UNIVERSIDADE FEDERAL DE SÃO CARLOS

DEPARTAMENTO DE COMPUTAÇÃO
ENGENHARIA DE COMPUTAÇÃO

Amanda Basso de Oliveira

Modelagem de arquitetura baseada em microsserviços com foco em segurança

São Carlos - SP

2022

Amanda Basso de Oliveira

Modelagem de arquitetura baseada em microsserviços com foco em segurança

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação da Universidade Federal de São Carlos, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientação Prof. Dr. Hélio Crestana Guardia

São Carlos - SP

2022

Dedico este trabalho a todas as pessoas que, direta ou indiretamente, contribuíram para mais essa fase de sucesso na minha vida.

Agradecimentos

Agradeço, primeiramente, aos meus pais, Carlos e Renilda, pelo apoio incondicional ao longo de todas as jornadas de minha vida. Eles são os principais responsáveis pela minha formação acadêmica.

Agradeço a todos os meus amigos, em especial às minhas amigas Ana, Laís, Mariane e Rafaela. Vocês são uma fonte de inspiração e força contínuas.

Ao meu namorado, Gabriel, agradeço pela companhia e pelo incentivo. Obrigada por tornar a jornada mais leve e agradável.

Também agradeço ao professor Dr. Hélio Crestana Guardia por ter me apresentado ao tema, pela orientação e por todos os ensinamentos.

Por fim, agradeço a todos meus amigos de faculdade pelo suporte, convivência e experiências compartilhadas. Eles agregaram enormemente à minha jornada como aluna do curso de Bacharelado em Engenharia de Computação.

"Nada na vida deve ser temido, somente compreendido. Agora é hora de compreender mais para temer menos.
(Marie Curie)

Resumo

Os dados são protagonistas da Quarta Revolução Industrial, uma vez que permitem que o mundo digital se torne cada vez mais conectado e inteligente. Entre os diversos tipos de dados, pode-se destacar os dados pessoais e sensíveis, que exigem maior cautela no tratamento, devido às informações que podem revelar. Nesse contexto, urge considerar os impactos negativos do roubo e do vazamento de dados por ciber-criminosos que, além das implicações legais e jurídicas resultantes de leis como a Lei Geral de Proteção de Dados Pessoais (LGPD), podem ser financeiramente e moralmente catastróficos a empresas, governos e indivíduos.

Do ponto de vista do armazenamento e acesso remoto a dados sensíveis, a arquitetura do software utilizado pode ter papel crucial na segurança de um sistema. Atualmente, diversas aplicações são desenvolvidas seguindo o estilo arquitetural conhecido como Arquitetura Baseada em Microsserviços(O'REILLY...,). Sua popularidade deve-se, principalmente, às suas vantagens, dentre as quais pode-se citar a autonomia das entidades que constituem a arquitetura, o baixo acoplamento e a tolerância a falhas. O modelo baseado em microsserviços permite diferentes formas de implementação, vista sua alta flexibilidade, o que pode impactar na segurança do sistema resultante.

Assim, este projeto visa criar uma proposta de arquitetura baseada em microsserviços que lide com dados sensíveis, baseando-se nas necessidades do projeto Amive. Os resultados obtidos incluem uma proposta de arquitetura que considera aspectos relacionados à segurança, como gestão da identidade e do acesso e encriptação de dados, e à escalabilidade, ao mesmo tempo que englobam questões de segurança a serem tratadas em trabalhos futuros.

Palavras-chave: Segurança da informação, Arquitetura baseada em microsserviços, projeto Amive, Gestão da Identidade e do Acesso, Segurança de dados em repouso, Segurança de dados em trânsito.

Abstract

Data are key players in the Fourth Industrial Revolution as they enable the digital world to become increasingly connected and intelligent. Among the various types of data, personal and sensitive data demand greater caution throughout the treatment due to the information they can reveal. In this context, it is urgent to consider the negative impacts of data leakage and theft by cyber criminals, which, in addition to the legal and juridical implications resulting from laws such as the Brazilian General Data Protection Act, might be financially and morally catastrophic to companies, governments, and individuals.

From the point of view of remote storage and access to sensitive data, the architecture of the software can play a crucial role in the security of a system. Today, several applications are developed following the architectural style known as Microservices-Based Architecture(O'REILLY...,). Its popularity is due, mainly, to its advantages, among which are the autonomy of the entities that constitute the architecture, low coupling, and fault tolerance. The model based on microservices allows different forms of implementation, given its high flexibility, which can impact the security of the resulting system.

Thus, this project aims to create an architecture proposal based on microservices that deals with sensitive data, based on the needs of the Amive project. The results obtained include an architecture proposal that considers aspects related to security, such as identity and access management and data encryption, and scalability, thus encompassing security issues to be addressed in future works.

Keywords: Information Security, Microservices-based Architecture, Amive project, Identity and access management, Data at rest security, Data in transit security.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

A Quarta Revolução Industrial trouxe uma miríade de tecnologias que transformam, continuamente, os mundos físico, digital e biológico. Avanços em Inteligência Artificial, Robótica, Engenharia Genética, Internet das Coisas (IoT, da sigla em inglês), Computação Quântica, entre outros, são responsáveis por modificações intensas à vida moderna, ao ponto de se tornarem indispensáveis.

Os dados, por sua vez, são protagonistas dessa revolução, uma vez que permitem e impulsionam o desenvolvimento das áreas citadas. Dados são usados para aprimoramento dos modelos de Aprendizado de Máquina e para troca de informações entre dispositivos interconectados, dentro do contexto de IoT e da Robótica. Eles permitem que o mundo digital se torne cada vez mais conectado e inteligente e é por causa deles que os sistemas tecnológicos sofrem recrudescentes melhorias e ampliam seu alcance, tornando-se ubíquos.

Entre os diversos tipos de dados, pode-se destacar os dados pessoais, que exigem maior cautela no tratamento, devido às informações que podem revelar. Dados pessoais e sensíveis são tratados continuamente por diversas aplicações que, além de prezar pela segurança cibernética destes, devem seguir leis e políticas de *compliance*.

Nesse contexto, urge considerar os impactos negativos do vazamento e roubo de dados por ciber-criminosos que, além das implicações legais e jurídicas resultantes de leis como a Lei Geral de Proteção de Dados Pessoais (LGPD), são financeiramente e moralmente catastróficos a empresas, governos e indivíduos.

Do ponto de vista do armazenamento e acesso remoto a dados sensíveis, a arquitetura do software utilizado pode ter papel crucial na segurança de um sistema. Atualmente, os indivíduos lidam, diariamente, com aplicações que cumprem os mais diversos objetivos. Elas são usadas com a finalidade de realizar transferências bancárias, consultar saldos, realizar compras online e até mesmo aprender novos idiomas. Inúmeros dados são processados por essas aplicações que, invariavelmente, estão sujeitas aos impactos negativos de vazamento ou roubo de dados previamente mencionados.

O modelo arquitetural baseado em microsserviços permite diferentes formas de implementação, vista sua alta flexibilidade, o que pode impactar na segurança do sistema resultante. Cada vez mais popular (O'REILLY. . . ,), este estilo apresenta vantagens, dentre as quais é possível citar autonomia, baixo acoplamento, reusabilidade, componibilidade e tolerância a falhas. Ele é sobretudo utilizado para aplicações que usam contêineres e para aplicações em nuvem, devido às vantagens citadas.

Assim, este projeto visa criar uma proposta de arquitetura baseada em microsser-

viços que lide com dados sensíveis, baseando-se nas necessidades do projeto Amigo Virtual Especializado (Amive)[1] desenvolvido por pesquisadores da Universidade Federal de São Carlos (UFSCar).

De forma resumida, o projeto Amive visa construir uma solução para identificação automática de um possível perfil depressivo (PPD). Para tanto, pressupõe a construção de um modelo multifatorial fundamentado na área da Saúde Mental.

A solução lidará com dados sensíveis fisiológicos, como frequência cardíaca e informações referentes ao monitoramento de sono, além de dados sensíveis que armazenam conteúdo de posts de Facebook dos usuários. Estes dados serão coletados e tratados pela infraestrutura computacional para treinamento do modelo e, em seguida, para identificação de PPDs.

Portanto, urge criar uma arquitetura de solução robusta da perspectiva da segurança e da escalabilidade. Nesse sentido, utilizar o projeto Amive como cenário para este trabalho agrega à pesquisa aqui apresentada, porque fornece o contexto para a proposta da solução e o ambiente com pesquisadores multidisciplinares, e corrobora com os propósitos do projeto Amive, concomitantemente.

1.1 Objetivos

Este trabalho tem o objetivo de propor uma arquitetura de aplicação baseada em microsserviços que englobe melhores práticas de segurança. Para tanto, este projeto se apoiará sobre três pilares fundamentais: a gestão da identidade e do acesso (IAM, da sigla em inglês), a proteção dos dados levando em consideração aspectos legais como a LGPD e a segurança da arquitetura.

