questão do funcionamento *offline*: o aplicativo trata a questão de conexão com a Internet em uma camada bastante superficial, na forma de verificações de conectividade antes da realização de carregamentos das telas. Isso faz com que a maioria das funcionalidades do sistema não ofereça nenhum tipo de tratamento para funcionamento *offline*. O que acontece, caso o dispositivo perca a conexão, é que o aplicativo oferece um *feedback* para o usuário informando sobre a falta de conexão com a Internet, como mostrado na Figura 15.

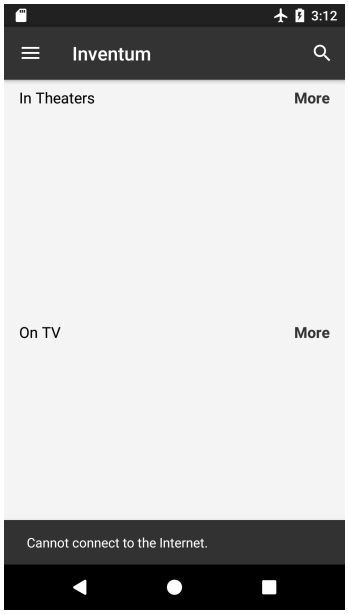


Figura 15 – Tela do aplicativo Iventum em um cenário sem conexão com a Internet, mostrando uma mensagem de *feedback*, sem o tratamento da biblioteca *OfflineManager*

Outro tratamento parcial oferecido, no sentido de funcionamento em caso de falta de conexão, é o armazenamento em *cache* local dos dados de filmes, séries de TV e atores. Sempre que o usuário acessa os detalhes de um desses três dados, é feita uma verificação para saber se eles já estão em *cache*. Se sim, os dados retornados são os dados da *cache*, senão os dados retornados são buscados pela requisição ao servidor, que além de mostrar os dados buscados, atualiza as entradas na *cache* para o dado recém buscado.

Esse tratamento faz que, em caso de falta de conexão, ainda existam alguns dados disponíveis para o acesso, desde que o usuário já tenha navegado por aquela tela anteriormente. Portanto, é possível afirmar que o aplicativo quase não oferece funcionalidades para cenários de utilização em modo *offline*, uma vez que o único tratamento real oferecido é o de *cache*, mas depende da disponibilidade prévia daquele dado na *cache*, e esse tratamento não considera nenhum outro aspecto de usabilidade ou interação. Pode-se afirmar, portanto, que esse tratamento tem como principal objetivo melhorar o desempenho.

É possível afirmar também que a aplicação não oferece nenhum tratamento para o caso de o servidor estar indisponível, pois não existe nenhum código que trata desse aspecto. O que aconteceria, nesse cenário, seriam retornos de mensagens de erro possivelmente difíceis de serem compreendidas por um usuário comum, pois os erros causados por exceções de indisponibilidade do servidor não são tratados de maneira refinada, são apenas retornados ao usuário final.

Analisando também pelo fator de experiência de usuário quando a conexão com a Internet não está disponível, algumas observações podem ser feitas: algumas telas, como por exemplo a tela inicial da aplicação, apesar da mensagem de *feedback* informar a falta de conexão, apresentam uma interface bastante ruim, pois nada é carregado e a tela fica em branco, como mostrado na Figura 15.

Outras telas, como a da funcionalidade de busca rápida, não apresentam *feedback* sobre ações realizadas em momentos de falta de conexão, o que gera comportamento inconsistente. Nessa busca, quando existe Internet disponível, o usuário digita os caracteres da busca e para cara carácter digitado, uma busca é realizada, fazendo com que os resultados mudem para cara carácter digitado conforme a palavra se forma. Se a Internet cair em algum momento nesse processo, os resultados param de serem trazidos, sem nenhuma mensagem de *feedback*.

Além disso, nessa tela, o botão contextual de submeter a busca pelo teclado virtual apenas esconde o teclado da tela. Não permite ao usuário a ação de buscar, depois que um texto foi digitado com conexão *offline*. Existe ainda a tela “Discover”, que apesar de não apresentar comportamento inconsistente, pode causar um pouco de frustração, à medida que o usuário gasta tempo preenchendo as opções e personalizando uma busca, e ao clicar para buscar, se não houver conexão, nada acontece e ele terá que realizar a mesma busca e digitar novamente suas opções em um segundo momento.

É claro que alguns desses problemas citados anteriormente na usabilidade em um cenário *offline* podem ser resolvidos puramente com refatorações de código sem a utilização da biblioteca *OfflineManager*. O simples fato de analisar a aplicação nesse contexto já permite que algumas alterações sejam feitas de maneira a melhorar a qualidade geral do aplicativo em um cenário de conexão intermitente com a Internet.

A principal delas, que inclusive foi identificada no experimento de comparação (a ser apresentado mais adiante, na Seção 4.3) é a funcionalidade do "*Pull to refresh*" (ou arraste para recarregar). É uma função que a maioria dos aplicativos atuais implementa, e opera de maneira que o usuário pode arrastar a tela da aplicação para baixo para forçar um recarregamento dos dados. Outras alterações possíveis seriam a melhoria de algumas mensagens de *feedback*, implantação de alguns processos de recarregamento dos dados baseado no estado da conexão, ou mudança de alguns fluxos de utilização para comportar possíveis casos de perda de conexão.

Observa-se porém, que as alterações sugeridas, que entende-se que seriam as possíveis alterações a fim de melhoria em um cenário *offline*, tendem a se aproximar de mudanças no código que não passam somente pela camada de comunicação e de dados, mas sim diretamente pelas camadas de interface e de usabilidade.

Tendem também a se aproximar cada vez mais de alterações mais complexas nos fluxos de funcionamento da aplicação, deixando para trás as pequenas alterações, que são poucas. Isso mostra que de fato existe um interesse em aplicar a biblioteca do *OfflineManager*, a fim de agrupar essas alterações necessárias em uma biblioteca que resolva os diferentes cenários, nas diversas camadas da aplicação, de forma mais centralizada e direta.

4.2.3 Refatoração e validação da solução final

Depois de ter analisado os requisitos de funcionamento *offline* na aplicação Inventum, partiu-se para a etapa de desenvolvimento de fato, que consistiu em instalar a biblioteca *OfflineManager*, e aplicar o tratamento nas chamadas que a aplicação fazia ao *back-end*.

Começando pela instalação, o processo é bem simplificado pois a mesma foi configurada de forma a ser utilizada com o Gradle[5], que é o gerenciador de dependências da plataforma Android. Portanto, basta adicionar a dependência no Graddle da aplicação. Após sincronizar o projeto, a biblioteca *OfflineManager* já está disponível para utilização.

Nesta etapa de desenvolvimento, o conhecimento sobre a aplicação adquirido na etapa anterior foi bastante útil para identificar de forma mais rápida os pontos onde chamadas HTTP eram executadas por meio do Retrofit. Além disso, o projeto utiliza uma organização de pacotes bastante popular no cenário de desenvolvimento Android, separando as interfaces definidas do Retrofit em um pacote chamado *service*. Dessa forma, encontrar os métodos HTTP implementados com o Retrofit para o *back-end* da aplicação é uma tarefa simples. A análise desses métodos implementados permite também a compreensão sobre os dados que são retornados por cada chamada.

[5] https://docs.gradle.org/4.0.1/userguide/userguide.html

Primeiramente, para o modo verboso foi sempre atribuído o valor verdadeiro, pois o estudo prévio da aplicação e suas funcionalidades mostrou, como relatado anteriormente, que a falta de *feedbacks* para o usuário é um problema para a aplicação em cenários de uso *offline*.

Sobre os modos de tratamento escolhidos, diversas telas do sistema executam ações de forma transparente ao usuário – por exemplo carregamento de dados nas telas, carregamento de mais resultados nas listagens, dentre outros. A fim de manter essa característica, para essas telas e funcionalidades, o tratamento escolhido foi o modo transparente para dispositivo *offline*.

Um exemplo de utilização desse modo foi na tela inicial: antes da aplicação da biblioteca *OfflineManager*, ao abrir o aplicativo sem conexão, a tela inicial era mostrada vazia, pois sem conexão nenhum filme era mostrado e nenhum tratamento era aplicado. Além disso, quando a conexão era retomada, não existia meio de recarregar os dados da tela.

Depois de aplicar a biblioteca na tela inicial, utilizando o modo de tratamento transparente para dispositivo *offline* e para servidor indisponível, a funcionalidade continua transparente ao usuário porém melhorada. Os dados continuam não sendo exibidos caso a conexão não esteja disponível, mas o tratamento da biblioteca garante que eles sejam atualizados assim que a conexão for retomada.

O modo transparente foi utilizado também nas listas de filmes e séries de TV, acessíveis pelo menu lateral, pelos mesmos motivos da tela inicial. E ainda nessas listas de filmes e de séries de TV, ele foi utilizado para a paginação e carregamento de mais resultados. Quando o usuário navega para o fim da lista de itens carregados, se não houver conexão, uma chamada transparente será armazenada, e os resultados estarão disponíveis assim que a conexão retornar. Aqui, o modo transparente é bastante interessante pois permite uma navegação mais fluída pela lista, e a retomada de navegação automaticamente quando a conexão é retomada, sem a necessidade de o usuário realizar um recarregamento da tela.

Para a tela de detalhes de um filme, foi utilizado o modo flexível. Aqui, a tela oferece uma grande quantidade de informações referentes ao filme selecionado, e a interação é mais pontual, por apresentar os detalhes de um filme dentre os diversos disponíveis nas listagens. Por isso, uma confirmação do usuário antes de o tratamento ser aplicado pareceu ser mais adequada, justificando a escolha do modo flexível.

Assim, ao entrar na tela de detalhes, sem conexão com a Internet, o aplicativo verifica se aquele filme está armazenado em cache. Se sim, os detalhes serão exibidos de forma transparente ao usuário. Se não, uma requisição será necessária. Nesse ponto, o modo flexível interage com o usuário por meio de um pop-up para confirmar a intenção

em armazenar essa chamada e executá-la posteriormente. Pode ser que o usuário desista dessa ação, cancelando o tratamento, e prosseguindo para outras interações. Senão, pode ser que ele confirme o tratamento, pois se interessa pelo carregamento em segundo plano caso a conexão volte. Isso faz com que os dados estejam disponíveis em um segundo momento quando ele abrir o dispositivo móvel que foi deixado com a aplicação nesta tela.

É importante mencionar que o refinamento dos modos utilizados para cada ponto de interação pode ser ajustado de acordo com testes de usabilidade e validação pelos clientes finais da aplicação.

Ainda na tela de detalhes de filme foi aplicada uma chamada no modo transparente, para carregar a seção de vídeos, caso o aplicativo perca a conexão nesta tela, ou caso o usuário entre para ver os detalhes de um filme já em cache, enquanto o dispositivo está desconectado. Esta seção de vídeos não é estritamente necessária, portanto seu carregamento é feito em segundo plano, de forma transparente, caso a conexão seja retomada enquanto o usuário consome os detalhes de um filme selecionado.

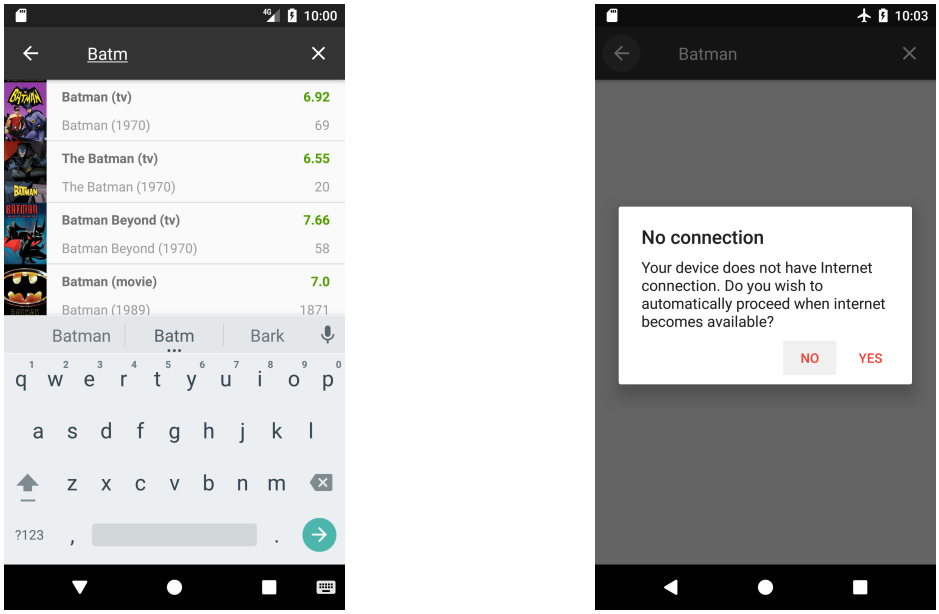
As últimas duas funcionalidades onde a biblioteca tratou cenários de utilização *offline* foram a pesquisa rápida, e a tela de "Discover". Na pesquisa rápida, uma pequena mudança no código da aplicação atribuiu ação ao botão de confirmar no teclado virtual, para realizar a chamada quando pressionado.

Outra mudança foi que, para o caso de o dispositivo estar sem conexão, a busca não ser executada a cada caractere inserido, mas somente quando o botão para buscar for pressionado. Isso para evitar que diversas chamadas sejam feitas e encadeadas durante o tratamento, sendo que o resultado da pesquisa a ser mostrado vai ser somente o da última chamada realizada, quando a conexão for retomada.

Dessa forma, o funcionamento da aplicação ficou da seguinte forma: com o dispositivo conectado, a busca ocorre a cada caractere pressionado, e os resultados vão sendo exibidos de acordo com a palavra que se forma. Caso não haja conexão, o usuário digita toda a frase que deseja pesquisar, e ao pressionar o botão de pesquisa, a biblioteca trata de forma flexível, informando ao usuário que a conexão não está disponível e perguntando se ele deseja que essa pesquisa seja executada em segundo plano com a retomada de conexão. As telas para esses dois casos são apresentadas na Figura 16:

Essas duas mudanças se mostraram necessárias para o bom funcionamento do tratamento em modo *offline*, inserido pela biblioteca *OfflineManager*. Principalmente, a mudança que evita o encadeamento das chamadas caso o dispositivo esteja sem conexão, evitando o armazenamento desnecessário de diversas chamadas intermediárias, antes da última, que teria seus resultados exibidos.

Além disso, essa última evidenciou o que pode ser um trabalho futuro para a biblioteca *OfflineManager*. Quando existir um cenário onde diversas chamadas são encadeadas,



(a) Conectado à Internet (b) Sem conexão com a Internet

Figura 16 – Telas do aplicativo Inventum que mostram a busca rápida com conexão disponível, e sem conexão disponível, tratadas pela biblioteca *OfflineManager*

e somente uma delas deve ser válida, a biblioteca poderia oferecer um tratamento para esse caso, de forma a descartar automaticamente as outras chamadas encadeadas. Poderia ser possível a configuração de um valor de intervalo máximo de tempo para que esse descarte ocorra, e também um modo de tratar qual a chamada seria a válida: a primeira, a última, ou alguma intermediária baseada em algum valor de retorno obtido pela chamada. Esse novo tratamento evitaria a necessidade de alteração no código da aplicação, como ocorreu neste trabalho.