O projeto Amive, previamente citado, será utilizado como embasamento e referência, mas as boas práticas de segurança pesquisadas para este trabalho podem ser utilizadas em diversas soluções.

1.2 Justificativa

O presente trabalho estuda as melhores práticas de segurança relacionadas tanto ao tratamento de dados, sensíveis ou não, quanto ao desenvolvimento de arquitetura baseada em microsserviços. O intuito é propor uma arquitetura de sistema baseada em microsserviços com foco em segurança e escalabilidade, levando em consideração aspectos apresentados ao longo dos capítulos 2 e 3.

[1] <https://www.amive.ufscar.br/projeto>

Conquanto exista extensa literatura sobre segurança de aplicações, que une perspectivas da gestão da identidade e do acesso e da arquitetura da solução, ainda há poucos estudos de casos que englobem a segurança destes pontos de vista com as questões levantadas pela Lei Geral de Proteção de Dados.

A revisão bibliográfica contém, dentre outros, a publicação do *National Institute of Standards and Technology* (NIST) intitulada *"Security Strategies for Microservices-based Application Systems"*(CHANDRAMOULI, 2019), diretamente relacionada ao pilar de segurança da arquitetura. Além disso, diversos outros materiais acerca da gestão da identidade e do acesso (IAM, da sigla em inglês) foram encontrados, como documentação online de IAM da *Amazon Web Services* (AWS)(IAM. . . ,). A Lei 13.709/2018, também conhecida como LGPD, é referida múltiplas vezes na revisão bibliográfica para balizar o tratamento de dados em contexto brasileiro.

1.3 Organização do trabalho

Este trabalho está dividido em cinco capítulos. No capítulo 1, são apresentados a proposta de pesquisa, os objetivos e a justificativa. O capítulo 2, por sua vez, compreende a fundamentação teórica necessária para a realização da pesquisa, passando por conceitos de arquiteturas baseadas em microsserviços, por breve descrição da Lei Geral de Proteção de Dados Pessoais (LGPD), por introdução à compreensão de ameaças a arquiteturas e pelo projeto Amive, inspiração para este projeto.

No capítulo 3, é realizada uma sumarização de melhores práticas, dividida em três pilares: gestão da identidade e do acesso, proteção dos dados e segurança da arquitetura baseada em microsserviços. O capítulo 4 contém a proposta de arquitetura de sistema baseada em microsserviços com foco em segurança e escalabilidade resultante deste trabalho. Por último, no capítulo 5, são expostas as conclusões do trabalho e possíveis temas de pesquisa que podem ser explorados futuramente a partir deste projeto.

2 Fundamentação Teórica

Este capítulo apresenta os principais conceitos que embasam o presente trabalho. Eles incluem os fundamentos de arquiteturas baseadas em microsserviços, os quais compreendem aspectos arquiteturais, mecanismos de comunicação e contêineres.

Além disso, os principais pontos da LGPD são apresentados, seguidos por descrição sobre frameworks utilizados por profissionais de segurança para modelagem de ameaças em aplicações. Por fim, é realizada descrição do projeto Amive, base para este trabalho.

2.1 Arquitetura baseada em Microsserviços

A arquitetura de aplicações monolítica é representada por um modelo unificado, projetado individualmente para cada aplicação. As funcionalidades do software são implementadas por várias entidades altamente acopladas e sem responsabilidade bem definida (AL-DEBAGY; MARTINEK, 2018).

Conquanto tais aplicações cumpram com seu objetivo, a manutenção e a alteração de software nesse estilo arquitetural são complexas e altamente custosas, dado o emaranhado de código que se forma a partir de determinada quantidade de linhas de código (LOC, da sigla em inglês) (KRUIDENBERG, 2018).

Não obstante, códigos complexos e emaranhados apresentam um empecilho para implementação contínua (*continous deployment*, em inglês) e para execução eficiente de metodologias ágeis, uma vez que o processo de adição e alteração de software é mais tortuoso e vagaroso. Além disso, escalar aplicações monolíticas torna-se um trabalho difícil visto que os diferentes módulos internos da aplicação podem tornar-se incompatíveis com o passar do tempo (CHANDRAMOULI, 2019).

Aplicações baseadas em microsserviços, em contraposição, são constituídas por múltiplas entidades, os microsserviços, que se comunicam síncrona ou assincronamente e possuem baixo acoplamento entre si.

O estilo arquitetural baseado em microsserviços é conduzido pelos seguintes guias (RICHARDSON; SMITH, 2016):

- Os microsserviços devem ser gerenciados, escalonados, replicados, modificados e implementados independentemente de outros microsserviços.
- Cada microsserviço deve seguir o Princípio da Responsabilidade Única, que define que cada módulo, classe ou função deve ter uma única responsabilidade, ou seja,

deve tratar apenas de uma funcionalidade.

- Os microsserviços devem ser projetados de forma a serem tolerantes a falhas.
- Serviços confiáveis como bancos de dados e caches devem ser utilizados para gerenciamento de estado.

Através dos pontos-chave mencionados, é possível concluir que os princípios de projeto para microsserviços são, entre outros, a autonomia entre as entidades, baixo acoplamento, reusabilidade, componibilidade, tolerância a falhas e alinhamento de Interfaces de Programação de Aplicações (APIs, da sigla em inglês) com processos de negócios (CHANDRAMOULI, 2019).

Além disso, é importante ressaltar que a autonomia entre os microsserviços possibilita que equipes diferentes de um mesmo projeto trabalhem em cada um deles separadamente, sem limitações em relação às tecnologias adotadas e às entregas das outras equipes. A única ressalva é que devem ser implementadas APIs para que os outros microsserviços e clientes possam acessar as funcionalidades oferecidas.

2.1.1 Comunicação entre Microsserviços

Em aplicações baseadas em microsserviços, normalmente cada microsserviço é um processo distinto que se comunica com outros microsserviços através de mecanismos de comunicação interprocesso (IPCs, da sigla em inglês). Além disso, microsserviços são definidos através de uma Linguagem de Descrição de Interface (IDL, da sigla em inglês), visto que a IDL é independente de qualquer linguagem de programação e permite a comunicação entre os serviços implementados utilizando diferentes tecnologias e linguagens de programação.

Isso resulta em uma API que, como mencionado, funciona como o meio pelo qual serviços e clientes comunicam-se entre si. APIs são peças-chave em uma aplicação baseada em microsserviços pois é por meio delas que os serviços distintos ligam-se entre si.

O mecanismo de IPC escolhido é responsável pela natureza da definição de API (RICHARDSON; SMITH, 2016). Mecanismos assíncronos, baseados em mensagens, resultam em APIs feitas por canais de mensagem e tipos de mensagem. Respostas síncronas, por outro lado, têm como consequência APIs cujos modelos seguem o formato de requisição e resposta, como no caso dos Localizadores de Recursos Uniformes (URLs, da sigla em inglês).

A comunicação entre microsserviços distintos, geralmente com o propósito de compartilhar dados ou atualizar informações, pode ser estabelecida através de padrões de interação (CHANDRAMOULI, 2019). Os principais padrões de interação estão definidos a seguir:

- ***Request-reply:*** este padrão é comumente implementado de forma síncrona, como em chamadas a serviços Web sobre HTTPS. No padrão *request-reply*, um microsserviço realiza uma requisição para obter informação ou para realizar alguma ação e, então, espera por uma resposta. Existe, portanto, uma forte dependência de tempo de execução entre os dois microsserviços comunicantes.

- ***Publish-subscribe:*** também conhecido como abordagem baseada em eventos. Neste padrão, um microsserviço, conhecido como *publisher* transmite mensagens em *broadcast* de forma assíncrona. Os serviços que estiverem registrados para receber mensagens relacionadas àquele evento em específico, conhecidos como *subscribers*, receberão a mensagem.

  A Figura 1, ilustrada abaixo, exemplifica o funcionamento do modelo *publish-subscribe*.