Já na tela de "Discover" o tratamento foi bastante semelhante aos outros tratamentos aplicados. O modo de tratamento para dispositivo *offline* foi o flexível, com o modo verboso ativado para que haja tanto interação com o usuário, como *feedbacks* sobre a completude das ações. Portanto, se o usuário realizar uma pesquisa elaborada enquanto o dispositivo estiver sem conexão, ele pode interagir confirmando o armazenamento para que essa pesquisa seja executada e os resultados populados conforme o dispositivo retoma a conexão.

Nessa funcionalidade, é bastante interessante a aplicação da biblioteca e o tratamento da chamada pois esse é um fluxo que pode ser passível de frustração para o usuário. Pode ser que ele gaste um tempo pensando e elaborando os campos para a realização de uma pesquisa refinada, e ao submeter, a falta de conexão barra a continuidade do fluxo. Além disso, ele teria que ficar manualmente tentando novamente executar a pesquisa, até que ela seja possibilitada com a retomada de conexão.

Assim, pode-se se observar que a biblioteca agrega certo valor nesse ponto, pois ao

salvar essa requisição, ela livra o usuário de ter que tentar novamente por si, e também evita a frustração de ter que preencher os campos da pesquisa novamente. Os resultados estarão prontamente disponíveis uma vez que a conexão seja retomada.

Este cenário, então, pode ser expandido para uma análise mais ampla: aumenta o valor da utilização da biblioteca, em cenários onde o usuário interage elaborando pesquisas ou preenchendo formulários. Isso não somente em chamadas HTTP do tipo GET, que apenas requisitam dados do servidor, mas principalmente para chamadas HTTP do tipo POST, que enviam dados. Qualquer tipo de envio de formulários ou dados ao servidor, depois que o usuário utilizou e gastou tempo preenchendo e elaborando, pode ser encapsulado para que seja executado em um segundo momento, mesmo que a conexão não esteja disponível no momento do envio.

Junto a isso, mensagens de *feedback* informam ao usuário sobre os processos que estão acontecendo, para que ele sempre saiba o que está acontecendo na aplicação com os seus dados, com suas interações.

Por fim, mas não menos importante, todas as chamadas foram também tratadas para o caso do servidor não estar disponível, pois é um mecanismo que também está presente nos chamadas realizadas por meio da biblioteca, como foi explicado no capítulo 3. Para esses casos, porém, no contexto desse aplicativo, a escolha dos modos de tratamento caso o servidor encontre-se indisponível foi bastante direta. Para todas as chamadas, fez sentido a utilização do modo flexível.

Como nesta aplicação a indisponibilidade do servidor é um cenário bastante raro[6], é interessante haver interação do usuário para decidir se prossegue com tratamento *offline* ou não, caso isso aconteça.

O usuário será sempre informado também com *feedbacks* sobre ações realizadas devido ao modo verboso ativado. Apenas nas chamadas de carregamento de filmes, nas listas com paginação, que se utilizou o tratamento Sem Ação, pois entende-se que o cenário de servidor indisponível ou instável poderia atrapalhar o fluxo do carregamento das mensagens paginadas, portanto ali foi aplicado o tratamento que na verdade demanda disponibilidade do servidor, caso contrario o usuário será informado do problema e o carregamento será interrompido.

4.2.4 Avaliação do processo e considerações finais

Feito todo o processo de refatoração e implantação da biblioteca *OfflineManager*, tem-se o último passo da metodologia informalmente adotada, que é a avaliação sobre a aplicação final, considerando o requisito de funcionamento em modo *offline*. É claro que

[6] Em nenhum momento, durante o período de um mês de utilização da aplicação para essa avaliação, o *back-end* do TMDb ficou indisponível.

essa avaliação precisaria da validação de usuários reais, bem como validação do responsável pelo produto final a fim de, pelos processos de software, avaliar se o requisito foi alcançado de maneira satisfatória. No escopo desta pesquisa, uma avaliação inicial é feita do ponto de vista de desenvolvimento. Algumas considerações, por fim, são apresentadas sobre o processo de implantação da biblioteca.

Como foi discutido anteriormente, todas as chamadas que o aplicativo realizava foram tratadas com a biblioteca. A maioria delas utilizando o modo transparente para o cenário de dispositivo *offline*, e o modo flexível para o servidor *offline*.

Algumas chamadas, porém, como os detalhes de filme, série de TV ou pessoa, a pesquisa e o "Discover", utilizaram o modo flexível para o caso de dispositivo *offline*. Isso pois essas telas exigiam um comportamento mais refinado na questão de exibir mais informações, ou de possuir uma interação mais refinada, com preenchimento de dados para a posterior execução da chamada.

No geral, esses dois modos trataram bem o requisito de funcionamento *offline* quando o dispositivo perde conexão com a Internet. Por ser uma aplicação mais informal, de consulta de dados, no contexto de filmes e séries de TV, pode-se assumir que sua utilização é bastante intermitente e dependente da vontade do usuário em buscar alguma informação. Essa dependência da vontade do usuário significa que um típico usuário dessa aplicação a utiliza apenas quando sente vontade de interagir com ela, diferenciando-se de uma aplicação de cunho mais prático, que demanda uma utilização mais pragmática, como por exemplo aplicações bancárias. Devido a essa característica, os tratamentos para utilização em modo *offline* podem ser mais flexíveis e menos engessados, o mesmo acontece com o tratamento adotado para funcionamento em caso de servidor *offline*.

Em outras aplicações, porém, que necessitam de maior pontualidade na interação, ou até de maior segurança dos dados, é possível assumir que existiriam chamadas que utilizassem o modo *Enforce* (para dispositivo *offline*) ou *NoAction* (para servidor indisponível), cujas finalidades são fazer com que a chamada seja possível apenas no caso do dispositivo possuir conexão, ou o servidor estar disponível.

Já do ponto de vista do desenvolvedor, o processo de implantação foi bastante direto. A instalação da biblioteca é feita com apenas uma linha de código. A utilização da biblioteca para encapsular uma chamada do Retrofit também requer apenas algumas linhas de código para implementar.

Aproveita-se a criação da chamada com a interface do Retrofit, passando os parâmetros que serão os cabeçalhos da chamada, e utiliza-se o método fornecido pela biblioteca *OfflineManager* para a execução tratada desta chamada. No Inventum, pode-se aproveitar as mesmas lógicas implementadas nos métodos de *callbacks*, sendo necessário apenas a adição de um *cast* explícito pois o *callback* oferecido pela biblioteca é genérico, enquanto

o oferecido pelo Retrofit já é tipado.

Outro fator bastante simples é a sobrescrita das mensagens de *feedback*. A documentação da biblioteca *OfflineManager* oferece um guia bastante claro sobre quais *strings* devem ser sobrescritas para alterar o texto de cada mensagem mostrada pela biblioteca. Portanto, bastou uma leve ponderação sobre quais mensagens seriam alteradas, uma rápida procura na documentação para saber as chaves que identificavam essas mensagens, e a sobrescrita no arquivo de *strings* da aplicação.

Por fim, como considerações finais, é possível afirmar que, por métricas de código, o processo de implantação da biblioteca em um projeto real de aplicativo, é bastante direto e rápido:

- 5 atividades alteradas, sendo que para cada uma delas apenas uma inicialização da biblioteca *OfflineManager* é necessária;
- 5 fragmentos alterados, sendo que para cada um deles apenas uma inicialização da biblioteca *OfflineManager* é necessária;
- Dentro dessas atividades e fragmentos, um total de 13 chamadas do Retrofit foram alteradas para serem tratadas com a biblioteca;
- Cada alteração representa uma chamada ao método `treatedCall`, e foi implementada por uma média de 20 linhas de código. Isso sem considerar a lógica da aplicação implementada nos métodos de *callback* do Retrofit, que foi reaproveitada para os *callbacks* do método `treatedCall`;
- 2 linhas necessárias para adicionar a biblioteca *OfflineManager* ao projeto, por meio do Graddle;

Porém, o grande desafio desse processo se encontra nas etapas de:

- Identificação e avaliação do requisito de funcionamento em modo *offline* da aplicação;
- Escolher os tratamentos a serem adotados em cada chamada, em caso de dispositivo sem conexão;
- Escolher os tratamentos a serem adotados em cada chamada, em caso de servidor indisponível;
- Decisão sobre a utilização do modo verboso, que depende da usabilidade desejada e do contexto da aplicação e da chamada em específico;

- Refatorações necessárias em outros trechos de código, para que o fluxo da aplicação seja adaptado ao tratamento oferecido pela biblioteca.

Contudo, já se esperava que essas etapas se mostrassem desafiadoras, uma vez que se relacionam com o projeto da aplicação. Assim, essas decisões dependem bastante do contexto em que o aplicativo será inserido, seu objetivo principal e seu público alvo. Para a aplicação no Inventum, pouca refatoração de código foi necessária para a adaptação aos fluxos fornecidos pela biblioteca *OfflineManager*, mas esse fator, em outras aplicações, vai depender de certa forma do projeto inicial da aplicação e do estágio em que o requisito de funcionamento *offline* vai ser inserido.

4.2.5 Protodiretrizes para a escolha dos modos de tratamento

Algumas justificativas para a escolha dos modos de tratamento utilizados foram dadas anteriormente, durante a apresentação do processo de refatoração do aplicativo Inventum. Nesta seção, é consolidado e apresentado um conjunto de protodiretrizes adotadas para a escolha dos modos de tratamento. Primeiramente são apresentadas para o modo de tratamento em caso de dispositivo *offline*. Essa escolha baseia-se na funcionalidade da aplicação onde a chamada é feita, e também em quesitos de usabilidade para a tela em questão.

Protodiretriz 1 – tela de informação detalhada: Para telas onde são apresentados detalhes de uma entidade na aplicação, no caso filmes, séries de TV ou pessoas, foi utilizado o modo flexível. São telas onde a interação é pontual e bem definida (usuário deseja saber informações sobre aquele item em específico), e não uma simples interação exploratória com a aplicação. Além disso, é uma funcionalidade que traz muitas informações para o usuário, e pode consumir recursos como bateria, processamento ou banda de internet. Por esses motivos, uma ação de confirmação do usuário antes de prosseguir com o tratamento é justificada.

Protodiretriz 2 – funções baseados em preenchimento de dados: Essa protodiretriz generaliza qualquer funcionalidade que requisita ou envia dados, baseados em um preenchimento de formulário feito pelo usuário. Ou seja, no caso do aplicativo Inventum, representa os dois modos de busca presentes: a busca rápida e o "Discover". Em ambos os casos o usuário precisa preencher informações para que a busca seja realizada. Para essa situação, o modo de tratamento escolhido foi o modo flexível. São funções que dependem bastante do momento e contexto de utilização. Neste caso, as buscas podem ser diferentes dependendo do que motivou o usuário a realiza-las, em determinado momento. Assim, justifica-se uma confirmação do usuário antes que a aplicação prossiga com o tratamento em modo *offline*: pode ser que

o usuário confirme o tratamento para ver os resultados posteriormente, quando a conexão for retomada. Mas pode ser que o usuário simplesmente cancela a busca e desista da interação naquele momento.

Protodiretriz 3 – carregamentos secundários de dados: Este cenário representa carregamentos secundários de dados. São dados que não são essenciais para o fluxo de utilização da aplicação, mas enriquecem a experiência uso e os dados disponíveis. No caso do Inventum, é a seção de vídeos, presente na tela de detalhes de filmes ou séries de TV. É uma seção que apresenta vídeos e trailers disponíveis para o filme ou série de TV em questão. Para esse caso, foi escolhido o modo transparente, pois por serem dados secundários eles podem ser tratados de forma transparente, e mostrados somente quando estiverem disponíveis. Não há necessidade de desviar a atenção do usuário das ações principais, com confirmações e *feedbacks* para dados secundários.

Protodiretriz 4 – carregamento de dados para telas principais: Essa protodiretriz representa de forma geral, chamadas que carregam dados para popular telas principais do sistema. No caso do Inventum, representa as chamadas nas telas de listagem de filmes e séries de TV, na tela inicial, e na lista de vídeos disponíveis nos detalhes de um filme. São funcionalidades principais do sistema, e em caso de conexão indisponível, não são apresentadas. Aqui, o tratamento adotado depende exclusivamente do contexto da aplicação, e do tipo de dado que é carregado, como ele é carregado, e como é apresentado. Para o caso do Inventum, são listas de filmes ou séries de TV, e são vários carregamentos: na tela inicial são executados 2 para trazer a lista horizontal de filmes e a de series de TV. Nas telas de listagem de filmes ou de séries de TV, são executadas 4 carregamentos para disponibilizar as listas com ordenações diferentes (Populares, Mais recentes, etc). Por causa desses carregamentos serem simultâneos e encadeados, e os dados serem listas mostradas em partes diferentes da tela, foi adotado o modo transparente de tratamento. Isso faz com que não sejam necessárias repetidas confirmações por parte do usuário. Assim, *feedbacks* quando as chamadas forem completadas já são suficientes para manter o usuário informado, permitindo que ele retome a utilização da aplicação em um segundo momento, onde os dados já estarão carregados. Caso o contexto da aplicação fosse diferente, e essas chamadas se aproximassem mais da protodiretriz 1, o modo escolhido seria o flexível. Por fim, pode ser que o contexto da aplicação exija conexão pelo menos para as telas principais, onde se concentram os dados principais do sistema. Nesse caso, o modo escolhido seria o *Enforce*.

A seguir, são mostradas as protodiretrizes para a escolha do modo de tratamento no caso de servidor indisponível. Como foi discutido no capítulo 3, esse tratamento refere-

se não tanto sobre a chamada em si sendo executada, mas também é escolhido por base nas características ou estatísticas do servidor em questão.

Um servidor estatisticamente mais intermitente demanda mais utilização de tratamentos semelhantes aos escolhidos para o caso de dispositivo *offline*. Isso para que a usabilidade seja consistente, e haja transparência para o usuário, que não deve se preocupar se a função está indisponível por causa da conexão com a Internet ou por causa do servidor estar indisponível. Portanto, nesse cenário, é possível concluir que o tratamento para o caso de servidor indisponível, deve se aproximar do tratamento escolhido para o caso de dispositivo *offline*.

Por outro lado, para servidores que raramente encontram-se indisponíveis, pode ser interessante que o usuário, além de ser avisado dessa indisponibilidade, possa controlar se deseja que sua ação seja tratada ou não. Isso para que o usuário consiga distinguir esse cenário esporádico, dos tratamentos mais frequentes para casos de dispositivo *offline*, e assim decida de forma adequada sobre prosseguir ou não com o armazenamento e novas tentativas de sua chamada.

Portanto, as protodiretrizes propostas para escolha do tratamento, em caso de servidor indisponível, são duas:

Protodiretriz 5 – servidor intermitente: Quando o servidor intermitente, o modo de tratamento escolhido deve ser o mesmo do modo escolhido para tratamento em caso de dispositivo *offline*;

Protodiretriz 6 – servidor raramente indisponível: Quando o servidor raramente torna-se indisponível, o modo escolhido de tratamento deve ser o flexível, para que o usuário seja capaz de diferenciar esse cenário e confirmar ou não o tratamento, dado o contexto da chamada que foi realizada.

Esse conjunto proposto de protodiretrizes é assim denominado pois foram feitas com base em uma única aplicação, portanto as decisões sobre os modos escolhidos, e os raciocínios adotados para cada um são restritos a esse cenário. Com a aplicação e testes em mais aplicações diferentes, esse conjunto poderia ser validado e ajustado até se tornar um conjunto consolidado de diretrizes para a escolha dos modos de tratamento. Isso pode ser considero um possível trabalho futuro.