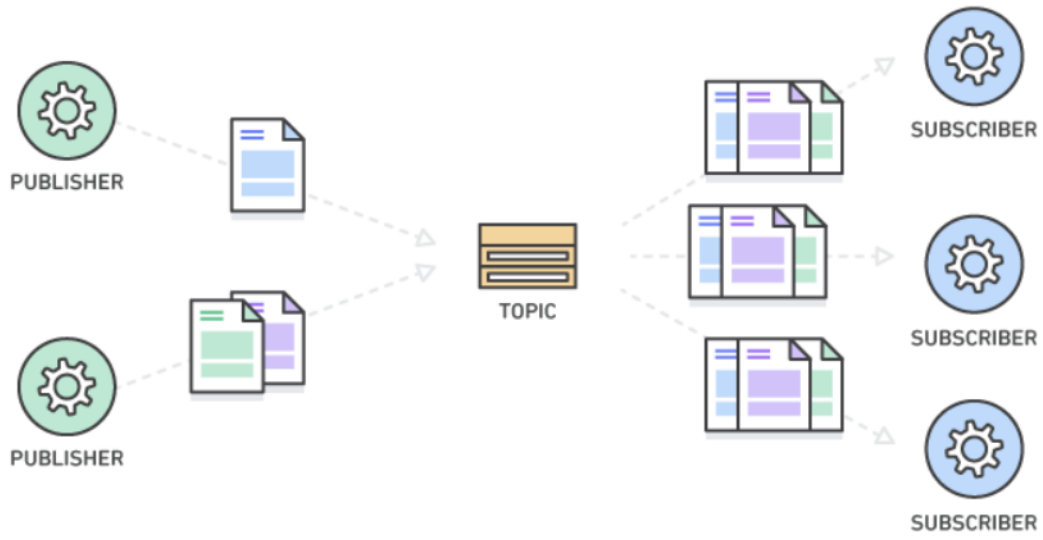

Figura 1 – Comunicação entre serviços seguindo o modelo *publish-subscribe*. Fonte (AWS...,).

2.1.2 Virtualização e conteinerização

Virtualização e conteinerização exercem funções essenciais no desenvolvimento de aplicações, sobretudo porque tornam mais flexível e ágil o desenvolvimento de software, tanto em ambiente de testes quanto de produção (SCHEEPERS, 2014).

Virtualização é uma abstração que transforma a relação “um para um” entre os recursos de hardware de uma máquina e o sistema operacional em "um para muitos". Isso é possível através do hipervisor, um processo que cria e executa máquinas virtuais (VMs, da sigla em inglês). Cada VM inclui uma cópia completa do sistema operacional (SO), da aplicação, das bibliotecas e dos binários necessários.

Conquanto o uso de VMs apresente várias vantagens, como a possibilidade de usar o mesmo hardware para desenvolver e testar aplicações para diferentes ambientes, redução de custos, aumento da produtividade da equipe de desenvolvimento e abastecimento acelerado de aplicações e recursos, o processo de inicialização pode ser lento, pois as VMs podem ocupar bastante espaço de memória RAM e de armazenamento.

A Figura 2 representa um diagrama de funcionamento da virtualização. É possível observar a camada do hipervisor entre a infraestrutura de hardware e as VMs.

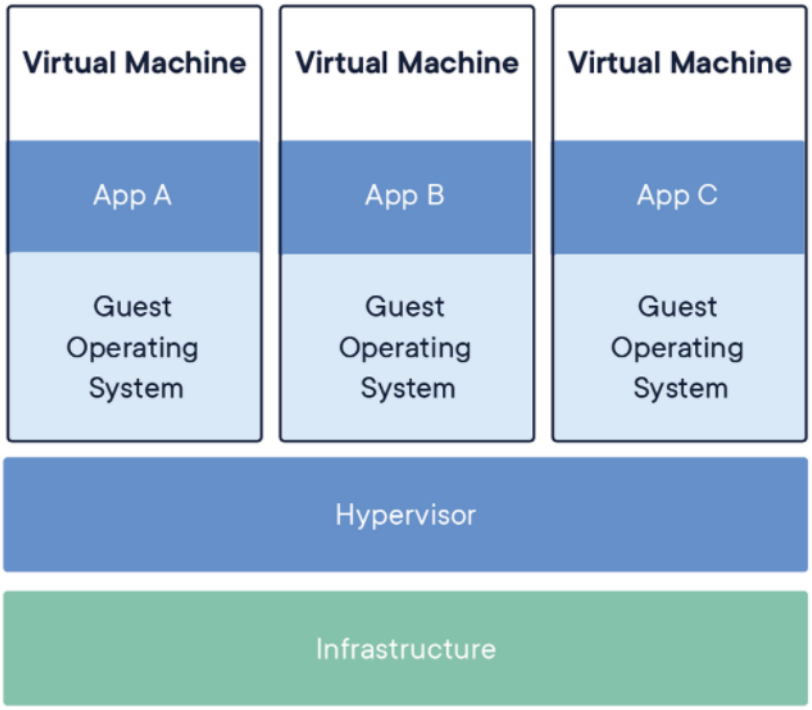

Figura 2 – Representação de virtualização. Fonte (DOCKER. . . ,).

Conteinerização, por sua vez, é o empacotamento de software com as bibliotecas do SO e as dependências necessárias para executar o código, com o objetivo de criar um único executável leve, o contêiner, que executa consistentemente em qualquer infraestrutura. Trata-se de uma abstração ao nível da camada de aplicação que permite múltiplos contêineres executarem na mesma máquina e compartilharem o *kernel* do SO com outros contêineres. Cada um deles executa como processos em espaços de usuário isolados.

A Figura 3 ilustra o funcionamento de aplicações conteinerizadas. É possível observar que a tecnologia de contêiner, representada na imagem por Docker[1], funciona em cima da camada do SO, e as aplicações são executadas em espaços de usuários isolados que compartilham o mesmo SO.

Visto que o contêiner é um ambiente isolado e portátil que possui todas as bibliotecas, dependências e arquivos de configuração para a execução, é possível executar a aplicação em

[1] <https://www.docker.com/>

várias infraestruturas diferentes sem a necessidade de refatorá-la. Ele confere flexibilidade e agilidade aos processos de desenvolvimento e teste de aplicações e, no contexto de soluções baseadas em microsserviços, cada serviço é implementado como contêiner.

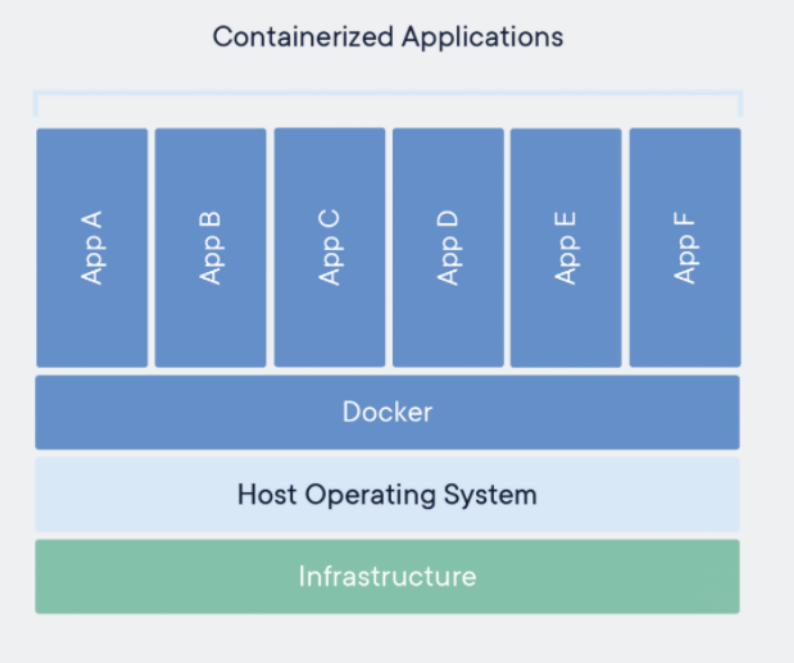

Figura 3 – Representação de conteinerização. Fonte (DOCKER. . . ,).

Contêineres são instâncias em execução de imagens de contêiner. Imagens de contêineres são pacotes executáveis de software que incluem o código, as ferramentas de sistema, bibliotecas de sistema e configurações para executar a aplicação. Elas podem ser armazenadas localmente ou em repositórios, como o *Docker Hub*[2].

2.1.3 Bancos de dados

O armazenamento dos dados é realizado através de bancos de dados, que podem ser de dois tipos: relacional e não-relacional, também conhecidos como SQL e NoSQL, respectivamente.

Bancos de dados SQL são exemplos de bancos de dados relacionais, nos quais os dados são armazenados em tabelas. Eles seguem, como todos bancos de dados relacionais, quatro propriedades relacionadas à transação de dados, conhecidas pelo acrônimo ACID (DEEPAK, 2016):

- **Atomicidade:** é necessário que a transação ocorra por completo ou que não ocorra. Essa propriedade também é conhecida como "tudo ou nada".
- **Consistência:** as transações vão mudar o estado do banco de dados de um estado válido a outro.

[2] <https://hub.docker.com/>

- **Isolamento:** executar transações de forma concorrente deve ter o mesmo resultado de executá-las serialmente.
- **Durabilidade:** uma vez que a transação ocorreu, o banco de dados não pode voltar ao estado anterior.

Bancos de dados SQL são essenciais para soluções que precisam assegurar as propriedades ACID, por exemplo, em aplicações com alto número de transações interdependentes. No entanto, eles apresentam limitações quando se trata de sistemas em constante mudança ou crescimento.