4.2.6 Discussão

Retomando novamente o objetivo desta pesquisa e os dois pontos críticos citados no início deste capítulo, observa-se que, neste processo de adição da biblioteca *Offline-Manager* ao aplicativo Inventum, ambos foram atendidos satisfatoriamente.

Além de adicionar a possibilidade de uso em modo *offline* da aplicação, os modos de tratamento escolhidos, em conjunto com o modo verboso, inseriram interações com o usuário final e *feedbacks* que resolveram problemas de usabilidade apresentados inicialmente. Melhoraram a experiência de uso em algumas telas como as listagens e a tela inicial, realizando o carregamento automático quando a conexão é retomada.

Permitiram também a realização de pesquisas em modo *offline*, que são executadas posteriormente quando a internet torna-se disponível. Tudo isso fornecendo *feedbacks* de forma a manter o usuário informado sobre as ações realizadas.

Assim, com relação ao primeiro ponto crítico, pode-se considerar que o requisito de funcionamento *offline* foi adicionado com sucesso e de forma a englobar todas as camadas da aplicação.

Ademais, a adição da biblioteca *OfflineManager* ao aplicativo Inventum foi feita em um estágio avançado do seu processo de desenvolvimento. As funcionalidades já haviam sido implementadas por completo, e o código já disponibilizado na plataforma do GitHub. Mesmo assim, alterações mínimas sobre a aplicação foram necessárias. Foram feitas apenas a fim de eliminar redundantes tratamentos ou mensagens de *feedback*, ou de melhorar algumas telas que passaram a ser tratadas com a biblioteca.

Portanto, com relação ao segundo ponto crítico, isso mostra que o esforço necessário para a implementação do requisito de funcionamento *offline* no aplicativo Inventum, por meio da biblioteca *OfflineManager*, foi reduzido, mesmo com o sistema já pronto.

Essa versão final do Inventum, após o tratamento apresentado nesta seção, com a biblioteca *OfflineManager*, será utilizado na atividade de avaliação a ser apresentada posteriormente. Nela, um experimento comparativo foi realizado com sujeitos que avaliaram o Inventum sem a aplicação da biblioteca *OfflineManager*, e com a biblioteca aplicada.

4.3 Experimento: aplicação com e sem a biblioteca em um cenário de uso *offline*

O terceiro mecanismo de avaliação adotado foi um experimento de comparação entre um aplicativo sem a utilização da biblioteca, com o mesmo aplicativo utilizando a biblioteca *OfflineManager*, em cenários diferentes de utilização, sob o ponto de vista do usuário final. A aplicação utilizada foi o Inventum, mesmo aplicativo usado na avaliação anterior.

Nesta avaliação, os sujeitos do experimento puderam ao final responder a um questionário avaliando conceitos de usabilidade de ambas as aplicações, e comparando os diferentes aspectos encontrados nos fluxos que envolviam a intermitência de conexão. Neste sentido, esta avaliação visou determinar se a solução provida pela biblioteca, ao passar

por todas as camadas e chegar ao *front-end*, atinge um nível adequado de usabilidade.

Para a realização desse experimento primeiramente foram selecionados alguns candidatos, com diferentes características e ocupações: um candidato do sexo masculino, que trabalha como gerente de projetos e dois candidatos do sexo feminino, um trabalha como analista de testes, e outro como gerente de qualidade. Esses candidatos foram abordados, e concordaram em participar do experimento que teve duração média de 30 minutos para cada participante. Foi executado um experimento por vez, em momentos diferentes do dia, com um sujeito por vez.

Cada execução envolveu a utilização de duas versões do aplicativo: o aplicativo Inventum original, sem a biblioteca (aplicativo 1), e o aplicativo Inventum modificado, com a biblioteca (aplicativo 2).

Além desse processo inicial, o experimento foi definido em quatro fases simples:

1. Treinamento sobre a aplicação, para familiarização com suas funcionalidades e telas, sem ocorrer nenhum erro por falta de conexão ou inatividade do servidor;
2. Execução de fluxos onde a Internet encontrava-se intermitente ou indisponível, no aplicativo 1;
3. Execução de fluxos onde a Internet encontrava-se intermitente ou indisponível, no aplicativo 2;
4. Submissão de questionário avaliativo para comparação entre as duas aplicações.

O treinamento foi feito pelo autor desta dissertação, e consistiu de uma explicação sobre todas as telas do aplicativo, bem como uma explicação sobre outras características do sistema, como contexto da aplicação e seu objetivo principal. Foi explicitado que o contexto do Inventum é o de uma aplicação de consulta de dados voltado para filmes e séries de TV, e não requer cadastro ou *login*. Seu objetivo é disponibilizar informações sobre filmes e séries de TV para o usuário, bem como listas dinâmicas dos mesmos, ordenadas de diversas formas como por exemplo: mais recentes ou mais populares. Além disso, disponibiliza informações sobre atores e também fornece funcionalidade de busca.

Para a explicação das telas, o sujeito recebeu um dispositivo Android de última geração com o aplicativo 1 instalado, e foi instruído pelo autor desta dissertação, que acompanhou o treinamento, a percorrer tela por tela do sistema, seguindo uma ordem pré-definida. Essa ordem foi previamente definida pelo autor desta dissertação e simplesmente segue um fluxo de forma a percorrer todas as telas do sistema. Para cada tela, foram explicadas suas funcionalidades para que o próprio sujeito pudesse executar essas funcionalidades no dispositivo, de forma a compreender seu funcionamento, os resultados esperados e os possíveis erros.

Ao final da explicação das telas, o sujeito ainda pode tirar dúvidas remanescentes, e também executar fluxos livremente, de forma a se familiarizarem com a aplicação. É importante mencionar que o treinamento foi feito exclusivamente com a Internet conectada.

Após o treinamento, o sujeito, que já estava em posse de um dispositivo Android com o aplicativo 1 instalado, recebeu também um dispositivo Android, do mesmo tipo e mesmas características, porém com a aplicação 2 instalada. Em seguida, foi instruído a utiliza-los seguindo o roteiro pré-definidos, de forma a executar as próximas fases do experimento.

4.3.1 Fluxos executados no experimento

Para que fosse possível uma comparação entre os dois aplicativos, definiu-se um roteiro base para o usuário seguir durante o experimento. O principal objetivo desse roteiro foi fazer com que os fluxos de funcionamento ficassem evidentes, a fim de possibilitar uma comparação deste cenário entre uma aplicação onde não existe tratamento para funcionamento em modo *offline*, e uma aplicação tratada se utilizando da biblioteca proposta neste trabalho de mestrado.

É importante mencionar que apesar de um roteiro pré-definido ter sido estabelecido, ele não foi restritivo. O usuário no experimento ainda tinha liberdade para experimentar funcionalidades de acordo com sua vontade, permitindo que a experiência de uso fosse a mais natural possível. O roteiro proposto foi explicado verbalmente para os sujeitos, pelo autor desta dissertação, no início de sua execução. Durante sua execução ele foi recapitulado, de forma a garantir que fosse finalizado. Além disso, esse roteiro proposto foi também disponibilizado de forma textual no questionário de avaliação enviado aos sujeitos, no final do experimento.

Dessa forma, o roteiro proposto seguiu, por base:

1. Abrir a aplicação
2. Observar as listas de Filmes e TV Series, apresentadas na tela inicial
3. Selecionar um filme para ver mais detalhes
4. Na tela de detalhes, clicar em Vídeos para ver lista de vídeos disponíveis sobre aquele filme
5. Voltar para a tela inicial
6. Repetir passos 3-6 para uma Série de TV, selecionada na tela inicial

7. Clicar no ícone de pesquisa no canto superior direito, e pesquisar por o nome de algum filme
8. Nos resultados, selecionar o filme para ver detalhes
9. Voltar para a pesquisa e pesquisar o nome de algum ator
10. Nos resultados, selecionar um ator para ver detalhes
11. Voltar para a tela inicial
12. Pelo menu lateral, navegar para a seção *Movies* (Listagem de Filmes)
13. Rolar as listas para observar o carregamento automático de mais resultados
14. Pelo menu lateral, navegar para a seção *TV Shows* (Listagem de Séries de TV)
15. Rolar as listas para observar o carregamento automático de mais resultados
16. Pelo menu lateral, navegar para a tela "Discover"
17. Preencher um conjunto de dados e submeter a busca
18. Observar os resultados da busca

Esse roteiro percorre basicamente todas as telas do aplicativo Inventum. No entanto, o que interessa para o experimento são os cenários de utilização em modo *offline*. Portanto, para que isso pudesse ser observado e comparado, o roteiro real seguiu os passos descritos acima, porém entre cada um deles a Internet foi desconectada, e a funcionalidade referente ao passo foi executada em modo *offline*. O processo de desconectar a Internet foi feito de forma manual, onde os sujeitos eram instruídos a desabilitar a conexão do dispositivo.

Além de ser executada em modo *offline*, foi simulado também para os passos acima, uma retomada de conexão enquanto o aplicativo ainda estava rodando, na tela em questão. Por fim, foi simulado também um cenário de retomada de conexão, porém com o aplicativo Inventum em segundo plano, com outro aplicativo sendo utilizado ou dispositivo bloqueado e sem ser utilizado. Para isso, os sujeitos eram instruídos a bloquear a tela do dispositivo ou abrir outra aplicação qualquer instalada no dispositivo, antes de retornarem a conexão. Assim, um exemplo de execução real do roteiro proposto, para cada funcionalidade descrita, seguiria os seguintes passos:

1. Internet desconectada e dispositivo fica sem conexão;
2. Tela iniciada ou aberta, ou funcionalidade executada em modo *offline*;

3. Internet é retomada – aplicativo estando em primeiro plano, aplicativo estando em segundo plano com outro aplicativo aberto ou aplicativo estando em segundo plano com o dispositivo bloqueado.

4. Se o aplicativo estiver em segundo plano e/ou o dispositivo estiver bloqueado, desbloquear o dispositivo e/ou voltar o aplicativo para primeiro plano

5. Observar a tela ou funcionalidade em questão;

Essa execução real foi feita para as funcionalidades descritas nos passos 1, 3, 4, 6, 7, 9, 10, 12, 13, 14, 15 e 17, do roteiro proposto.

O terceiro passo da execução real descrita, onde há variações no sentido de o aplicativo estar em primeiro plano, segundo plano e/ou dispositivo bloqueado, foi executado de forma arbitrária para as funcionalidades do roteiro proposto. Ou seja, para algumas funcionalidades a conexão foi retomada com a aplicação em primeiro plano, para outras, com a aplicação em segundo plano e outro aplicativo sendo executado, e por fim, algumas funcionalidades foram executadas de forma a retomar a conexão com o dispositivo bloqueado.

Também, algumas funcionalidades foram executadas mais de uma vez, de forma a verificar o comportamento em mais de um cenário de retorno.

A ideia é que os sujeitos executem as funcionalidades do roteiro com diferentes estados durante o retorno, ou até a mesma funcionalidade mais de uma vez com estados diferentes, para conhecerem a experiência de uso nesses três casos diferentes (aplicativo em primeiro plano, aplicativo em segundo plano com outra aplicação aberta e dispositivo bloqueado). A funcionalidade específica que foi executada em cada caso, não pareceu ser um fator de influência no experimento.

Esse roteiro proposto, da forma como foi descrito, e as características desse processo de execução foram adotadas para ambos os aplicativos: o aplicativo 1, sem a biblioteca *OfflineManager* aplicada, e o aplicativo 2, com a biblioteca aplicada. Assim, os sujeitos conheceram os modos como cada aplicativo se comporta nos cenários de uso *offline*, para responderem, na próxima etapa, um questionário de forma a comparar as aplicações.

4.3.2 Questionário de avaliação e resultados obtidos

Após a execução do experimento, comparando os roteiros de execução com fluxos em modo *offline* e retomada de conexão, em ambos os aplicativos, foi pedido para que os sujeitos submetessem um questionário avaliando os dois aplicativos, a fim de colher dados sobre a experiência de usuário de cada um deles. O roteiro executado seguiu os passos descritos seção anterior, onde a Internet era desconectada e reconectada para simular os cenários de utilização em modo offline. Ele foi executado tanto no aplicativo 1 (aplicativo

Inventum sem aplicação da biblioteca *OfflineManager*), quanto no aplicativo 2 (aplicativo Inventum com aplicação e tratamento pela biblioteca *OfflineManager*).

As três primeiras perguntas do questionário são para caracterizar os sujeitos e questionam: área de trabalho, descrição da atuação principal, e "Em uma escala de 0 a 10, quão presente são os *smartphones* em sua vida cotidiana". Assim, tem-se os seguintes perfis:

Sujeito 1: Essa participante trabalha em empresa, na posição de gerente de qualidade. Atribuiu nota 10 para a presença de *smartphones* em sua vida cotidiana, em uma escala de 0 a 10;

Sujeito 2: Esse participante trabalha em empresa, na posição de gerencia de projetos. Atribuiu nota 8 para a presença de *smartphones* em sua vida cotidiana;

Sujeito 3: Essa participante também trabalha em empresa, na posição de analista de testes. Atribuiu nota 9 para a presença de *smartphones* em sua vida cotidiana;

A terceira questão sobre a presença de *smartphones* no cotidiano dos sujeitos é uma informação importante para saber a relevância que interações com aplicativos tem na vida dos sujeitos do experimento. As respostas fornecidas (10 para o sujeito 1, 8 para o sujeito 2 e 9 para o sujeito 3) indicam que os participantes utilizam com grande frequência *smartphones* e seus aplicativos. Assim, pode-se dizer que esses sujeitos saberão avaliar diferenças de utilização em modo *offline* dos aplicativos, devido ao grande contato com aplicativos em geral e com os diferentes cenários a que são submetidos durante seu uso diário.

A quarta pergunta do questionário avalia: "Com que frequência você se encontra impossibilitada de usar alguma aplicação, ou parte dela, por fala de conexão com a internet?". Todos responderam que enfrentam problemas com frequência média, onde as opções eram: nunca, pouca frequência, frequência média, frequentemente, e sempre. Isso indica, neste caso, que para esses sujeitos, falta de conexão não é um evento raro e existe em seus cotidianos. As próximas perguntas se referem ao aplicativo Inventum, e ao experimento proposto.

A quinta questão avalia, primeiramente, se os sujeitos compreenderam as funcionalidades do aplicativo: "Navegando pela aplicação pela primeira vez, com a internet sempre conectada, como foi a compreensão do aplicativo e suas funcionalidades?". Todos responderam que compreenderam perfeitamente.

A sexta questão, ainda sobre a aplicação e sem entrar no contexto de utilização *offline*, avalia a usabilidade da aplicação, em um cenário onde ela está sempre conectada, ou seja, usabilidade nos fluxos normais de sua utilização. A pergunta exata foi: "Ainda

nessa primeira navegação [com a internet sempre conectada], como você avaliaria a experiência de usuário da aplicação?". Aqui as respostas foram diversificadas: o sujeito 1 respondeu que a usabilidade é ruim, o sujeito 2, que a usabilidade é boa, e o sujeito 3 respondeu que a usabilidade é média.