Embora existam diversas técnicas para escalar bancos de dados relacionais, como replicação controlador/agente, os bancos de dados não-relacionais, ou NoSQL, têm melhor desempenho quando o volume de dados é muito grande e a aplicação precisa ser escalada constantemente (WANG; YANG, 2017), embora não possuam a confiabilidade de bancos de dados relacionais.

Além disso, processos de modelagem de dados iterativos e adaptativos funcionam melhor com bancos de dados NoSQL, visto que mudar a estrutura não impactará nos ciclos de desenvolvimento e é possível armazenar diferentes tipos de dados sem a necessidade de planejar os tipos de dados armazenados com antecedência.

Nesse sentido, diz-se que bancos de dados NoSQL seguem as propriedades incluídas no acrônimo BASE (do inglês, *Basically Available*, *Soft state* e *Eventual consistency*):

- **Basicamente disponível:** garante que os dados estarão disponíveis e que haverá uma resposta para cada requisição que, no entanto, pode ser de falha.
- **Estado leve:** o estado do sistema pode mudar com o tempo.
- **Consistência eventual:** caso pare de receber entradas, o sistema se tornará consistente eventualmente.

É importante ressaltar que o conteúdo armazenado em bancos de dados deve ser sempre encriptado, o acesso deve ser concedido somente aos usuários com as devidas permissões atribuídas e, preferivelmente, a chave deve ser armazenada por um gerenciador de segredos. A seção 3 abordará essas questões em maiores detalhes.

2.2 Lei Geral de Proteção de Dados Pessoais

Ao tratar da segurança de dados em uma aplicação, além dos aspectos relacionados a vulnerabilidades e ameaças, é preciso levar em consideração as questões jurídicas e

de *compliance*. Uma da leis mais relevantes no Brasil, nesse contexto, diz respeito ao tratamento de dados e é conhecida como Lei Geral de Proteção de Dados Pessoais (LGPD):

> A Lei Geral de Proteção de Dados Pessoais (LGPD), Lei nº 13.709, de 14 de agosto de 2018, dispõe sobre o tratamento de dados pessoais, inclusive nos meios digitais, por pessoa natural ou por pessoa jurídica de direito público ou privado, com o objetivo de proteger os direitos fundamentais de liberdade e de privacidade e o livre desenvolvimento da personalidade da pessoa natural.(LGPD...,)

O tratamento de dados é considerado qualquer atividade que utiliza um dado pessoal, ou seja, um dado que possibilite a identificação, direta ou indireta, de pessoa natural, como coleta, classificação, utilização, acesso, reprodução, transmissão, distribuição, processamento, arquivamento, armazenamento e eliminação.

Há dois agentes de tratamento dos dados no contexto da LGPD. O primeiro deles é o controlador, definido como pessoa natural ou jurídica, de direito público ou privado, a quem compete as decisões referentes ao tratamento de dados pessoais. O segundo, por outro lado, é o operador, também definido como pessoa natural ou jurídica, de direito público ou privado, mas que realiza o tratamento de dados pessoais em nome do controlador.

Ao desenvolver uma aplicação que lida com dados pessoais, sejam eles sensíveis ou não, urge tratar os dados em alinhamento com a LGPD, de forma a evitar multas, problemas jurídicos e comprometimento de reputação com clientes e/ou terceiros.

É importante ressaltar que quaisquer outras leis e políticas de *compliance* que afetem o desenvolvimento e lançamento da aplicação, devem ser analisadas e seguidas ao longo do desenvolvimento, do lançamento à manutenção de uma aplicação.

2.3 Como entender as ameaças à arquitetura

Visto que o presente trabalho visa unir esforços para propor uma implementação segura de arquitetura, é preciso refletir sobre as formas através das quais uma aplicação pode ser ameaçada ou comprometida.

O processo de identificação e enumeração de potenciais ameaças, sejam elas vulnerabilidades estruturais ou falta de mecanismos de defesa, é chamado de modelagem de ameaças.

Existem diversos frameworks que visam a ajudar profissionais da segurança na identificação de ameaças em aplicações, como o Framework das Quatro Questões (*Four Question Framework*, em inglês)(OWASP..., b). Trata-se de um framework composto

por quatro perguntas que buscam ajudar os profissionais responsáveis pela segurança da informação a realizar a modelagem de ameaças:

- **Avaliar o escopo:** "Em que estamos trabalhando?"
- **Identificar os pontos vulneráveis:** "O que pode dar errado?"
- **Identificar contramedidas ou gerenciar riscos:** "O que podemos fazer sobre isso?"
- **Avaliar o trabalho realizado:** "Fizemos um bom trabalho?"

2.3.1 Avaliar o escopo

A primeira questão busca enfatizar a importância de se entender a aplicação. Isso pode ser realizado de diversas maneiras, como leitura de documentação e ilustração de diagramas de fluxo. A depender do tamanho da aplicação, contudo, pode ser tarefa complexa, sobretudo se os envolvidos na tarefa tiveram apenas contato recente com a aplicação.

No entanto, construir diagramas de entrada e saída da aplicação, em que cada bloco representa um serviço ou módulo da aplicação e apenas as entradas e saídas são evidenciadas, pode ser um ponto de partida. Isso porque esses diagramas, chamados de diagramas de fluxo, focam na relação entre as partes em detrimento do funcionamento interno de cada uma delas.

Uma forma interessante de ilustrar os diagramas de arquitetura é através do modelo C4[3]. Trata-se de um modelo composto por quatro camadas, cada uma progressivamente mais detalhada que a outra: contexto, contêiner, componente e código.

A Figura 4 representa as quatro camadas do modelo C4 e a progressão de detalhes entre uma camada e a outra. A primeira camada, de contexto, é a mais superficial e visa ilustrar a aplicação como uma caixa-preta que interage com usuários e sistemas externos ao receber entradas e retornar saídas. A segunda camada, por sua vez, é a camada de contêineres, na qual os módulos funcionais do sistema são representados como contêineres e ilustrados em alto-nível, juntamente com as comunicações estabelecidas entre eles.

A camada de componentes detalha os aspectos individuais de cada contêiner representado na camada anterior e mostra os componentes que o constitui. Por último, a camada de código é composta por diagramas que podem ser utilizados para detalhar os componentes e mostrar sua implementação.

A depender dos objetivos, é possível desenhar diagramas para todas ou apenas para as camadas selecionadas. A arquitetura proposta para o projeto Amive será representada,

[3] <https://c4model.com/>

no capítulo 4, utilizando as camadas de contexto e de contêineres, visto que o detalhamento exigido para as próximas camadas é mais adequado a uma arquitetura em processo de implementação.

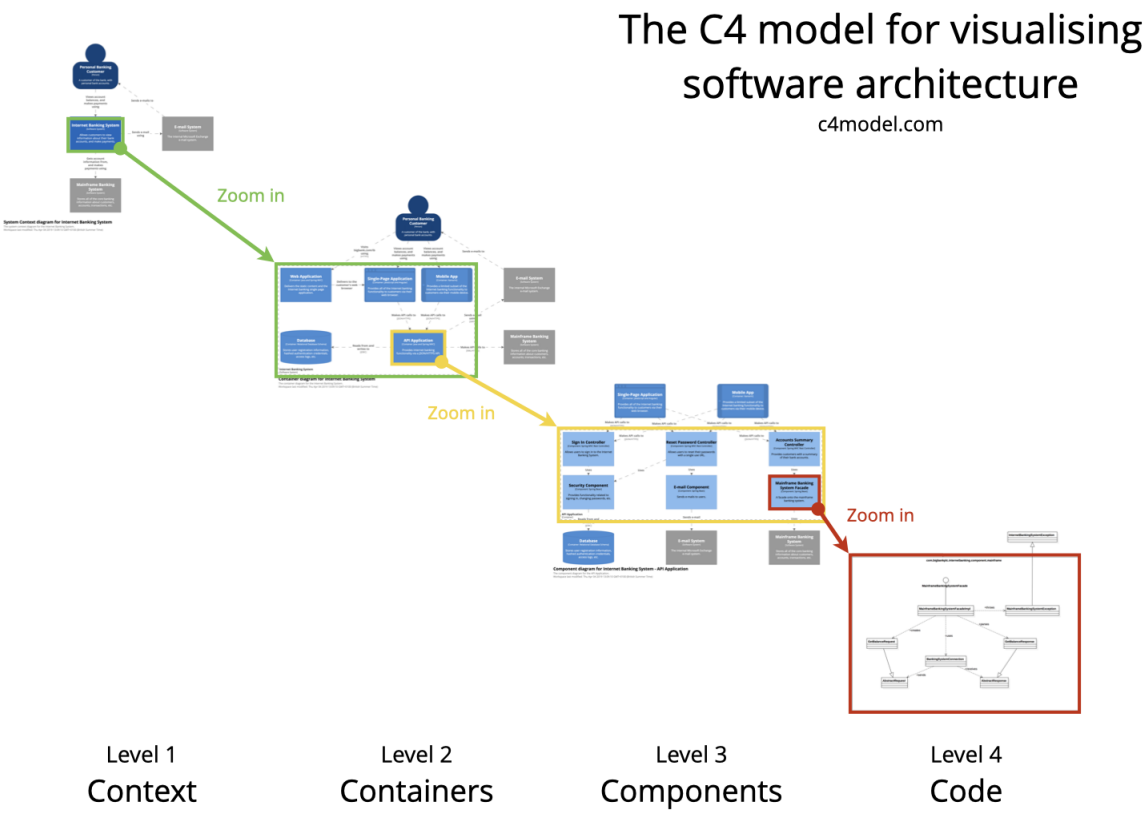

Figura 4 – Esquema de camadas do modelo C4. Fonte: (C4. . . ,).

2.3.2 Identificar pontos vulneráveis

A segunda pergunta, por sua vez, visa identificar potenciais ameaças. Há inúmeras técnicas e metodologias de modelagem que buscam auxiliar os profissionais na identificação e na enumeração de ameaças(THREAT. . . ,), como OCTAVE[4], PASTA[5] e STRIDE[6].

A adoção de um framework de modelagem de ameças específico dependerá do objetivo do profissional de segurança da informação, como modelagem orientada ao desenvolvimento ou à prática. Contudo, para propósitos deste trabalho, adota-se o framework STRIDE para embasar o foco em seis princípios da segurança, descritos na Tabela 1:

STRIDE é um acrônimo para *Spoofing*, *Tampering*, *Information Disclosure*, *Denial of Service* e *Elevation of Privileges*, que em português correspondem, respectivamente,

[4] <https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/rm-ra-methods/m_octave.html>
[5] <https://versprite.com/blog/what-is-pasta-threat-modeling/>
[6] <https://www.microsoft.com/security/blog/2007/09/11/stride-chart/>

Tabela 1 – Framework STRIDE

Propriedade	**Ameaça**	**Definição**	**Exemplo**
Autenticação	Falsificação	Fazer-se passar por algo ou por outra pessoa.	Roubo de nome de usuário e de senha para acessar conta de terceiros.
Integridade	Adulteração	Modificar dados ou código.	Modificar um pacote de dados em trânsito.
Não-repudiação	Repudiação	Alegar não ter realizado uma ação.	Alegar não ter alterado um arquivo.
Confidencialidade	Divulgação de Informações	Exposição de informações a uma parte não autorizada.	Enviar dados para uma parte não autorizada.
Disponibilidade	Recusa de Serviço	Negar ou diminuir serviço a usuários.	Rotear pacotes para um "buraco-negro".
Autorização	Elevação de Privilégios	Adquirir permissões sem a devida autorização.	Permitir que um usuário remoto execute comandos sem autorização.

a falsificação, adulteração, repudiação, divulgação de informações, recusa de serviço e elevação de privilégios, seis formas de ameaçar uma aplicação.

As propriedades relacionadas à segurança que são contrapartida a essas ameaças, são: autenticação, integridade, não-repudiação, confidencialidade, disponibilidade e autorização, como descrito na Tabela 1. Em especial, os princípios de confidencialidade, integridade e disponibilidade formam o conceito da tríade CIA (*CIA triad*, em inglês), conhecido por englobar os princípios primordiais de segurança da informação (RJAIBI; RABAI, 2015).

São essas propriedades que devem sempre ser almejadas, seja ao longo do desenvolvimento de uma aplicação ou durante a fase de manutenção, após o *release*.

Além disso, existem frameworks que objetivam evidenciar os modelos de comportamento de ciber-criminosos, como o MITRE *Adversarial Tactics, Techniques & Common Knowledge* (ATT&CK)[7]. Trata-se de uma base com táticas e técnicas de agentes maliciosos observadas no mundo real e, por esta razão, pode ser usada como um fundamento para o desenvolvimento de modelagem de ameaças específicas para empresas e entidades governamentais.

As Figuras 5 e 6 ilustram, cada uma, partes da matriz ATT&CK. Nelas, é possível visualizar as táticas e técnicas que um agente malicioso pode usar para atacar diversos tipos de aplicações. As empresas podem usá-la como um recurso para entender a metodologia de adversários, propor soluções para mitigar os riscos e implementar soluções de detecção de comprometimento.

[7] <https://attack.mitre.org/>

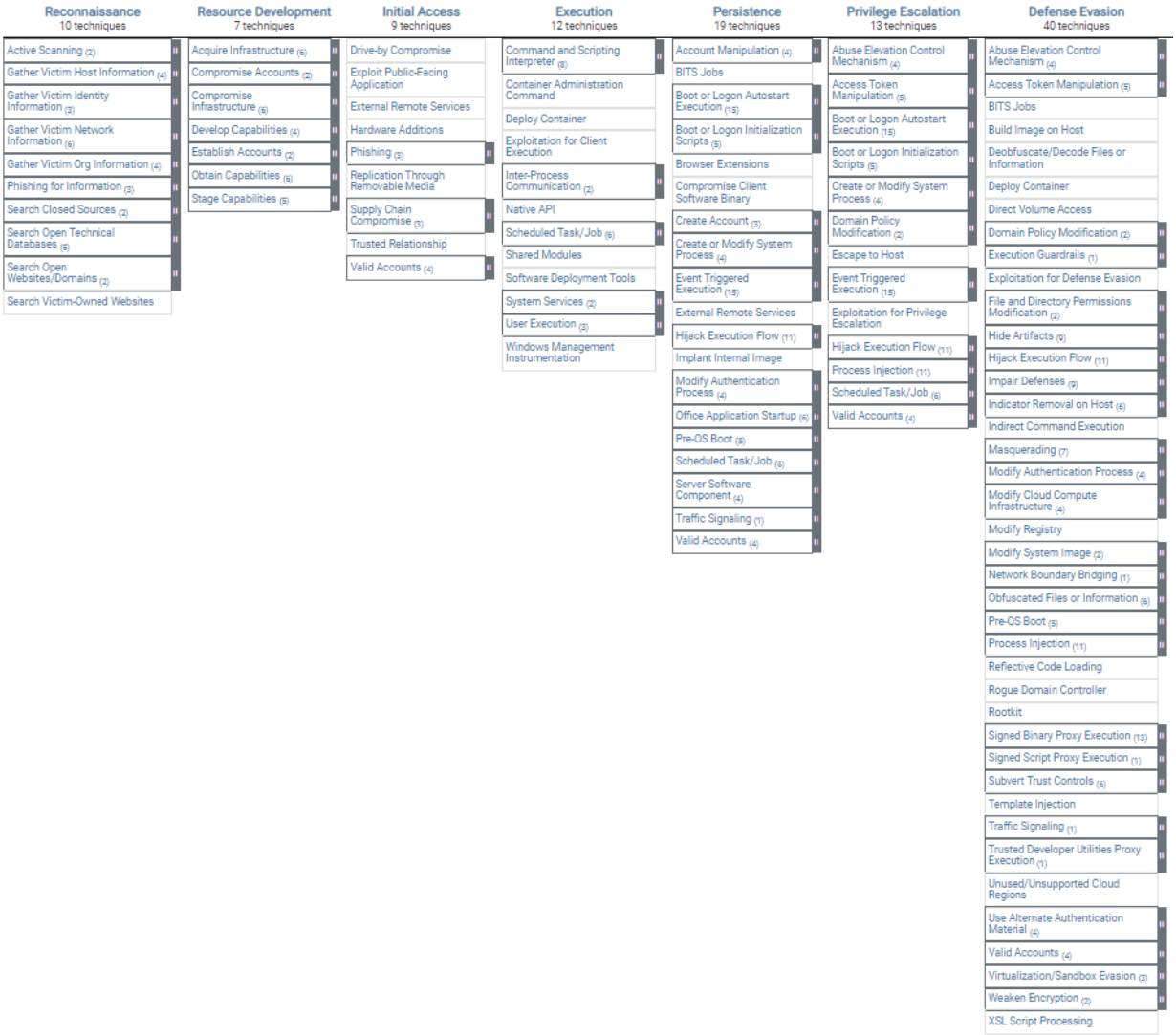

Figura 5 – Primeira parte da matriz MITRE ATT&CK. Fonte: (MITRE. . . ,).

2.3.3 Identificar contramedidas ou gerenciar riscos

A terceira questão busca identificar contramedidas para as ameaças. As opções são:

- **Consertar:** aplicar as medidas cabíveis para remover a funcionalidade vulnerável, por exemplo através de modificação ou remoção de código.
- **Mitigar:** adicionar medidas de segurança extras para reduzir as chances de exploração da vulnerabilidade.
- **Transferir:** deslocar os riscos para outras organizações, por meio de contratos de licenciamento. Nesse caso, a vulnerabilidade não foi excluída.
- **Aceitar:** como no item anterior, aceitar a vulnerabilidade implica em não excluí-la. Nesse caso, é preferível aceitar os riscos a remover, mitigar ou transferir a vulnerabilidade.