Depois de avaliarem sobre a usabilidade em um cenário sempre conectado, a sétima questão foi: "Entendendo o objetivo da aplicação, como você avaliaria o requisito de funcionamento *offline* nela?", para que os sujeitos avaliassem se o requisito de funcionamento *offline* agregaria valor para a aplicação. O sujeito 1 entendeu que sim, agrega certo valor, e os sujeitos 2 e 3 entenderam que agrega bastante valor. Esse resultado indica que o aplicativo é válido para esse experimento, pois o requisito de funcionamento *offline* é aplicável.

Sobre o roteiro de execução para o experimento, a oitava questão indaga: "Como foi a compreensão e execução do roteiro offline no aplicativo 1?". Aplicativo 1 é o aplicativo Inventum sem tratamento para uso em modo *offline*, apenas com os tratamentos desenvolvidos inicialmente em seu projeto. Esse roteiro se trata do roteiro especificado na seção anterior, incluindo falta de conexão com a Internet e retomada de conexão durante a execução das funcionalidades. Os sujeitos 1 e 3 responderam que compreenderam bem o roteiro, e o sujeito 2 compreendeu muito bem. Isso indica que não houve dificuldades no entendimento dos fluxos executados.

Tendo compreendido o roteiro do experimento, a nona questão foi uma pergunta aberta: "Liste os problemas encontrados na execução desde roteiro". O sujeito 1, único a listar problemas específicos, disse que o principal problema na execução foi a falta da funcionalidade "*pull to refresh*" – arrastar para atualizar. Essa funcionalidade é uma característica que atualmente é bastante comum em aplicativos que utilizam dados da Internet, e funciona de forma a permitir o usuário utilizar um gesto de arrastar a tela para baixo com o dedo para iniciar uma atualização da página com os dados do servidor.

De fato, como o sujeito 1 observou, essa funcionalidade melhoraria a experiência de uso em algumas telas. Com ela, quando dados não fossem carregados por falta de conexão, o usuário poderia em um segundo momento iniciar uma atualização da tela, a fim de carregar os dados e continuar a utilização da aplicação quando a conexão fosse retomada. É uma funcionalidade que não diz respeito ao contexto de utilização em modo *offline*, mas que agrega valor para esse cenário, melhorando a usabilidade no sentido de permitir ao usuário corrigir possíveis problemas de carregamento.

Outra observação feita pelo sujeito 1 é que alguns pequenos problemas presentes no aplicativo incomodam a execução dos fluxos propostos, apesar de compreender que não são o foco do estudo. De fato, a aplicação da biblioteca não visou corrigir esses problemas, que não se relacionam com os cenários de utilização em modo *offline*.

Em seguida, ainda sobre a execução do roteiro proposto, simulando cenários de falta de conexão, a décima questão, objetiva, diz: "Como você avaliaria, do ponto de vista de experiência de usuário, o funcionamento *offline* do aplicativo 1?". Todos afirmaram que o aplicativo 1 possui usabilidade ruim.

A décima primeira questão, aberta, pede: "Liste, se houver, os problemas encontrador na experiência de usuário, no funcionamento offline da aplicativo 1". O sujeito 1 listou os seguintes problemas:

- Falta do "*pull to refresh*" para forçar atualização da página, impedindo atualizar os dados na tela em um segundo momento onde a conexão pode ter voltado;
- A tela inicial do aplicativo fica vazia quando o aplicativo é aberto sem conexão com a Internet, sugere que um aviso melhor sobre o problema fosse exibido;
- Falta mensagens de feedback sobre falta de conexão em algumas telas, como na pesquisa rápida;
- Em algumas telas existe duplicação de feedback sobre falta de conexão: aparece mensagem tanto no rodapé quando na tela principal;
- Botão na tela de "Discover" que não muda de texto se o usuário estiver buscando filmes ou séries de TV. O texto é sempre "*search movies*" (Pesquisar filmes). Esse ponto, no entanto, não se relaciona com fluxos de utilização *offline*. É na verdade um problema do aplicativo Inventum.
- Aplicativo não mostra de nenhuma forma que as listas de filmes ou séries de TV, onde há paginação, que os dados são carregados de vinte a vinte itens. Nota-se que este ponto também se refere a uma lógica de negócios do próprio aplicativo, saindo também do contexto de utilização em modo *offline*.

Já o sujeito 2 apenas listou como problema que "o aplicativo se torna praticamente inútil quando não há conexão com a Internet". Indicando que não conseguiu utilizar nenhuma funcionalidade de forma satisfatória quando não havia conexão com a Internet. O sujeito 3 listou os seguintes problemas:

- Não era possível ver detalhes de filmes que não tinham sido previamente abertos;
- Caso a tela de detalhes de filme estivesse aberta no momento de retomada de conexão, era necessário que o processo de atualização fosse iniciado pelo usuário (não era automático);

- Além da tela de detalhes de filmes, outras telas como detalhes de séries de TV ou listagens, quando volta a conexão, os dados não são atualizados de forma automática;

- Problema de interface em uma tela onde uma mensagem de erro sobrepõe informações previamente carregadas na tela.

Dessa listagem é possível tirar algumas conclusões interessantes: os principais problemas de usabilidade em um cenário *offline*, para essa aplicação, são a falta de *feedbacks* adequados de forma a melhor instruir o usuário (problema listado por dois sujeitos), e a falta de um mecanismo que atualize os dados automaticamente, pois quando a conexão é retomada, o usuário além de não ser avisado, precisa manualmente forçar uma atualização dos dados. Essa atualização de dados era feita de forma a sair da tela em questão ou fechar o aplicativo, e voltar novamente para àquela tela.

Após listarem os problemas encontrados, a décima segunda questão pede: "Liste, se houver, pontos positivos na experiência de usuário, no funcionamento offline da aplicativo 1". O sujeito 1 listou apenas que, ao entrar na seção filmes/séries de TV pelo menu lateral, as quatro abas que ordenam os dados de forma diferente são carregadas, o que permite uma navegação transparente por elas. O sujeito 2 não conseguiu listar nenhum ponto positivo, e o sujeito 3 ressalta a funcionalidade de *cache*, que agrega valor pois permite visitar filmes previamente visitados, mesmo sem conexão com a Internet.

Esses pontos positivos de fato são interessantes, principalmente a ressalva sobre o valor agregado pelo mecanismo de *cache*. Apesar de não interagir de forma direta na experiência de usuário da aplicação, pois não informa o usuário de nenhuma forma o que foi ou não armazenado, ele interage de forma indireta, pois permite uma navegação mais transparente, desde que aquele dado tenha sido visitado previamente.

A décima terceira questão parte para a coleta de dados sobre o aplicativo 2, que é o mesmo aplicativo Inventum, porém com a biblioteca *OfflineManager* aplicada. Pergunta: "Como foi a compreensão e execução do roteiro offline no aplicativo 2?". Trata-se do mesmo roteiro proposto descrito na seção anterior, e utilizado no aplicativo 1.

Os sujeitos forneceram as mesmas respostas que forneceram sobre compreensão do roteiro para o aplicativo 1: os sujeitos 1 e 3 compreenderam bem, e o sujeito 2 compreendeu muito bem. De fato, as respostas foram consistentes em vista que o roteiro proposto é o mesmo, para que comparações sejam possíveis entre sua execução no aplicativo 1 e no aplicativo 2.

Em seguida, a décima quarta questão pede: "Liste os problemas encontrados na execução desde roteiro". Refere-se a execução do roteiro proposto no aplicativo 2. Nenhum sujeito listou problemas encontrados. Isso mostra que a execução do roteiro na aplicação

2 apresentou menos pontos de dificuldades, mesmo não implementando a funcionalidade de "*pull to refresh*", problema que foi citado na execução do roteiro na aplicação 1.

A décima quinta questão pergunta: "Como você avaliaria, do ponto de vista de experiência de usuário, o funcionamento *offline* do aplicativo 2?". Para essa pergunta, todos os sujeitos responderam que consideram a experiência boa, dentre as opções: muito ruim, ruim, média, boa e muito boa. Essa avaliação contrasta com a avaliação sobre o aplicativo 1, onde todos responderam que foi ruim. As duas próximas perguntas, abertas, pediram para os sujeitos listarem os problemas encontrados no âmbito de experiência de usuário e depois os pontos positivos, se houverem.

A décima sexta questão pede: "Liste, se houver, os problemas encontrador na experiência de usuário, no funcionamento offline da aplicativo 2". Para a os problemas encontrados, os sujeitos listaram os seguintes problemas: o sujeito 1 relatou o botão na tela de "Discover" que não muda o texto referente a qual pesquisa está sendo feita – filmes ou séries de TV; e o fato de o aplicativo não informar que o carregamento por paginação das listagens, quando existe, é feito de 20 em 20 itens. Esses problemas, porém, como pode ser observado, não dizem respeito ao quesito de funcionamento *offline* em si, e acabam por sair do escopo tratado pela biblioteca. Para resolvê-los, seria necessária uma refatoração na própria aplicação a fim de corrigir ou melhorar esses problemas.

O sujeito 2 relatou: "Eu entendo que isso é configurável mas mesmo assim compensa mencionar: a experiência seria melhor se eu não precisasse pedir ao aplicativo para que ele recarregasse o conteúdo ao reconectar à Internet.". No caso, o que o sujeito diz com "precisar pedir para o aplicativo que ele recarregasse o conteúdo ao reconectar à Internet" refere-se ao método de tratamento oferecido pelo modo flexível, explicado no capítulo 3, onde é oferecido um pop-up no qual o usuário pode escolher prosseguir ou não com o tratamento *offline*.

Em sua visão, o sujeito 2 não considera necessário esse ponto de interação para o tratamento em modo *offline*, que foi aplicado principalmente na funcionalidade de ver detalhes de filme, no aplicativo Inventum. Por essa opinião, é discutível se o modo de tratamento flexível escolhido para essa funcionalidade foi o mais adequado, no contexto da aplicação Inventum, de navegar por filmes e séries de TV. É mais provável que seja um desajuste do modo de tratamento escolhido, guiado pela protodiretriz 1 explicada na seção 4.2, do que um indicador de falta de necessidade do modo flexível como um todo.

Ainda, essa foi a opinião apenas do sujeito 2. Portanto, opiniões semelhantes teriam que ser observadas e o experimento repetido com outros sujeitos, antes que conclusões sejam feitas sobre o acerto do modo escolhido para a funcionalidade em questão, e a validade da protodiretriz 1 que guiou essa escolha.

A décima sétima questão, sobre os pontos positivos observados: "Liste, se houver,

pontos positivos na experiência de usuário, no funcionamento offline do aplicativo 2". O sujeito 1 relatou:

- O "pull to refresh" não foi necessário, pois o aplicativo tratou e carregou o conteúdo automaticamente com a retomada de conexão;
- Mensagens sempre estavam avisando o usuário sobre a falta de conexão e sua retomada, mesmo com o aplicativo em segundo plano;
- Oferecer opção para o usuário escolher se quer ou não que o conteúdo seja carregado automaticamente (confirmação oferecida pelo modo flexível). Neste ponto, acrescenta: "O que é muito bom (poder escolher se quer ou não que o conteúdo seja carregado) quando o usuário está usando rede de dados móveis e tem o costume de deixar muitas aplicações ativas no celular".

Já o sujeito 2 apenas listou que o fato de o conteúdo ser atualizado automaticamente quando a Internet é reestabelecida, "já é um grande avanço em relação ao aplicativo 1". Ou seja, entende que esse recurso já agregou melhoria sobre o fluxo do aplicativo 1, onde não havia nenhum tipo de tratamento para uso em modo *offline*.

O sujeito 3 listou os seguintes pontos positivos:

- O pop-up oferecido pelo modo de tratamento flexível, que pergunta se o usuário deseja ou não seguir com o tratamento *offline*, é interessante por "permitir a liberdade do usuário";
- Quando a conexão volta, os dados são carregados automaticamente, ou não, de acordo com a opção a confirmação do pop-up para aquela tela;
- Outras telas oferecem o carregamento automático por padrão, quando a conexão é retomada;
- Nas telas de detalhes de filme/série de TV, mesmo que já abertos anteriormente e exibidos quando o dispositivo está *offline* por estar armazenado em *cache*, a seção de vídeos não é carregada. Porém, quando a conexão é retomada eles voltam a ser exibidos;
- Busca mesmo com o dispositivo *offline*, ainda traz resultados.

Em suma, é possível concluir que pelas características observadas pelos participantes, as vantagens de usabilidade oferecidas no aplicativo 2 estão diretamente relacionadas aos tratamentos oferecidos pela biblioteca *OfflineManager*. Em alguns detalhes, é possível notar a influência dos modos diferentes de tratamento: em algumas telas o sujeito achou

interessante poder confirmar ou não o tratamento *offline*, e em outras, o carregamento automático oferecido pelo modo transparente foi simplesmente observado como um ganho de usabilidade, por disponibilizar os dados automaticamente.

Décima oitava questão, penúltima do questionário, pergunta: "Comparando ambas as aplicações, qual você avalia com melhor experiência de usuário, para funcionamento *offline*?". Nesta, todos marcaram o aplicativo 2 como de melhor usabilidade.

Em seguida, foi pedido para que os sujeitos usassem uma escala de zero a dez para indicar o quão melhor em termos de usabilidade uma aplicação é em relação a outra. Essa questão foi proposta para tentar quantificar a melhoria de usabilidade nos cenários de funcionamento *offline*, de forma a entender se, pelo menos para o contexto da aplicação utilizada no experimento, a melhoria foi pequena ou significativa. Nesta escala, que é a última pergunta da avaliação do experimento de comparação, o sujeito 1 marcou 9 na escala, o sujeito 2 marcou 8 e o sujeito 3 também marcou 9. Indicando assim, que no contexto deste experimento, a melhoria foi significativa.

4.3.3 Discussão

Retomando os objetivos desta pesquisa, bem como os dois pontos críticos mencionados no início deste capítulo, esse experimento de comparação indica que pelo menos o primeiro ponto crítico foi atendido. O experimento teve como objetivo comparar a usabilidade em cenários de uso em modo *offline*, em dois aplicativos: o aplicativo 1, que era o aplicativo Inventum sem a utilização da biblioteca *OfflineManager*, e o aplicativo 2 que que era o aplicativo Inventum com a utilização da biblioteca *OfflineManager*.

As avaliações, por meio das respostas do questionário, mostraram indícios que a aplicação da biblioteca melhorou a usabilidade do aplicativo Inventum nos cenários avaliados. Questões fechadas tiveram respostas que indicam melhor usabilidade do aplicativo 2, enquanto questões abertas, onde os sujeitos do experimento puderam listar problemas nos aplicativos comparados, e também os pontos positivos de cada um, mostraram que alguns problemas enfrentados no aplicativo 1, não foram enfrentados no aplicativo 2.

Dentre os problemas listados, alguns foram mencionados por mais de um sujeito do experimento, ao se referirem ao aplicativo 1: dificuldade para carregar novamente os dados quando a conexão é retomada e falta de mensagens de *feedback*. Para o aplicativo 2, esses problemas não foram mencionados. Na verdade soluções para eles foram relatadas nos pontos positivos encontrados: aplicativo 2 oferece tratamento de forma a carregar os dados automaticamente, fornece mais mensagens de *feedback* e fornece possibilidade de confirmação para o tratamento da chamada em modo *offline*, por exemplo.

É importante mencionar que algumas características citadas pelos sujeitos, como por exemplo a funcionalidade de "*pull to refresh*", não foram atendidas pela biblioteca *Of-*

flineManager, bem como alguns *bugs* presentes no código original do aplicativo Inventum não foram resolvidos.