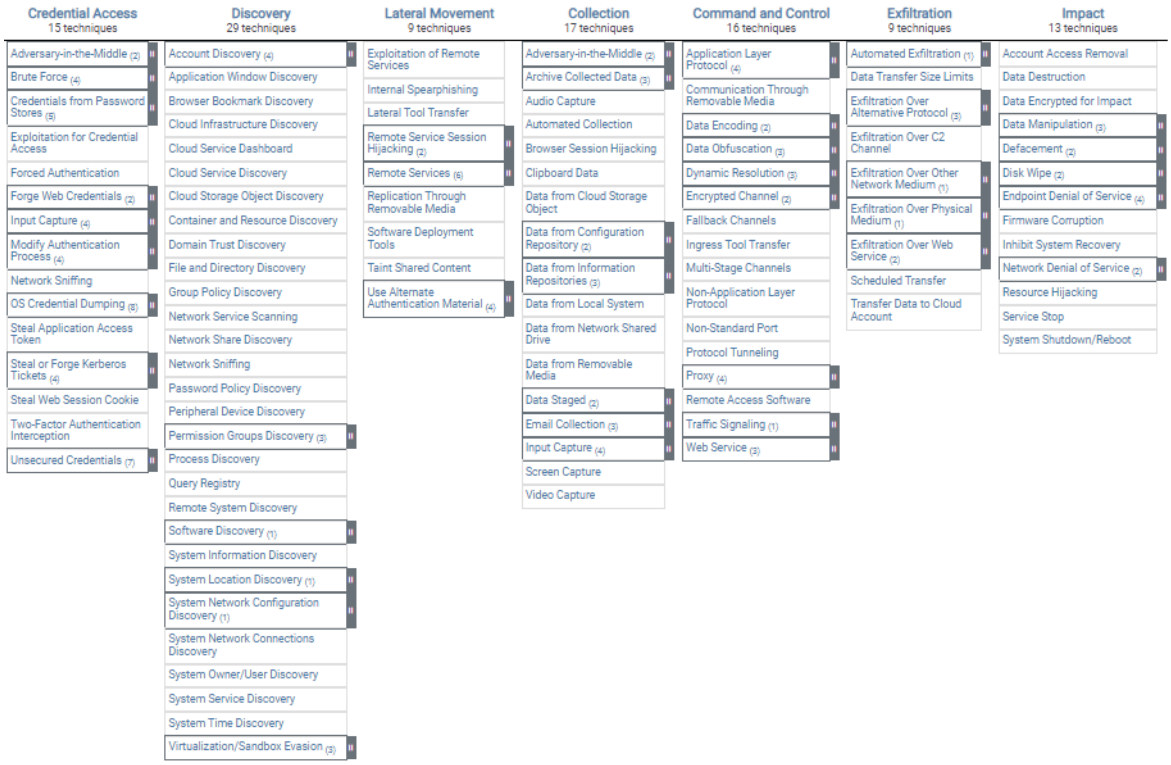

Figura 6 – Segunda parte da matriz MITRE ATT&CK. Fonte:(MITRE. . . ,).

2.3.4 Avaliar o trabalho realizado

Por último, a quarta pergunta busca gerar reflexão sobre o trabalho desenvolvido na modelagem de ameaças. Trata-se de uma etapa de "lições aprendidas"cujo objetivo é tornar o processo mais robusto e identificar possíveis pontos de melhoria.

2.4 Projeto Amive

Como foi introduzido no capítulo 1, a base e inspiração para este trabalho é o projeto Amive. A arquitetura da solução, em construção durante o período de escrita desta pesquisa, deve ser orientada à segurança por conta do tratamento de dados altamente sensíveis, como informações de monitoramento de sono e conteúdo de postagens de Facebook.

O sistema usará canais para coleta de dados. O primeiro deles é um relógio SmartWatch Samsung, que coletará dados fisiológicos como frequência cardíaca, contagem de passos, realização de exercício físico e monitoramento de sono. O usuário deverá baixar os dados coletados pelo relógio através do aplicativo Samsung Health[8] e, em seguida, abrir o aplicativo do Amive para enviar os dados para o sistema. A opção de envio de dados é granular, ou seja, o usuário consegue selecionar quais tipos de dados enviar.

O segundo canal é o Facebook. Os posts publicados pelo usuário serão coletados

[8] <https://play.google.com/store/apps/details?id=com.sec.android.app.shealth&hl=pt_BR&gl=US>

através da API do Facebook e, visto que o servidor terá acesso ao token da API do usuário, a própria aplicação poderá coletar esses dados. O terceiro canal, por sua vez, é o ChatBot que será desenvolvido para comunicação com o usuário e coleta de dados de sentimento subjetivo. Os dados serão enviados para o servidor sempre que houver comunicação entre o ChatBot e o usuário. Na versão final, o ChatBot também validará dados recebidos dos modelos de classificação e os registrará, de forma a retroalimentar o modelo.

Além disso, também serão coletados dados psicométricos dos usuários a partir de 3 questionários. O primeiro deles busca coletar dados relativos a características demográficas, como data de nascimento e forma de ingresso na UFSCar. O segundo, por sua vez, trata-se do PHQ-9, um método utilizado por entrevistadores treinados em estudos populacionais para identificação de PPD e que contém questões relacionadas ao interesse em realizar atividades, ao apetite, à concentração, entre outras. O último questionário é o WHODAS 2.0, que engloba questões sobre cuidado pessoal, relação com pessoas, atividades de vida diária e outras perguntas relacionadas.

A Figura 7 mostra o diagrama alto-nível para a aplicação Amive. Os dados coletados através dos canais mencionados serão processados por módulos de classificação que buscarão identificar PPDs.

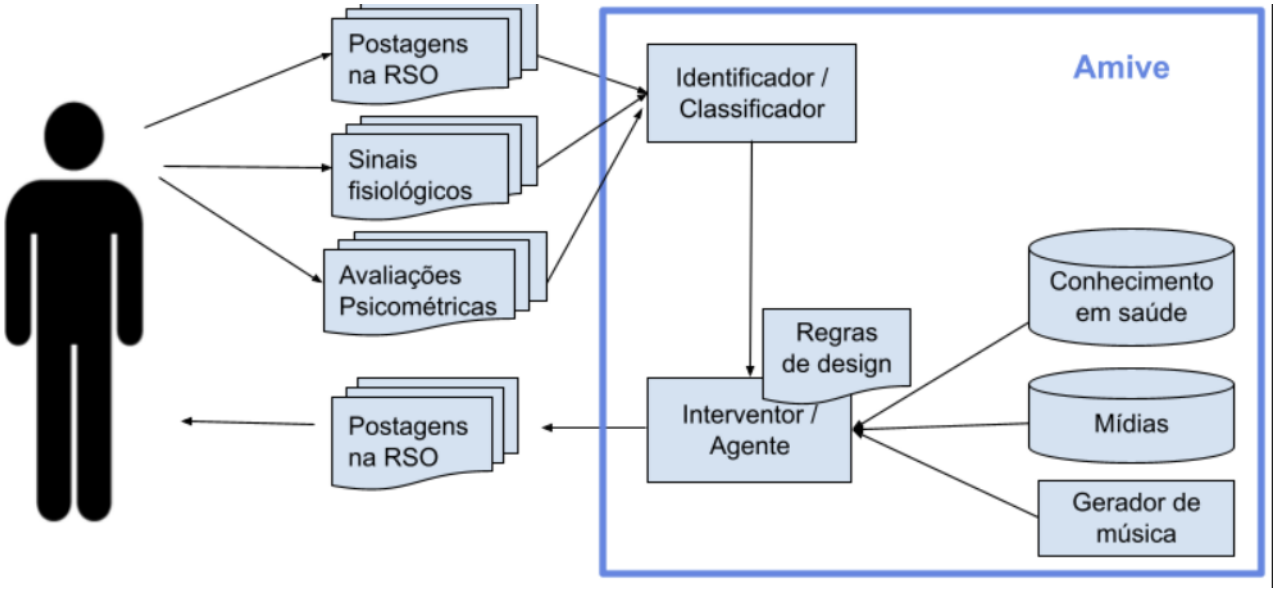

Figura 7 – Diagrama alto-nível do projeto Amive. Fonte: pesquisadores do projeto.

3 Fundamentos da pesquisa

O presente trabalho junta aspectos da arquitetura da aplicação, da coleta e da segurança dos acessos de modo a identificar as principais áreas de atuação da segurança. Para isso, é preciso investigar a arquitetura baseada em microsserviços, o gerenciamento de identidades e o controle de acesso, abordados nas próximas subseções.