Contudo, com relação ao cenário de utilização em modo *offline*, os resultados da comparação mostram que a biblioteca *OfflineManager* atingiu seu objetivo, ao ser aplicada no aplicativo 2 do experimento. Como o propósito do experimento era avaliar quesitos de usabilidade, entende-se que o tratamento oferecido pela biblioteca, e avaliado no experimento, trata também da camada de interface, atendendo assim ao primeiro ponto crítico.

Com relação ao segundo ponto crítico, o experimento não tratou de questões de código. Os sujeitos já receberam os aplicativos a serem comparados previamente instalados em dispositivos Android. Portanto, não é possível concluir por meio desse experimento de comparação, se a aplicação da biblioteca *OfflineManager* reduz esforços de implementação do requisito de funcionamento *offline*, em estágio inicial ou avançado do desenvolvimento de aplicações.

4.4 Avaliação heurística

O último mecanismo de avaliação, que se volta mais para a camada de interface da aplicação final produzida por meio da aplicação da biblioteca *OfflineManager*, foi uma avaliação heurística de usabilidade, considerando os pontos de interação da biblioteca com o usuário final.

De acordo com Salgado, Rodrigues e Fortes (2016), em um trabalho de atualizar uma revisão sistemática sobre pesquisas de avaliação heurística voltada ao meio mobile, foi possível listar um aumento de mais de 200% na quantidade de pesquisadores que estão utilizando heurísticas para medir usabilidade em aplicações móveis, e um aumento de quase 400% no número de estudos selecionados que aplicam heurísticas para avaliação de usabilidade em dispositivos móveis. Isso durante os dois anos deste mapeamento (2014-2015), em comparação com o primeiro mapeamento sobre o período de 2004 a 2013. Isso mostra que o processo de avaliação heurística vem sendo bastante estudado e referenciado na literatura a fim de avaliar usabilidade em dispositivos móveis.

O objetivo desta seção é discutir as heurísticas adotadas para avaliação, e expor como a biblioteca se comporta diante dessas métricas de usabilidade. Para isso, precisa-se primeiramente escolher o conjunto de heurísticas a serem adotadas na avaliação. Apesar das heurísticas tradicionais de Nielsen (MACK; NIELSEN, 1994) serem as mais utilizadas para avaliações de usabilidade, elas não foram criadas e pensadas para o cenário de aplicações móveis (NETO; PIMENTEL, 2013). Por tanto, tem-se valor em encontrar um conjunto revisado de heurísticas adaptadas para este cenário.

A avaliação proposta nesta dissertação se baseia no trabalho proposto por Inostroza et al. (2013), que propõe um conjunto de 12 heurísticas voltadas ao cenário de aplicações móveis. É importante mencionar que o estudo apresentado em (INOSTROZA et al., 2013) foi feito de forma a atualizar as heurísticas propostas previamente por Inostroza et al. (2012). A seguir serão apresentados alguns embasamentos teóricos para contextualizar essa avaliação. Em seguida são apresentadas as heurísticas, e por fim sua aplicação à biblioteca *OfflineManager*.

4.4.1 Dispositivos móveis

Dispositivos móveis são pequenos aparatos eletrônicos, com certa capacidade de processamento e conexão permanente ou intermitente com a Internet. Assim, pode-se considerar diversos aparelhos como dispositivos móveis: aparelhos GPS, laptops, mp3 players e *smartphones* (INOSTROZA et al., 2012).

Dispositivos móveis baseados em toque são considerados todos os dispositivos móveis que possuem uma tela sensível ao toque, e por isso o trabalho de Inostroza et al. (2013) pode ser aplicado a quase todos esses dispositivos móveis, quase sem adaptação. Porém, o seu foco principal são os smartphones, devido a sua crescente popularidade e facilidade de acesso. No contexto dessa dissertação, a biblioteca foi construída para a plataforma Android, sistema operacional popular para smartphones, e a avaliação heurística proposta é executada com isso em mente.

4.4.2 Avaliação de usabilidade em dispositivos móveis

Para poder fazer uma avaliação de usabilidade em aplicações de dispositivos móveis, é preciso primeiro entender os aspectos do produto em questão. Alguns itens que influenciam o design de dispositivos móveis, são (HEO et al., 2009) (LEE et al., 2006):

- São utilizados com as mãos;
- São operados de maneira *wireless* (sem fios); e
- Suportam conexão com a Internet e adição de diversas aplicações.
- Possuem um pequeno espaço de tela;
- Geralmente os botões que apresentam possuem mais de uma função;
- Possui poder limitado de processamento.

O último aspecto, porém, pode ser desconsiderado no cenário atual dos *smartphones*, em vista da grande evolução de *hardware* que ocorreu nos últimos anos nesses dis-

positivos. Hoje em dia esses aparelhos possuem quase o mesmo poder computacional que computadores pessoais tradicionais.

Outra questão é a diferença no contexto de utilização de dispositivos móveis, que se difere bastante do contexto de aplicações tradicionais. Nestas, a interação é bem definida em termos de luzes, sons e formas de interação, via teclado e mouse. Já no contexto mobile, esses termos são incertos, e depende de onde a interação com a aplicação está acontecendo, e como ela é realizada. O usuário pode estar utilizando a aplicação com apenas uma mão, em uma fila de banco, como pode também estar utilizando a aplicação com as duas mãos, sentado no banco de um parque (INOSTROZA et al., 2013). Existem ainda outros modos de interação possíveis, que nem existem fora do contexto de aplicações móveis, como por exemplo a utilização de giroscópio ou do acelerômetro do dispositivo como forma de entrada de dados. Exemplos disso são: vídeos 360 onde o usuário pode virar o dispositivo em um eixo para ver diferentes ângulos do vídeo; ou a ação de balançar o dispositivo rapidamente para apagar um texto que foi escrito na aplicação. Como foi mostrado, avaliação de usabilidade no contexto de dispositivos móveis pode ser difícil, se não houver o emprego das ferramentas adequadas.

4.4.3 Heurísticas propostas por Intronza et al. para avaliação de usabilidade em dispositivos móveis

Na tabela 2 são apresentadas as heurísticas propostas no trabalho de Inostroza et al. (2013) e que foram utilizadas para a avaliação da biblioteca *OfflineManager*.

A seguir, é apresentado o resultado de uma análise sobre como a biblioteca *OfflineManager* pode auxiliar no atendimento a essas heurísticas.

4.4.4 Resultado da avaliação

Antes de avaliar as heurísticas, é importante mencionar que o projeto a ser avaliado é uma biblioteca Android, e não um aplicativo completo. Por isso, pode ser que algumas heurísticas não se apliquem diretamente sobre a biblioteca, ou sobre as interações fornecidas por ela, mas sim sobre a aplicação que a está utilizando. Ou seja, a biblioteca *OfflineManager* foi pensada para atender diferentes contextos de aplicações móveis, e algumas heurísticas avaliam características de responsabilidade da aplicação, podendo assim não serem atendidas pela biblioteca.

Para cada heurística apresentada na seção anterior, é feita uma breve discussão sobre se a biblioteca *OfflineManager* ajuda ou não a atender essa heurística.

H1 – Visibilidade do status do sistema: A biblioteca ajuda no atendimento a essa heurística no que se refere ao aspecto da conexão somente, por meio do modo ver-

Heurística	Descrição
H1 - Visibilidade do status do sistema	O dispositivo deve manter o usuário informado sobre todo progresso e mudanças de estado por meio de feedbacks em um intervalo razoável de tempo.
H2 – Assemelhamento entre o sistema e o mundo real	O dispositivo deve falar a linguagem do usuário, ao invés de termos técnicos e linguagens orientadas ao sistema. O sistema deve usar convenções do mundo real e mostrar informações de um modo lógico e natural.
H3 – Controle e liberdade do usuário	O dispositivo deve permitir ao usuário desfazer ou refazer suas ações, e também fornecer uma saída de emergência, para permitir saída de estados indesejados. Essas ações devem, preferencialmente, serem fornecidas por meio de botões físicos, ou semelhantes.
H4 – Consistência e padrões	O sistema deve seguir convenções estabelecidas, sobre a condição de que o usuário deve poder realizar ações de um modo familiar, padronizado e consistente.
H5 – Prevenção de erro	O Sistema deve esconder ou desativar funcionalidades que não estão disponíveis, avisar sobre ações críticas e prover acesso a informações adicionais.
H6 – Minimizar o fardo de memória do usuário	O sistema deve visivelmente oferecer objetos, ações e opções, a fim de prevenir a memorização de informação de uma parte para outra dos elementos visuais.
H7 – Customização e atalhos	O sistema deve prover opções de configuração básicas e avançadas, permitir definições e customização (ou criação) de atalhos para ações frequentes.
H8 – Eficiência de uso e desempenho	O sistema deve ser capaz de carregar as informações e mostrá-las em um período razoável de tempo e minimizar os passos necessários para a realização de uma tarefa. Animações e transições devem ocorrer suavemente.
H9 – Estética e design minimalista	O sistema deve evitar mostrar informações não desejadas no contexto definido de utilização.
H10 – Ajudar os usuários a reconhecer, diagnosticar e a se recuperar de erros no sistema	O sistema deve mostrar mensagens de erro em uma linguagem familiar ao usuário, indicando o erro de uma forma precisa, e sugerindo soluções construtivas.
H11 – Ajuda e documentação	O dispositivo deve fornecer ajuda e documentação fáceis de serem encontradas, centradas na ação atual sendo executada pelo usuário, e provendo passos concretos a serem seguidos.
H12 – Interações físicas e ergonômicas	O dispositivo deve fornecer botões físicos ou similares para funcionalidades principais, localizados em posições reconhecíveis pelo usuário, que devem se encaixar na postura natural das mãos do usuário.

Tabela 2 – Tabela com a definição das heurísticas de usabilidade para dispositivos móveis, propostas por Inostroza et al. (2013)

boso. Nesse modo, são mostrados *feedbacks* referentes a todas as ações tomadas e tarefas completadas relativas à conexão. Além disso, há o modo flexível, que oferece um ponto de interação com o usuário, por meio de um pop-up onde o próprio usuário decide se o tratamento da chamada vai ser realizada ou não. No entanto, além desses modos predefinidos, a biblioteca não oferece meios para ajustar a quantidade de *feedback* de um modo mais fino, o que seria o ideal.

H2 – Assemelhamento entre o sistema e o mundo real: A biblioteca ajuda para o atendimento a essa heurística, pois oferece todos os meios necessários para que o desenvolvedor possa usar uma linguagem adequada. Porém isso só se aplica no aspecto da conexão, onde a biblioteca atua. Todas as mensagens oferecidas pela biblioteca, tanto de *feedback* quanto de interação por meio de pop-up, podem ser sobrescritas pelo desenvolvedor permitindo adequação da linguagem utilizada.

H3 – Controle e liberdade do usuário: Para essa heurística, o controle a ser oferecido, no sentido de desfazer ou refazer e oferecer saídas de emergência, fica sob responsabilidade da aplicação como um todo, tratando a funcionalidade em questão nesse sentido. A biblioteca atua somente no aspecto de conexão e tratamento de chamadas para uso *offline*. E mesmo nesse aspecto, o máximo que ela oferece é o modo flexível, onde o usuário decida se o tratamento *offline* será realizado ou não. Uma vez que o tratamento da chamada é realizado pela biblioteca, não existe mecanismo para que ela seja cancelada. Portanto, a biblioteca não auxilia no atendimento a essa heurística.

H4 – Consistência e padrões: A biblioteca tenta garantir essa heurística, no sentido de que todas as interfaces que oferece são elementos padrões da plataforma Android.

H5 – Prevenção de erro: A biblioteca oferece ajuda para o atendimento a essa heurística. Um exemplo disso são os modos de tratamento *Enforce* e *NoAction* que previnem que erros aconteçam no envio de chamadas, por falta de conexão com a Internet ou com o servidor, respectivamente. Idealmente, além de evitar os possíveis erros, os elementos de interface que disparam essas chamadas deveriam também ser ocultos, de acordo com essa heurística. Isso pode ser considerado um trabalho futuro deste mestrado – associar os tratamentos da biblioteca com os elementos de interface respectivos, para que a interface seja tratada de forma ainda mais refinada. No mais, a biblioteca também ajuda detectando problemas de conexão ou ausência de servidor, fornecendo fluxos alternativos para que erros sejam evitados. Isso se da por meio dos outros tratamentos oferecidos, como o *Transparent* ou o *Flexible*.

H6 – Minimizar o fardo de memória do usuário: A biblioteca ajuda no sentido de oferecer modos padrão de interação, conforme a heurística H4. Também ajuda por

meio do modo verboso. Nesse modo a biblioteca irá causar o aparecimento de mensagens informativas a todo momento em que o status da conexão mudar e ações realizadas pela biblioteca forem completadas, ajudando o usuário a retomar fluxos de utilização, ou simplesmente se lembrar de como está a conexão.

H7 – Customização e atalhos: A biblioteca não oferece nenhum tipo de ajuda com relação a essa heurística.

H8 – Eficiência de uso e desempenho: Em tese, a biblioteca não acrescenta nenhuma piora no desempenho do aplicativo, quando adicionada para tratar os cenários de uso em modo *offline*. Porém, ela utiliza alguns recursos a mais para seu funcionamento, por exemplo: estrutura de *hash-map* para armazenar as chamadas, escalonamento e execução dos serviços em segundo plano, etc. E esses são pontos que poderiam gerar piora no desempenho. Portanto, para responder adequadamente sobre essa heurística, seria necessário realizar testes de desempenho com a biblioteca.

H9 – Estética e design minimalista: A biblioteca oferece ajuda para o atendimento dessa heurística. Um exemplo disso é o modo de tratamento transparente, para caso de dispositivo *offline* e de servidor *offline*. Com eles, em certos casos, é possível reduzir a quantidade de informações fora do contexto definido de utilização ou da chamada, permitindo um design minimalista. O modo verboso marcado como falso também serve para essa função, reduzindo a interação com o usuário no tratamento *offline* apenas para pontos em que a biblioteca requisita confirmação (caso a combinação seja modo flexível para dispositivo e servidor, e modo verboso falso). Portanto, cabe ao desenvolvedor escolher a melhor forma de usar os tratamentos disponíveis. O único ponto possível de problema é o componente *Toast* para *feedback*, pois ele é apresentado mesmo com a aplicação em segundo plano ou com outra aplicação sendo executada em primeiro plano. Apesar de funcionar como um tipo de notificação ao usuário, ele pode interferir com a utilização de outras aplicações.

H10 – Ajudar os usuários com relação a erros no sistema: A biblioteca oferece meios para ajudar com essa heurística em dois pontos. O primeiro, se da por permitir que as mensagens de pop-up e *feedbacks* sejam configuradas para serem familiares ao usuário. As mensagens padrão fornecidas pela biblioteca são em inglês e foram definidas para serem o mais genéricas possíveis, portanto cabe ao usuário decidir sobre sua viabilidade, ou implementar mensagens que sejam mais claras no contexto de sua aplicação. O segundo ponto é a ajuda na recuperação de erros causados por falta de conexão, que se da por diferentes meios: tratando a chamada de forma silenciosa, impedindo a continuação de ação, perguntando ao usuário sobre a continuidade do tratamento, etc.