3.1 Gestão da Identidade e do Acesso

A IAM é o conjunto de medidas e configurações adotadas para gerenciamento e manutenção do acesso aos recursos e aos dados de um sistema. Trata-se de peça fundamental para garantir princípios da segurança como confidencialidade, integridade e não-repudiação.

A infraestrutura de um sistema é formada por componentes, como redes de comunicação, bancos de dados e servidores web. Caso um desses recursos ou os dados utilizados por eles sejam acessados por indivíduos sem conhecimento ou com más intenções, os danos aos sistemas, às empresas e aos indivíduos podem ser catastróficos.

Esta subseção tratará das questões relacionadas ao primeiro pilar deste trabalho, a gestão da identidade e do acesso.

3.1.1 Tipos de contas de usuário

As contas de acesso de um sistema são diferenciadas de acordo com o nível de privilégios que possuem. Privilégios são as permissões de acesso aos recursos, como permissão de leitura e/ou escrita no banco de dados.

3.1.1.1 Administrador

Trata-se da conta administrativa principal, que possui todas as permissões dentro de um sistema. É de suma importância proteger essa conta, visto seu alto nível de privilégios e a impossibilidade de garantir não-repudiação.

3.1.1.2 Usuário comum

Trata-se da conta de usuário comum. Seus privilégios variam desde “apenas leitura” até os privilégios de administrador. É importante garantir que cada usuário tenha uma conta individual com as permissões estritamente necessárias (princípio do mínimo privilégio).

3.1.1.3 Serviços

Aplicações ou acesso por linha de comando podem utilizar contas de serviços para conseguir acesso aos recursos desejados. Isso é possível por meio de uma identificação de chave e uma chave secreta para adquirir acesso.

3.1.2 Grupos

Atribuir permissões individualmente a cada conta de usuário dentro de um sistema pode ser factível até certo ponto. Em grande escala, torna-se cada vez mais complexo gerenciar as permissões para as contas de forma individual e manual. Por esta razão, é possível agrupar as contas e atribuir as mesmas permissões a estes grupos.

No contexto do projeto Amive, por exemplo, pode-se destacar dois grupos triviais, como usuário comum e profissional da saúde. Contas de usuário que fazem parte do mesmo grupo possuem o mesmo conjunto de permissões, que devem seguir o Princípio do Mínimo Privilégio.

É possível - e essencial - criar grupos mais granulares de forma a garantir que cada usuário tenha apenas o menor conjunto de permissões necessárias para realizar suas funções dentro do sistema. Dessa forma, o sistema torna-se mais robusto contra escalação de privilégios.

3.1.3 Papéis

Enquanto grupos permitem agrupar usuários, papéis, por sua vez, agrupam permissões. É possível atribuir um conjunto de permissões, ou um papel, a uma conta isoladamente ou mesmo a um grupo de contas.

Papéis diminuem a complexidade de atribuir permissões a contas de usuários individuais e a grupos. Em larga escala, é essencial utilizar papéis, com diversos níveis de permissão e granulares o suficiente de modo a atender ao Princípio do Mínimo Privilégio, para atribuição de permissões.

3.1.4 Manutenção do ciclo de vida das contas de usuário

Além de estabelecer as permissões corretas para cada usuário de acordo com o Princípio do Mínimo Privilégio, é de suma importância realizar a manutenção do ciclo de vida do usuário.

Isso implica, por exemplo, remover usuários que estão inativos há muito tempo ou que não terão mais acesso aos recursos do sistema. Essa medida auxilia na prevenção de vazamento de credenciais, sobretudo por pessoas que não estão mais afiliadas ao sistema.

Além disso, é fundamental garantir que o processo de criação de novos usuários seja administrado. É uma prática comum, por exemplo, criação de vários usuários logo após a *release* do sistema, visto que as contas de acesso estão sendo criadas pela primeira vez. Esse será o caso do projeto Amive, que cadastrará 300 usuários para coleta de informações sensíveis para treino do modelo.

No entanto, se uma conta de usuário for criada meses após o cadastro inicial de usuários, é importante entender se aquela conta é válida ou se se trata de um possível ciber-criminoso que ganhou acesso à permissão de criação de contas e está deixando uma porta de acesso não documentada (*backdoor*, em inglês).

3.1.5 Credenciais Temporárias

Credencias temporárias funcionam de forma similar a chaves de acesso, mas possuem validade de curto prazo e são geradas dinamicamente. Dessa forma, não é preciso embarcar credenciais em aplicações nem estabelecer um processo de rotação, visto que sua criação é feita de forma dinâmica.

Além disso, caso a credencial seja roubada, o agente malicioso terá um prazo curto para comprometer o sistema. Por fim, credenciais temporárias possibilitam Federação de Identidade, como será explicado na subseção 3.1.6.

3.1.6 Federação de Identidade

Federação de Identidade é um sistema de autenticação e autorização de usuários baseado na confiança entre duas partes, de modo a possibilitar que um único mecanismo autentique diversos recursos de uma organização, por exemplo.

Nesse sistema, existem dois papéis envolvidos: o Provedor de Identidade (IdP, da sigla em inglês) e o Provedor de Serviços (SP, da sigla em inglês). O IdP é a autoridade da fonte de identidades, ou seja, é responsável pela autenticação do usuário, enquanto o SP, papel normalmente assumido por um serviço ou uma aplicação, controla o acesso aos recursos.

O SP confia no IdP para autenticar os usuários, de acordo com as informações oferecidas sobre eles. Após a autenticação, o IdP envia uma mensagem para o SP com os dados necessários para que o SP estabeleça uma sessão com o usuário e determine qual o nível de acesso devido.

3.1.6.1 Auth0

Auth0[1] é uma solução para adicionar serviços de autenticação e autorização em aplicações. Dois exemplos de uso do Auth0 são: adicionar serviço de autenticação e autorização em uma aplicação, tanto via nome de usuário e senha quanto através de contas em redes sociais, como o Facebook, e criar uma camada de segurança em uma API utilizando OAuth 2.0, que será discutido na subseção 3.1.6.2.

O Auth0 estabelece conexões com uma fonte de usuários, que pode incluir IdPs externos, como o Facebook. Ele também adiciona uma camada de abstração ao se interpor entre a aplicação e sua fonte de usuários, isolando a aplicação de mudanças na implementação de cada fonte.

3.1.6.1.1 Autenticação com o Facebook

Além disso, o Auth0 permite que o Facebook atue como um provedor de identidade social (IdP social) para aplicações nativas, sem a necessidade de redirecionamento via navegador web. O processo (AUTH0..., a) funciona como descrito a seguir:

- **Passo 1:** A aplicação autentica um usuário através do Kit de Desenvolvimento de Software (SDK, da sigla em inglês) do Facebook e adquire um Token de acesso.
- **Passo 2:** A aplicação, então, usa o Token de acesso para fazer uma requisição de Token de Acesso às informações da sessão do Facebook (*Facebook Session Info Acess Token*, em inglês).
- **Passo 3:** A aplicação usa o SDK do Facebook para recuperar o perfil do usuário.
- **Passo 4:** Por fim, a aplicação pode usar o token de acesso às informações de sessão do Facebook para autenticar com Auth0.

Existem diversos protocolos que especificam as diversas etapas de projetar um sistema de autenticação e autorização, desde o gerenciamento da identidade até a decisão de quais usuários podem acessar a aplicação e os dados. Dentre os padrões da indústria utilizados pelo Auth0 (AUTH0..., b), o presente trabalho abordará o *Open Authorization 2 (OAuth 2.0)*[2] e o *OpenID Connect (OIDC)*[3], descritos nas subseções 3.1.6.2 e 3.1.6.3, respectivamente.

[1] <https://auth0.com>
[2] <https://oauth.net/2/>
[3] <https://openid.net/>

3.1.6.2 OAuth 2.0

O framework de autorização OAuth 2.0 é um protocolo que permite ao usuário fornecer acesso a seus recursos protegidos a sites ou aplicações de terceiros, sem a necessidade de expor suas credenciais.

3.1.6.3 OpenID

Trata-se de uma camada de identidade acima do OAuth 2.0 e possibilita verificação simples da identidade do usuário, assim como a habilidade de pegar informações básicas sobre o perfil de um usuário através do IdP.

3.1.7 Gerenciamento de Segredos

Chaves de acesso a APIs, senhas de usuários, tokens de autenticação, certificados digitais e chaves privadas de encriptação são exemplos de segredos manipulados por aplicações rotineiramente. Visto que suas responsabilidades são associadas à autenticação e autorização, é crucial proteger estes segredos de agentes mal intencionados.

A empresa MITRE[4] é responsável por manter uma lista de tipos de vulnerabilidades, conhecida como *Common Weakness Enumeration* (CWE)[5], na qual estão presentes as vulnerabilidades mais comuns encontradas em software e em hardware. A CWE-256 é *"Plaintext Storage of a Password"*, ou seja, armazenamento de senha em texto não cifrado, que esteve nas top 25 vulnerabilidades mais perigosas de software do ano de 2021[6].