H11 – Ajuda e documentação: A biblioteca acrescenta alguns modos novos de interação com a aplicação. A ajuda e documentação providas pelo desenvolvedor devem incluir esses modos. Nesse sentido, deveria haver algum tipo de texto de ajuda padrão para que o desenvolvedor possa incorporar mais facilmente ao seu projeto. Mas isso não foi feito e pode ser considerado uma melhoria futura sobre a biblioteca que foi disponibilizada.

H12 – Interações físicas e ergonômicas: Não se aplica ao contexto da biblioteca. Questões físicas e ergonômicas devem ser tratadas pela aplicação, considerando o seu escopo e contexto, e fogem do âmbito de atuação da biblioteca *OfflineManager*.

4.4.5 Discussão

Retomando os pontos críticos mencionados no início deste capítulo, pode-se dizer que o primeiro deles foi atendido. A biblioteca *OfflineManager* oferece ajuda para atender a maioria das heurísticas de usabilidade utilizadas neste processo de avaliação. No entanto, algumas ressalvas foram feitas, em algumas das heurísticas, como por exemplo: a necessidade de um teste de desempenho para comprovar que a biblioteca não interfere no desempenho da aplicação.

Foi detectado também uma possível melhoria da biblioteca que ajudaria a atender uma heurística de forma ainda mais refinada: associar os tratamentos oferecidos aos componentes que realizam as chamadas. Isso ocorreria para a tratar também os elementos visuais em caso de falta de conexão, não apenas com *feedbacks* ou pop-up, mas escondendo ou desabilitando certos elementos de interface, por exemplo. Essa melhoria refinaria a ajuda oferecida para a heurística H5 – Prevenção de erro.

Para algumas heurísticas também, a biblioteca não oferece nenhum tipo de ajuda. Por exemplo a heurística H7 – Customização e atalhos, ou a H12 – Interações físicas e ergonômicas. Esses aspectos saem do escopo de tratamento oferecido pela biblioteca *OfflineManager*, e devem ser tratados diretamente no aplicativo onde ela é aplicada.

Dessa forma, dadas essas ressalvas, pode-se observar que no aspecto de conexão e funcionamento em modo *offline*, a biblioteca oferece ajudas satisfatórias para o atendimento das heurísticas apresentadas. Com isso, pode-se dizer que ela oferece apoio a solução do problema de funcionamento em modo *offline*, e o faz de forma a atender também a camada de interface e questões de usabilidade.

Com relação ao segundo ponto crítico, essa avaliação não tratou características específicas de código. Por isso, não fornece dados para conclusões sobre a redução de esforço de implementação ao tratar o problema de uso em modo *offline*.

4.5 Considerações finais

Considerando o objetivo deste trabalho de mestrado, que foi a construção de uma biblioteca para facilitar ao desenvolvedor a aplicação de comportamento em cenários de utilização *offline* em aplicativos Android, pode-se considerar que de maneira geral o trabalho foi bem sucedido e os objetivos foram alcançados.

Em primeiro lugar, as avaliações demonstraram que é possível utilizar-se da biblioteca em um projeto de aplicação real para atender a requisitos de funcionamento *offline*, e tratar problemas de intermitência de disponibilidade do servidor de *back-end* da aplicação.

Com relação aos objetivos, destacam-se observações feitas durante as avaliações sobre como a biblioteca pode ajudar a resolver os dois pontos críticos levantados no início desta dissertação, conforme discutido nas Seções 4.1.6, 4.2.6, 4.3.3 e 4.4.5.

Além disso, as avaliações possibilitaram identificar aspectos positivos e negativos da biblioteca, além de propiciar algumas lições aprendidas, conforme destacado nas seções deste capítulo. A seguir apresenta-se um resumo dessas observações e lições aprendidas. Para cada item que se segue, identifica-se a avaliação que propiciou a observação, da seguinte maneira: Opinião de Especialista (OE), Aplicação em Projeto Real (APR), Avaliação com Usuários (AU) e Avaliação Heurística (AH).

- O problema da falta de conexão é comum e precisa ser tratado (OE, AU);
- A biblioteca mostrou-se fácil de instalar, aprender e utilizar (OE, APR);
- A biblioteca se mostrou aplicável. sua utilização não fere princípios de desenvolvimento para dispositivos móveis, e utiliza componentes padrões da plataforma Android (OE, APR, AU, AH);
- Os modos oferecidos de tratamento para falta de conexão estão adequados (OE, APR, AU, AH);
- Outros modos de tratamento poderiam ser implementados, por exemplo: modo que armazena por mais tempo as chamadas ou as trata mesmo se a aplicação for encerrada (OE); modo que permite o cancelamento do tratamento depois dele ter sido feito (OE, AH); ou modo que trata chamadas encadeadas de forma configurável, realizando somente as desejadas e evitando chamadas desnecessárias (APR);
- Utilizar a biblioteca em um sistema já existente mostrou-se uma tarefa trivial, apesar de serem necessárias algumas pequenas adaptações (APR);
- A biblioteca poderia possibilitar a desativação de botões ou outros itens de tela de acordo com a presença/ausência de conexão (AH);

- Poderia ser oferecido algum tipo de texto de exemplo de ajuda da biblioteca para que o desenvolvedor insira na documentação de sua aplicação (AH);

- Um teste de desempenho da biblioteca é necessário para avaliar se de fato ela não influencia no desempenho da aplicação onde é utilizada (AH);

- As mensagens de *feedback* oferecidas pela biblioteca são adequadas (OE, AU, AH);

- O componente *Toast* como elemento para *feedbacks* pode não ser o mais adequado em alguns cenários, pois pode interferir em outras aplicações (OE, AH);

- Algumas refatorações ou pequenas funcionalidades, mesmo não tratando o problema de uso em modo *offline* em si, ajudam na experiência de uso em cenário de falta de conexão (APR, AU);

- Protodiretrizes para escolha dos modos de tratamento foram criadas e se mostraram adequadas em uma primeira utilização. Com realização de mais testes aplicando em diferentes projetos, elas podem ser consolidadas como diretrizes reais para utilização da biblioteca (APR);

- Customizar as mensagens de *feedback* oferecidas se mostrou ser um procedimento simples (APR);

- A biblioteca se mostrou adequada para tratar o problema de uso em modo *offline* (OE, APR, AU);

- A biblioteca, em cenários de uso em modo *offline*, mostrou indícios de melhoria na experiência de usuário da aplicação onde é utilizada (OE, APR, AU);

- A biblioteca poderia oferecer controle refinado sobre quais mensagens de *feedback* seriam exibidas, não somente se vão aparecer ou não, como é feito pelo uso do modo verboso (OE, AH);

- A biblioteca oferece ajuda para atender heurísticas de usabilidade, porém cabe ao desenvolvedor fazer a escolha correta dos modos de tratamento para que elas sejam atendidas de fato (AH);

4.5.1 Ameaças à validade

Apesar de permitirem observações interessantes e indícios de que os objetivos desta pesquisa foram alcançados, as avaliações realizadas possuem uma série de limitações e ameaças à validade. A seguir apresenta-se as principais que foram identificadas.

4.5.1.1 Opinião de Especialista

Para a avaliação por meio de opinião de especialistas, as primeiras ameaças que podem ser citadas são relacionadas aos especialistas selecionados. No capítulo 4 foram introduzidos alguns conceitos sobre essa metodologia de avaliação, e um deles é justamente sobre a escolha dos especialistas. Garcia, Almeida e Meira (2011) explicam que o número de especialistas selecionados não aumenta a credibilidade do estudo, pois suas opiniões não agregam valor para a avaliação por meio de distribuição estatística. Assim, existe um número a partir de onde pouco valor é agregado. Para o estudo apresentado nesta dissertação foram selecionados 7 especialistas, com um certo grau de confiança sobre esse ser um número adequado, porém a número de especialistas não pode ser descartado como uma possível ameaça.

Outra ameaça, diretamente relacionada com a primeira, refere-se a qualidade dos especialistas para o estudo avaliado. Novamente, o capítulo 4 também apresenta conceitos referentes aos critérios de seleção. Critérios que foram analisados, e estabelecidos previamente à seleção. Porém existe a possibilidade de os critérios estabelecidos não serem suficientes para garantir o nível desejado de experiência para os especialistas. Idealmente, mais colaboradores do Retrofit poderiam ter participado, pois saberiam avaliar a biblioteca *OfflineManager* também de forma comparativa com outros mecanismos para tratar o problema. Porém questões de tempo e de dificuldade no contato com esses especialistas dificultaram essa participação. Pode-se dizer que com o tempo e possível popularização da biblioteca na plataforma GitHub, outros especialistas forneçam feedback ou até sugestões sobre melhorias, o que ajudará em posteriores avaliações da biblioteca *OfflineManager*.

Ainda sobre os especialistas, é importante mencionar que 4 deles eram conhecidos do autor desta dissertação. Como sabiam que a proposta da biblioteca era auxiliar e facilitar o tratamento para uso em modo *offline* de aplicações, não se pode descartar um possível viés para favorecer a biblioteca *OfflineManager*, mesmo que não intencional, e consequentemente favorecer o autor desta dissertação. Para tentar resolver essa ameaça, foi pedido para que os especialistas utilizassem a biblioteca em seus projetos, e a avaliassem da maneira mais objetiva possível, independente dos resultados. Foi lhes dito também que a avaliação não tinha propósito comparativo, mas sim apenas o de encontrar problemas e identificar sua viabilidade, o que pode ter motivado uma análise mais crítica.

Por fim, ainda sobre os especialistas, outra ameaça possível é o fato de que eles não tiveram um longo período de tempo para utilizar a biblioteca. Assim, é possível que não tenham tido experiência suficiente para poder avaliar todas as suas nuances. Foi dado aos especialistas mês para a avaliação, sendo que a biblioteca completa foi lhes enviada junto com o questionário de avaliação, e dois tipos de documentação: textual, e um vídeo explicativo englobando processos de instalação e utilização.

Com relação à metodologia de coleta de dados, podemos citar duas ameaças: qualidade do questionário avaliativo, e tempo fornecido para respostas. Com relação à qualidade, o questionário é composto por um conjunto de questões fechadas e abertas. Isso permite que alguns aspectos fossem avaliados de forma mais objetiva, enquanto ainda podiam listar de forma livre questões como problemas encontrados, sugestões de melhoria, etc. O questionário foi elaborado com ajuda do grupo de pesquisa do autor desta dissertação, e foi revisado duas vezes com a ajuda de dois especialistas que trabalham na área de desenvolvimento móvel, e um pesquisador da área de engenharia de software. Esses três indivíduos que ajudaram nas revisões do questionário não participaram como especialistas na avaliação da biblioteca. Mesmo assim, é possível que tenha faltado algum questionamento, ou que algum questionamento tenha sido feito de forma errada.

Com relação ao tempo de resposta, existe a possibilidade de o tempo não ter sido suficiente para o fornecimento de respostas. Isso pode ter afetado na qualidade das respostas fornecidas. No entanto, dentro do prazo estabelecido de 1 mês para as respostas, nenhum especialista entrou em contato pedindo mais prazo para a avaliação. Isso pode ser um indício de que tempo não tenha sido um problema na avaliação.

4.5.1.2 Aplicação em Projeto Real

No processo de avaliação por aplicação em um projeto real, a principal ameaça a validade identificada é o viés de pesquisador. O processo de aplicar a biblioteca *OfflineManager* ao projeto do aplicativo Inventum foi feito pelo próprio autor desta dissertação, que mesmo de forma não-intencional, pode ter influenciado os resultados, no sentido de favorecer a biblioteca proposta neste trabalho de mestrado. Para evitar isso, porém, o processo foi feito da forma mais objetiva possível, e as considerações foram listadas a medida que os problemas foram encontrados.

Outra ameaça importante de ser mencionada é que o aplicativo Inventum é um projeto de médio porte. Dessa forma, é possível que a aplicação da biblioteca *OfflineManager* tenha sido facilitada, onde idealmente a avaliação poderia ter sido feita sobre um projeto de grande porte, se assemelhando ainda mais de um cenário real de desenvolvimento.

Por fim, ainda sobre o projeto escolhido, pode-se dizer que o aplicativo Inventum não tinha grande necessidade de funcionamento *offline*. Durante o processo de análise do aplicativo, explicado no capítulo 4, foi estudado como o aplicativo trata a questão de falta de conexão, e assim concluiu-se que adição de funcionalidades em modo *offline* adicionaria valor ao aplicativo, além de resolver alguns problemas existentes. Porém, é um aplicativo cujo contexto e utilização trata apenas de consultas ou visualização de informações, assim não oferece nenhum tipo de interação mais sofisticada, não havendo portanto necessidade de fato de tratamento para uso em modo *offline*.

Idealmente, para avaliar melhor os tratamentos oferecidos pela biblioteca *Offline-*

Manager, seria interessante o aplicativo refatorado oferecer funcionalidades de submissão de formulários, troca de mensagens, realização de transações, etc. Isso para que a avaliação seja feita também para esses diferentes cenários de utilização, que podem aumentar ou diminuir a necessidade de tratamento para uso em modo *offline*.

4.5.1.3 Avaliação com Usuários

Para o experimento com usuários, a primeira ameaça a validade identificada é o número de usuários participantes do experimento, que foi realizado com apenas 3 sujeitos. Apesar de algumas observações interessantes terem sido feitas nesse processo de avaliação, possibilitando algumas discussões, o baixo número de sujeitos como usuários no experimento pode ter empobrecido os resultados obtidos.

Além disso, o perfil dos sujeitos participantes não é suficientemente diversificado. Os participantes são todos envolvidos com tecnologia ou trabalham na área da computação, e a variação máxima de idade entre os sujeitos é de 10 anos do sujeito mais novo para o sujeito mais velho. Dessa forma, para que o experimento avaliasse diferentes tipos de usuário, seria necessário além de um número maior de participantes, que houvesse mais diversidade na idade dos sujeitos ou área de atuação.

Ainda, a coleta de dados por meio de questionário não é um método suficiente para a avaliação do experimento. Idealmente, em um experimento que envolve usabilidade e experiência de usuário, seria necessário a utilização de técnicas mais apuradas, como por exemplo registrar a tela do dispositivo e a face do usuário ou usar o *think aloud* (técnica onde os sujeitos são instruídos a falar em voz alta os raciocínios empregados na utilização do aplicativo, no experimento).

Outra ameaça a validade nesse avaliação com usuários, é a possibilidade de certo viés de aprendizado. Para a realização do experimento, primeiro os sujeitos utilizaram o aplicativo sem tratamento com a biblioteca *OfflineManager*, e depois utilizaram o aplicativo com a biblioteca aplicada. Dessa forma, é possível que na segunda utilização os usuários já estavam mais familiarizados com o aplicativo e por isso julgaram de forma a favorecer o que tinha a biblioteca *OfflineManager* aplicada. Para diminuir essa influência, no entanto, foi oferecido uma etapa anterior de treinamento, a fim de familiarizar os usuários previamente com a aplicação, antes da execução dos roteiros do experimento. Mesmo assim, é algo que precisa ser ajustado e corrigido em futuras avaliações de forma a eliminar esse fator de influencia.

Outra ameaça possível, é que os sujeitos participantes do experimento conheciam o autor dessa dissertação, que conduziu o experimento. Assim, mesmo que de forma não intencional, é possível que tenham avaliado os aplicativos de forma a favorecer o que utilizava os tratamentos da biblioteca *OfflineManager*. Para tentar diminuir essa influência, os sujeitos não foram informados sobre as características da biblioteca proposta nesse traba-

lho de mestrado. Também não foram informados sobre qual aplicativo a utilizava ou sobre como ela tratava as chamadas ou a utilização em modo *offline*. Foram instruídos para que simplesmente executassem o roteiro proposto nas duas versões do aplicativo Inventum, avaliassem da forma mais objetiva possível essas duas versões, e fizessem considerações sobre elas.

Por fim, o método como a falta de conexão com a Internet foi simulado pode não ter sido o ideal. No experimento, essa simulação foi feita de forma a instruir os sujeitos a desligarem a conexão, desligando o *Wi-Fi* do dispositivo, e a ligarem novamente em um segundo momento. Idealmente essa simulação deveria ser feita sem o usuário perceber.

4.5.1.4 Avaliação Heurística

Com relação a avaliação heurística apresentada nesta dissertação, algumas ameaças a sua validade podem ser listadas. A primeira delas é o fato de ser uma avaliação subjetiva. Foi realizada de forma a analisar a biblioteca *OfflineManager*, e avaliar sob o ponto de vista de usabilidade, se ela atende ou ajuda a resolver os pontos levantados pelas heurísticas propostas. E foi realizada pelo autor desta dissertação, portanto baseia-se em suas próprias experiências com a biblioteca *OfflineManager* ou seu próprio entendimento das heurísticas propostas.

Porém, esse é geralmente o modo como o processo de avaliação heurística é feito para usabilidade de *software*, como é o exemplo da avaliação feita no trabalho de Inostroza et al. (2013). O que deveria ser diferente, para melhorar a avaliação heurística da biblioteca *OfflineManager*, é que ela deveria ser feita por terceiros especialistas, e não pelo autor desta dissertação. E deveria ser feita não somente por um especialista, mas por vários, para que a avaliação seja feita sobre a agregação de suas opiniões individuais.

Por fim, como explicado no capítulo 4, essa avaliação foi feita sobre a biblioteca e sobre como ela poderia ajudar no atendimento às heurísticas. Não foi feita sobre um sistema completo construído com ela. Isso muda um pouco o enfoque do processo de avaliação heurística de usabilidade, o que pode se mostrar como uma ameaça a validade da avaliação aqui apresentada.

5 Considerações finais

Com a popularização de dispositivos móveis na forma de *smartphones* e a expansão das lojas de aplicativos, sistemas que rodam sobre esses dispositivos são cada vez mais comuns, tornando mais frequentes estudos sobre esse contexto. O aumento da conectividade dessas aplicações com serviços de *back-end*, ou com outros dispositivos, através da rede, faz com que esses sistemas sejam cada vez mais dependentes de conexão com a Internet. Nesse sentido, existe o problema de que em muitos lugares, conexão com a Internet não é algo garantido. Por esse motivo, no cenário de desenvolvimento de aplicativos, requisitos de funcionamento em modo *offline* são frequentemente discutidos.

Assim, muitas aplicações hoje no mercado já tentam resolver esse problema, e fornecem certos tipos de uso em modo *offline*, como o *Whats-App* ou o aplicativo de mensagens do *Facebook*. Além de aplicações de mercado, existem também algumas soluções para ajudar aos desenvolvedores de aplicações abordarem esse aspecto em seus aplicativos. Na plataforma Android, alguns exemplos podem ser citados como: usar interceptadores HTTP e implementar mecanismos de *cache*, utilizar servidores *sandbox* que já disponibilizam sincronização automática de dados, etc.

Existem também *frameworks* que consideram essa questão para desenvolvimento de aplicativos móveis. Um exemplo é o *Xamarin*, plataforma da Microsoft para desenvolvimento multiplataforma de aplicativos, que fornece implementação e um banco de dados próprio na nuvem, para tratar uso em modo *offline* por meio de sincronização de dados.

Além disso, alguns trabalhos acadêmicos também tratam essa questão, e tentam propor soluções para ela de formas diferentes. Um exemplo disso, é o trabalho de Perkins et al. (2015) que apresenta um serviço de sincronização de dados, na nuvem, utilizando um novo tipo de abstração de dados.

Existe também o trabalho de Peters et al. (2011) que propõe um novo protocolo REST para sincronização de dados, mantendo cópias em bancos locais na aplicação, tornando o tratamento *offline* transparente ao usuário.

Outras soluções de mercado podem ser utilizadas a fim de resolver essa questão, bem como outros trabalhos acadêmicos podem ser citados, na mesma linha desses. Porém, todas as propostas estudadas conseguem resolver o problema apenas parcialmente.

Isso porque, considerando o atual estado da prática e o atual estado da arte, bem como as características de desenvolvimento de aplicações para dispositivos móveis, é possível verificar que existem dois pontos que são críticos neste cenário:

1. Oferecer apoio à solução do problema de funcionamento em modo *offline*, de forma

a atender todas as camadas do *software*: desde o *back-end* até o *front-end*. Ou seja, oferecer tratamentos que não resolvam apenas as camadas de comunicação ou serviços, mas que se estendam até a camada de interface, tratando e resolvendo também questões de usabilidade.

2. Reduzir o esforço de implementação do requisito de funcionamento *offline* tanto em desenvolvimentos iniciais como em sistemas já desenvolvidos.

A ideia desta pesquisa foi elaborar e desenvolver uma abordagem, na forma de uma biblioteca Android, para facilitar aos desenvolvedores de aplicações a implantação desses requisitos de funcionamento *offline*. E para isso, não tratar somente o problema do ponto de vista de funcionalidade, mas também de forma a ajudar a resolver os dois pontos críticos citados anteriormente.

5.1 Contribuições alcançadas

Este trabalho de mestrado apresentou a construção de uma biblioteca sobre a plataforma Android, chamada *OfflineManager*, que fornece um método para encapsular chamadas da biblioteca de comunicação Retrofit.

A *OfflineManager* oferece um processo rápido de instalação e permite ao desenvolvedor de aplicativos inserir rapidamente tratamentos para uso em modo *offline*. Para isso, basta encapsular a chamada tradicional do Retrofit pela chamada tratada da biblioteca, passando um conjunto de parâmetros que especificam qual será o comportamento daquela chamada nos cenários de dispositivo sem Internet e de servidor indisponível.

Esses parâmetros também especificam qual será o comportamento daquela chamada com relação a *feedbacks* para o usuário, informando-o sobre as ações que estão sendo realizadas, e especificam também o número de tentativas a serem feitas quando o servidor não responder.

Enquanto a maioria dos trabalhos acadêmicos abordam o problema do ponto de vista de dados, por meio de sincronização de dados ou protocolos, o que mais se aproxima da biblioteca aqui proposta foi o trabalho de Perkins et al. (2015).

Esse trabalho, apresentado no capítulo 2 (PERKINS et al., 2015), propõe uma nova abstração de dados para facilitar sincronismo e manter a consistência de dados entre servidores de aplicações, e também entre as próprias aplicações. Para demonstrar e utilizar essa nova abstração de dados proposta, chamada `sTable` ou *Simba Table*, os autores criaram um serviço completo de sincronização de dados, para rodar sobre a plataforma Android, com objetivo de tentar resolver os problemas de utilização em modo *offline*. Esse serviço foi chamado de `Simba`.

Essa proposta se aproxima, até certo ponto, da biblioteca apresentada neste trabalho de mestrado. São semelhantes pois possuem o mesmo objetivo, de tentar tratar utilização em modo *offline*, e ambas oferecem soluções completas, ou seja, prontas para serem aplicadas em um sistema.

Porém, é importante mencionar que os autores do `Simba` (PERKINS et al., 2015) abordam o problema do ponto de vista de dados. Por isso, o enfoque de seu trabalho é a nova abstração de dados que propõe, em conjunto com os mecanismos de sincronização e consistência de dados apresentados. Enquanto para a construção da biblioteca aqui apresentada, o foco foi nas chamadas HTTP realizadas ao *back-end*, que representam a grosso modo as ações dependentes de conexão executadas pelos usuários.

Outra comparação interessante de ser feita: o `Simba` funciona sobre algumas estruturas como o *Simba Client* e o *Simba Cloud*. O *Simba Client* é um serviço que roda no dispositivo Android, de forma transparente, e serve como armazenamento local e seguro dos dados. Funciona também como *gateway* para comunicação com o *Simba Cloud*, que representa todo o *back-end* do serviço proposto. O *Simba Cloud* se responsabiliza por manter os dados de acordo com os protocolos escolhidos de sincronização, bem como manter a relação entre os dispositivos entre si.

Nota-se aqui que, apesar de avaliado como simples de implementar e adicionar a um projeto pelos autores, o `Simba` requer estruturas adicionais, e toda uma arquitetura para a implementação de sua utilização, bem como a utilização do *Simba Cloud* como *back-end*.

Em contraste, a biblioteca aqui proposta, necessita apenas ser instalada em um projeto por meio do Graddle, que já está pronta para utilização. Além disso, não necessita de nenhum servidor específico, desde que a comunicação com o mesmo seja feita por chamadas HTTP e utilize a biblioteca de comunicação do Retrofit para isso.

Por fim, um ponto importante também é o fato de que o `Simba` não trata questões de interface. Foi construído justamente para funcionar de forma transparente aos usuários. Enquanto a biblioteca *OfflineManager*, como foi bastante discutido ao longo dessa dissertação, tenta oferecer uma solução que aborda também a questão de interface, tratando questões de usabilidade.

O trabalho de Do et al. (2014), também apresentado no capítulo 2, apresenta uma arquitetura sobre *proxy/broker*, pra armazenar dados com maior relevância para o usuário, disponibilizando-os mesmo em um cenário de uso *offline*. A biblioteca *OfflineManager* também pode ser aplicada no contexto de aplicações sociais, e oferece tratamentos adequados para esse contexto. Porém, funciona de forma diferente, pois não considera relevância do conteúdo a ser disponibilizado. Trata apenas as chamadas para que ações possam ser executadas, e dados possam ser trazidos automaticamente em um segundo

momento, com a retomada de conexão.

É importante mencionar também que a biblioteca construída foi publicada no GitHub, e está disponível para a comunidade de desenvolvedores Android, contribuindo com uma nova ferramente para a plataforma, e fomentando discussões relevantes nesse ecossistema.

5.2 Trabalhos futuros

Este trabalho por possuir cunho bastante prático, com a construção de uma biblioteca para a solução do problema apresentado, tem como trabalhos futuros, primeiramente, a evolução da solução na forma da correção de possíveis *bugs* ou melhoria de alguns mecanismos utilizados para o seu funcionamento. O principal deles é o método como a biblioteca armazena as chamadas e o seu contexto, para que sejam executadas posteriormente em serviços escalonados pelo sistema operacional do Android, em *background*. O modo utilizado atualmente, como explicado no capítulo 3, é um *hash-map* armazenado em uma variável *static* que existe durante todo o ciclo de vida da aplicação.

Essa não é a melhor abordagem, por questões de segurança e garantia de dados. Esse mecanismo não permite que as chamadas permaneçam armazenadas se a aplicação for encerrada, ou o dispositivo reiniciado, por exemplo. Ainda, é possível que essa estrutura aumente o consumo de memória da aplicação, caso o número de chamadas for grande. Além da memória, outros recursos podem ser mais utilizados, como bateria e processamento. Assim, para essa consideração, seria necessário um teste de desempenho para entender de fato se o aumento no consumo de memória, bateria ou de processamento é problemático.

Portanto, uma nova abordagem pode ser estudada para resolver essa questão, porém é uma melhoria puramente técnica para a biblioteca.

Como foi citado anteriormente, e também no capítulo 4, seria importante a realização de um teste de desempenho da biblioteca. Como trabalho futuro pode-se citar uma avaliação em um cenário de uso extremo da biblioteca, por exemplo: armazenando um grande número de chamadas, ou armazenando chamadas com grande quantidade de dados. Com esse trabalho, seria possível uma confirmação sobre a tese de que a biblioteca não afeta questões de performance da aplicação onde é utilizada.

Um outro trabalho futuro proposto é a aplicação da biblioteca *OfflineManager* em um conjunto maior e diferente de aplicativos reais, a fim comprovar as protodiretrizes propostas nesta dissertação, para transformá-las em diretrizes consolidadas. Isso seria de extremo valor para o trabalho proposto neste mestrado – além de fornecer a solução por meio da biblioteca, oferecer também um conjunto de diretrizes para guiar a escolha dos

modos de tratamentos, de forma a melhor atender os diferentes contextos de aplicações e utilizações em modo *offline*.

Outro possível trabalho futuro é a adaptação dos mesmos conceitos utilizados nesta pesquisa, para a construção da mesma biblioteca para a plataforma iOS. Do ponto de vista da engenharia de software, as contribuições científicas não seriam muito ampliadas, porém diversos pontos de interesse seriam levantados do ponto de vista técnico, além do que seria uma contribuição também para a comunidade de desenvolvimento iOS.

Considerando a construção da biblioteca também para a plataforma iOS, um ponto que poderia agregar bastante valor no âmbito científico seria direcionar pesquisas na área de desenvolvimento dirigido por modelos, para permitir ao desenvolvedor de software subir um degrau a mais na abstração de soluções para o cenário de uso *offline*, além de permitir também um certo grau de customização sobre os tratamentos gerados, e produzir uma solução para ambas as plataformas, por meio de geradores automáticos de código.

Por fim, é necessário realizar mais estudos de caso implantando a biblioteca em aplicações de diferentes contextos, para possivelmente refinar os tratamentos oferecidos. É necessário também fazer mais estudos ou experimentos para avaliar a usabilidade em cenários onde o servidor encontra-se indisponível. Esses cenários se assemelham bastante, do ponto de vista do usuário, aos de falta de conexão com a Internet, porém por dificuldades maiores em simular esse cenário de maneira realista, eles acabaram por serem menos abordados nos experimentos realizados.

Referências

CHHIBBER, S.; APOSTOLAKIS, G.; OKRENT, D. A taxonomy of issues related to the use of expert judgments in probabilistic safety studies. *Reliability Engineering & System Safety*, v. 38, n. 1, p. 27 – 45, 1992. ISSN 0951-8320. Disponível em: <http://www.sciencedirect.com/science/article/pii/095183209290103R>. Citado na página 75.

COOKE, R. M. *Experts in Uncertainty: Opinion and Subjective Probability in Science*. [S.l.]: Oxford University Press, 1991. Citado na página 72.

DO, N. et al. Optimizing offline access to social network content on mobile devices. In: *INFOCOM, 2014 Proceedings IEEE*. [S.l.: s.n.], 2014. p. 1950–1958. Citado 5 vezes nas páginas 13, 38, 39, 45 e 129.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. In: *Proceedings of the 22Nd International Conference on Software Engineering*. New York, NY, USA: ACM, 2000. (ICSE '00), p. 407–416. ISBN 1-58113-206-9. Disponível em: <http://doi.acm.org/10.1145/337180.337228>. Citado na página 42.

GARCIA, V.; ALMEIDA, E. S. D.; MEIRA, S. *RiSE Reference Model for Software Reuse Adoption in Brazilian Companies*. Tese (Doutorado) — Universidade Federal de Pernambuco, 2011. Citado 4 vezes nas páginas 72, 73, 75 e 122.

HEO, J. et al. A framework for evaluating the usability of mobile phones based on multi-level, hierarchical model of usability factors. *Interacting with Computers*, v. 21, n. 4, p. 263–275, 2009. Disponível em: <+ http://dx.doi.org/10.1016/j.intcom.2009.05.006>. Citado na página 114.

INOSTROZA, R. et al. Usability heuristics validation through empirical evidences: A touchscreen-based mobile devices proposal. In: *2012 31st International Conference of the Chilean Computer Science Society*. [S.l.: s.n.], 2012. p. 60–68. ISSN 1522-4902. Citado na página 114.

INOSTROZA, R. et al. Usability heuristics for touchscreen-based mobile devices: Update. In: *Proceedings of the 2013 Chilean Conference on Human - Computer Interaction*. New York, NY, USA: ACM, 2013. (ChileCHI '13), p. 24–29. ISBN 978-1-4503-2200-3. Disponível em: <http://doi.acm.org/10.1145/2535597.2535602>. Citado 6 vezes nas páginas 9, 59, 114, 115, 116 e 125.

KITCHENHAM, B. et al. Large-scale software engineering questions – expert opinion or empirical evidence? *IET Software*, Institution of Engineering and Technology, v. 1, p. 161–171(10), October 2007. ISSN 1751-8806. Disponível em: <http://digital-library.theiet.org/content/journals/10.1049/iet-sen_20060052>. Citado na página 72.

LEE, Y. S. et al. Systematic evaluation methodology for cell phone user interfaces. *Interacting with Computers*, v. 18, n. 2, p. 304–325, 2006. Disponível em: <+ http://dx.doi.org/10.1016/j.intcom.2005.04.002>. Citado na página 114.

LI, M.; SMIDTS, C. S. A ranking of software engineering measures based on expert opinion. *IEEE Transactions on Software Engineering*, v. 29, n. 9, p. 811–824, Sept 2003. ISSN 0098-5589. Citado 4 vezes nas páginas 72, 73, 74 e 75.

MACK, R. L.; NIELSEN, J. *Usability inspection methods*. [S.l.]: Wiley & Sons New York, 1994. 25–62 p. Citado na página 113.

NETO, O. M.; PIMENTEL, M. d. G. Heuristics for the assessment of interfaces of mobile devices. In: *Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2013. (WebMedia '13), p. 93–96. ISBN 978-1-4503-2559-2. Disponível em: <http://doi.acm.org/10.1145/2526188.2526237>. Citado na página 113.

ORTIZ, N. et al. Use of expert judgment in nureg-1150. *Nuclear Engineering and Design*, v. 126, n. 3, p. 313 – 331, 1991. ISSN 0029-5493. Disponível em: <http://www.sciencedirect.com/science/article/pii/002954939190023B>. Citado na página 74.

PERKINS, D. et al. Simba: Tunable end-to-end data consistency for mobile apps. In: *Proceedings of the Tenth European Conference on Computer Systems*. New York, NY, USA: ACM, 2015. (EuroSys '15), p. 7:1–7:16. ISBN 978-1-4503-3238-5. Disponível em: <http://doi.acm.org/10.1145/2741948.2741974>. Citado 10 vezes nas páginas 13, 20, 34, 35, 36, 37, 45, 127, 128 e 129.

PETERS, M. et al. A client centric replication model for mobile environments based on restful resources. In: *Proceedings of the Workshop on Posters and Demos Track*. New York, NY, USA: ACM, 2011. (PDT '11), p. 22:1–22:2. ISBN 978-1-4503-1073-4. Disponível em: <http://doi.acm.org/10.1145/2088960.2088982>. Citado 7 vezes nas páginas 13, 20, 42, 43, 44, 45 e 127.

PICCO, G. P. et al. Software engineering for mobility: Reflecting on the past, peering into the future. In: *Proceedings of the on Future of Software Engineering*. New York, NY, USA: ACM, 2014. (FOSE 2014), p. 13–28. ISBN 978-1-4503-2865-4. Disponível em: <http://doi.acm.org/10.1145/2593882.2593884>. Citado 4 vezes nas páginas 17, 31, 32 e 33.

ROMAN, G.-C.; PICCO, G. P.; MURPHY, A. L. Software engineering for mobility: A roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA: ACM, 2000. (ICSE '00), p. 241–258. ISBN 1-58113-253-0. Disponível em: <http://doi.acm.org/10.1145/336512.336567>. Citado na página 31.

SAHNI, S.; RAMAMRITHAM, K. Delay tolerant applications for low bandwidth and intermittently connected users: The aaqua experience. In: *Proceedings of the 16th International Conference on World Wide Web*. New York, NY, USA: ACM, 2007. (WWW '07), p. 1117–1118. ISBN 978-1-59593-654-7. Disponível em: <http://doi.acm.org/10.1145/1242572.1242722>. Citado 5 vezes nas páginas 13, 17, 39, 41 e 45.

SALGADO, A. de L.; RODRIGUES, S. S.; FORTES, R. P. M. Evolving heuristic evaluation for multiple contexts and audiences: Perspectives from a mapping study. In: *Proceedings of the 34th ACM International Conference on the Design of Communication*. New York, NY, USA: ACM, 2016. (SIGDOC '16), p. 19:1–19:8. ISBN 978-1-4503-4495-1. Disponível em: <http://doi.acm.org/10.1145/2987592.2987617>. Citado na página 113.

TVERSKY, A.; KAHNEMAN, D. Judgment under uncertainty: Heuristics and biases. In: _____. *Utility, Probability, and Human Decision Making: Selected Proceedings of an Interdisciplinary Research Conference, Rome, 3–6 September, 1973*. Dordrecht: Springer Netherlands, 1975. p. 141–162. ISBN 978-94-010-1834-0. Disponível em: <https://doi.org/10.1007/978-94-010-1834-0_8>. Citado na página 74.

APÊNDICE A – Questionário de Avaliação da Biblioteca *OfflineManager*

Questionário de Avaliação da Biblioteca OfflineManager

Este formulário leva em média 15 minutos para ser respondido.

As respostas serão mantidas em TOTAL ANONIMATO, incluindo o seu nome, email ou identificação.

Sobre a Biblioteca:
O OfflineManager é uma biblioteca para encapsular chamadas do Retrofit, com a finalidade de fornecer diferentes níveis de tratamento para quando o celular do usuário estiver sem conexão, ou o servidor não estiver respondendo. A chamada para o endpoint é feita com o método treatedCall, e a parametrização é que define como serão os tratamentos para cada caso, o nível de feedbacks para o usuário, número de tentativas e os callbacks do Retrofit (onResponse e onFailure).

Para mais informações, veja a documentação completa em https://github.com/Sidarta/OfflineManager bem como instruções para instalação via Gradle da biblioteca.

*Obrigatório

1. **Endereço de e-mail** *

2. **Nome Completo (opcional)**

3. **Área de trabalho** *
   Marcar apenas uma oval.
   - ◯ Academia
   - ◯ Empresa
   - ◯ Outro: _____

4. **Descrição da atuação principal** *

5. **Experiência com Java** *
   Marcar apenas uma oval.
   - ◯ Nenhuma experiência
   - ◯ Menos de 1 ano
   - ◯ 1 - 2 anos
   - ◯ 2 - 3 anos
   - ◯ 3 - 4 anos
   - ◯ Mais do que 4 anos

6. **Experiência com Android** *
Marcar apenas uma oval.

- ◯ Nenhuma experiência
- ◯ Menos de 1 ano
- ◯ 1 - 2 anos
- ◯ 2 - 3 anos
- ◯ 3 - 4 anos
- ◯ Mais do que 4 anos

7. **Experiência com aplicações REST** *
Marcar apenas uma oval.

- ◯ Nenhuma experiência
- ◯ Pouco contato
- ◯ Contato médio: entendo o conceito, mas utilizei poucas vezes
- ◯ Contato frequente: entendo e já utilizei bastante em minha área de atuação
- ◯ Contato diário: utilizo diariamente nas aplicações relativas ao meu trabalho e atuação
- ◯ Grande experiência e contato diário: além de utilizar diariamente, possuo mais de 4 anos de experiência

8. **Experiência com a biblioteca Retrofit** *
Marcar apenas uma oval.

- ◯ Nenhuma experiência
- ◯ Menos de 1 anos
- ◯ 1 - 2 anos
- ◯ 2 - 3 anos
- ◯ 3 - 4 anos
- ◯ Mais do que 4 anos

9. **Quantidade de projetos utilizando Retrofit** *
Marcar apenas uma oval.

- ◯ 0 projetos
- ◯ 1 projeto pequeno-médio
- ◯ 1 projeto grande
- ◯ 1 - 3 projetos pequenos ou médios
- ◯ 1 - 3 projetos grandes
- ◯ Mais do que 3 projetos pequenos ou médios
- ◯ Mais do que 3 projetos grandes

10. **Como desenvolvedor, você já se deparou com uma situação onde precisou oferecer suporte offline para uma aplicação?** *
    Funcionamento offline significa a capacidade de o usuário conseguir utilizar a aplicação (ou parte dela) mesmo quando o dispositivo estiver sem conexão com a internet.
    Marcar apenas uma oval.

    - ◯ Não, nunca me deparei com essa situação.
    - ◯ Sim, já me deparei com essa situação uma única vez, mas em um caso muito específico.
    - ◯ Sim, já me deparei com essa situação mais de uma vez.
    - ◯ Sim, me deparo com essa situação frequentemente.
    - ◯ Sim, me deparo com essa situação basicamente em todos os projetos que participo.

11. **Caso a resposta anterior tiver sido SIM, como você implementou essa necessidade?**

OfflineManager

Questões a serem respondidas após a aplicação da biblioteca OfflineManager

12. **Dificuldade para instalar a Biblioteca** *
    Marcar apenas uma oval.

    - ◯ Difícil
    - ◯ Intermediário
    - ◯ Fácil
    - ◯ Trivial

13. **Dificuldade para entender a biblioteca** *
    Marcar apenas uma oval.

    - ◯ Não entendi como ela deve ser usada
    - ◯ Entendi, mas demorei e gastou tempo de desenvolvimento para aprender.
    - ◯ Entendi, tempo moderado de desenvolvimento para aprender.
    - ◯ Entendi de forma rápida

14. **Dificuldade em inicializar e aplicar a biblioteca a uma primeira chamada** *
    Marcar apenas uma oval.

    - ◯ Difícil
    - ◯ Dificuldade média
    - ◯ Fácil
    - ◯ Trivial

15. **Dificuldade em encontrar uma chamada (call) em que a biblioteca possa ser usada** *
Marcar apenas uma oval.

- ◯ Difícil para identificar pontos de utilização
- ◯ Dificuldade média para identificar pontos de utilização
- ◯ Fácil para identificar pontos de utilização
- ◯ Trivial para identificar pontos de utilização

16. **A biblioteca oferece 3 tratamentos diferentes, descritos abaixo, para a chamada quando o DISPOSITIVO estiver sem conexão com a internet. Eles são suficientes para resolver a necessidade de funcionamento offline?** *
Os tratamentos oferecidos são - 1. ENFORCE: conexão obrigatória, a chamada não será executada em caso de falta de conexão. 2. FLEXIBLE: pop-up interage com o usuário, perguntando se o usuário deseja que a chamada seja armazenada e executada em segundo plano quando o celular adquirir conexão. 3. TRANSPARENT: chamada é armazenada e executada em segundo plano (quando o celular adquirir conexão) de forma transparente e automática.
Marcar apenas uma oval.

- ◯ Sim, mais do que suficientes: alguns dos tratamentos nem chegam a ser necessários.
- ◯ Sim, oferece tratamentos suficientes.
- ◯ Não, faltam entre 1 a 2 tipos novos de tratamentos.
- ◯ Não, faltam mais que 2 tipos novos de tratamentos.
- ◯ Outro: _____

17. **A biblioteca oferece 3 tratamentos diferentes, descritos abaixo, para a chamada quando o SERVIDOR não estiver respondendo. Eles são suficientes para resolver uma necessidade de melhoria em caso de servidor intermitente ou offline?** *
Os tratamentos oferecidos são - 1. NOACTION: nada será feito caso o servidor estiver offline. 2. FLEXIBLE: pop-up interage com o usuário perguntando se ele deseja armazenar a chamada pra que ela seja tentada novamente, algumas vezes, até o servidor responder, ou acabar o número de tentativas. Cada tentativa, um novo pop -up interage com o usuário, para que ele decida, em cada tentativa, o que será feito. 3. TRANSPARENT: chamada será armazenada e tentada novamente algumas vezes, de forma transparente, até o servidor responder, ou acabar o número de tentativas. Obs.: o número exato de tentativas é especificado via parâmetro na chamada treatedCall.
Marcar apenas uma oval.

- ◯ Sim, mais do que suficientes: alguns nem chegam a ser necessários.
- ◯ Sim, oferece tratamentos suficientes.
- ◯ Não, faltam entre 1 a 2 tipos novos de tratamentos.
- ◯ Não, faltam mais que 2 tipos novos de tratamentos.
- ◯ Outro: _____

18. **Sobre as mensagens de feedback oferecidas (com o modo verboso = TRUE)** *
caso seja FALSE - são mostradas apenas mensagens estritamente necessárias: mensagem para o caso Enforce de celular sem conexão, e os pop-ups de interação dos modos Flexible (celular e servidor). Caso seja TRUE - todas as interações são seguidas por feedback informativo ao usuário. Obs.: o modo Transparent (tanto para celular quanto servidor offline) não mostra mensagens mesmo com verbose = true, pois por definição, ele executa as ações de forma transparente ao usuário.
Marcar apenas uma oval.

- ◯ Faltam mais que 2 mensagens diferentes
- ◯ Faltam 1 ou 2 mensagens diferentes
- ◯ Mensagens oferecidas são adequadas
- ◯ 1 ou 2 mensagens desnecessárias
- ◯ Mais do que 2 mensagens são desnecessárias
- ◯ Outro: _____

19. **Em sua opinião, como fica a experiência de usuário em apps com a biblioteca OfflineManager?** *
Marcar apenas uma oval.

- ◯ Péssima experiência de usuário depois de aplicar a biblioteca.
- ◯ Ruim a experiência de usuário depois de aplicar a biblioteca.
- ◯ Não altera a experiência já oferecida pelo app em si.
- ◯ Boa experiência de usuário depois de aplicar a biblioteca.
- ◯ Ótima experiência de usuário depois de aplicar a biblioteca

20. **Em sua opinião, a biblioteca OfflineManager é aplicável?** *
Marcar apenas uma oval.

- ◯ Sim.
- ◯ Sim, mas com algumas melhorias
- ◯ Apenas em alguns casos.
- ◯ Não

21. **Em sua opinião, existe algum problema em usar a biblioteca OfflineManager?** *
Marcar apenas uma oval.

- ◯ Sim, mais do que 1 problema.
- ◯ Sim, 1 problema.
- ◯ Não.

22. **Se sua resposta foi Sim para a pergunta anterior, liste os problemas identificados:** *

_____

_____

_____

_____

_____

23. **Liste melhorias que você identificou que podem ser feitas na biblioteca:** *

_____

_____

_____

_____

_____

24. **Em uma escala de 0 a 10, quanto você acha que a biblioteca OfflineManager pode ajudar nesses requisitos de funcionamento offline?** *
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
A biblioteca não ajuda em nada	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Resolve todos os requisitos de funcionamento Offline

25. **Em uma escala de 0 a 10, quanto você acha VIÁVEL aplicar a biblioteca OfflineManager em um projeto real?** *

Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Não é viável	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Muito viável

☐ Envie para mim uma cópia das minhas respostas.