Armazenar segredos em texto não-cifrado, além do problema trivial de permitir a leitura e uso de segredos por um agente humano mal intencionado caso o sistema esteja comprometido, possibilita vazamento público de segredos na eventualidade de serem enviados para repositórios onlines que permitem acesso público.

No artigo intitulado *"How Bad Can It Git"*, os pesquisadores envolvidos utilizaram duas abordagens complementares para detecção de segredos armazenados em repositórios no GitHub[7] e descobriram centenas de milhares de chaves de criptografia e de API vazadas, em uma taxa de milhares por dia (MELI; MCNIECE; REAVES, 2019). Isso evidencia dois pontos: armazenar segredos em repositórios públicos é um descuido rotineiro, assim como armazenar segredos não cifrados embarcados na aplicação.

Há três esforços que podem ser realizados no sentido de mitigar esses pontos. O primeiro deles é procurar por segredos no código da aplicação antes de enviá-lo a um repositório. Existem diversas técnicas que podem ser utilizadas, como busca por padrões

[4] <https://www.mitre.org/>
[5] <https://cwe.mitre.org/>
[6] <https://cwe.mitre.org/data/definitions/256.html>
[7] <https://github.com/>

utilizando expressões regulares (*regular expressions*, em inglês, ou *regex*, como é mais conhecido) ou através de ferramentas como a *git-secrets*[8], que automatizam essa busca. Caso o segredo já tenha sido enviado ao repositório remoto, o controle de versionamento o manterá no histórico e, por isso, será preciso modificar o histórico manualmente para apagar a existência dos segredos ou por meio de ferramentas que automatizam o processo, como a *git-filter-branch*[9] e a BFG[10].

O segundo esforço é utilizar um gerenciador de segredos em vez de embarcá-los no código da aplicação. Os gerenciadores de segredos, como o HashiCorp Vault[11], armazenam e realizam rotação de tokens, senhas, certificados, entre outros, o que é uma grande vantagem visto que segredos tornam-se paulatinamente menos confiáveis com o passar do tempo, como será mencionado na seção 3.1.8.

Por último, o terceiro esforço seria adotar encriptação de segredos, uma boa prática de segurança que deve ser adotada sempre que possível. A seção 3.2 tratará em maiores detalhes sobre encriptação de dados, tanto em repouso quanto em trânsito.

3.1.8 Política de senhas

O uso de senhas definidas pelo usuário para acesso a recursos é praticamente ubíquo em sistemas com IAM e, considerando que o vazamento ou roubo delas pode resultar em falsificação, adulteração e escalação de privilégios, é evidente a necessidade de adotar uma política forte de senhas.

Nenhuma senha é completamente segura, visto que um ciber-criminoso pode tentar todas as combinações possíveis, desde que possua os recursos computacionais e o tempo necessários. Assim, é de suma importância instaurar uma política forte de senhas de usuários cadastradas no sistema da aplicação, de modo a dificultar ataques que busquem quebrar a senha através de todas as combinações disponíveis, conhecidos como ataques de força bruta.

Uma política forte de senhas seria, por exemplo, exigir uma senha com mínimo de oito caracteres para contas com Autenticação Multifator (MFA, da sigla em inglês) habilitada e senha com mínimo de 14 caracteres para contas sem MFA habilitada, além de forçar uso de ao menos um caractere minúsculo, um caractere maiúsculo, um número e um caractere não-alfanumérico, como sugerido pelo guia de políticas de senhas do *Center for Internet Security* (CIS) (BENCHMARKS, 2021).

Essas precauções são fundamentais para evitar quebra de senhas por ataques de força bruta. A Figura 8 ilustra o tempo necessário para quebrar senhas, de acordo

[8] <https://github.com/awslabs/git-secrets>
[9] <https://git-scm.com/docs/git-filter-branch>
[10] <https://rtyley.github.io/bfg-repo-cleaner/>
[11] <https://www.vaultproject.io/>

com o tamanho delas e com os caracteres incluídos. Ressalta-se que a política de senhas mencionada anteriormente aumenta drasticamente o tempo necessário para que a senha seja descoberta, utilizando-se os recursos computacionais disponíveis atualmente.

Password Length	Numerical 0-9	Upper & Lower case a-Z	Numerical Upper & Lower case 0-9 a-Z	Numerical Upper & Lower case Special characters 0-9 a-Z %$
1	instantly	instantly	instantly	instantly
2	instantly	instantly	instantly	instantly
3	instantly	instantly	instantly	instantly
4	instantly	instantly	instantly	instantly
5	instantly	instantly	instantly	instantly
6	instantly	instantly	instantly	20 sec
7	instantly	2 sec	6 sec	49 min
8	instantly	1 min	6 min	5 days
9	instantly	1 hr	6 hr	2 years
10	instantly	3 days	15 days	330 years
11	instantly	138 days	3 years	50k years
12	2 sec	20 years	162 years	8m years
13	16 sec	1k years	10k years	1bn years
14	3 min	53k years	622k years	176bn years
15	26 min	3m years	39m years	27tn years
16	4 hr	143m years	2bn years	4qdn years
17	2 days	7bn years	148bn years	619qdn years
18	18 days	388bn years	9tn years	94qtn years
19	183 days	20tn years	570tn years	14sxn years
20	5 years	1qdn years	35qdn years	2sptn years

Figura 8 – Tempo necessário para quebrar uma senha de acordo com tamanho e caracteres. Fonte (THE. . . ,).

Como mencionado, nenhuma senha é completamente segura, fornecidos os recursos e o tempo suficientes. As senhas tornam-se, com o passar do tempo, cada vez menos confiáveis, porque os usuários tendem a reaproveitá-las em diversas contas, o que, invariavelmente, aumenta as chances de vazamento (AWAD et al., 2016).

Além disso, existem estratégias para reduzir o tempo necessário para um ataque de força bruta, como utilizar dicionários de palavras mais propensas a serem utilizadas como senhas pelos usuários. Dessa forma, em vez de tentar todas as combinações de caracteres possíveis, o ciber-criminoso pode tentar as palavras presentes em uma lista de propensão. Esses dicionários de senhas, conhecidos como *wordlists* em inglês, podem conter senhas comuns utilizadas por pessoas em geral ou até mesmo senhas comuns usadas por determinados grupos de pessoas.

Um dos dicionários de senhas mais famosos é o *rockyou.txt*[12], que contém milhões de senhas únicas. Ele foi formado após o hacking da empresa RockYou em 2009[13], quando as senhas dos usuários da empresa foram vazadas em texto não-cifrado e disponibilizadas

[12] <https://github.com/brannondorsey/naive-hashcat/releases/download/data>
[13] <https://cybernews.com/security/rockyou2021-alltime-largest-password-compilation-leaked/>

publicamente. Outros dicionários de senhas focam, por exemplo, em palavras do contexto brasileiro[14] e há, ainda, dicionários customizados que são criados a partir de sites de empresas com ferramentas como a *Custom Word List generator* (CeWL)[15].

Por esses motivos, políticas fortes de senhas também devem impedir reuso de senhas antigas, obrigar rotação de senhas e proibir palavras comuns, como aquelas disponíveis em dicionários de senhas, de serem utilizadas.

3.1.9 Autenticação Multifator

Autenticação Multifator (MFA, da sigla em inglês), adiciona proteção para as contas de usuário e de administrador. O propósito do MFA é utilizar ao menos dois fatores para acessar a conta, em vez de apenas um.

São três fatores, descritos a seguir:

- **Algo que se sabe**, como nomes de usuários, senhas e PINs.
- **Algo que se tem**, como tokens, gerados por software ou por hardware.
- **Algo que se é**, como impressão digital e reconhecimento facial.

No caso de autenticação de dois fatores, é necessário utilizar dois fatores para completar a autenticação. Um exemplo comum é exigir do usuário o nome de usuário, senha e token gerado por software, adquirido pelo usuário através de uma tecnologia de multifator como Authy[16] e Google Authenticator[17].

Dessa forma, caso um agente malicioso consiga roubar o nome de usuário e senha de uma conta, a autenticação não será bem sucedida na ausência do token gerado pela tecnologia de multifator.

3.1.10 Ataques

Os ataques mais comuns que dizem respeito a IAM envolvem adivinhação de senhas de usuários e roubo de credenciais.

3.1.10.0.1 Adivinhação de senhas

Como mencionado previamente, é possível tentar adivinhar senhas de usuários através de ataques de força bruta, seja por meio de tentativas de todas as combinações possíveis ou de senhas prováveis em uma lista (*wordlist*).

[14] <https://github.com/BRDumps/wordlists>
[15] <https://www.kali.org/tools/cewl/>
[16] <https://authy.com/>
[17] <https://en.wikipedia.org/wiki/Google_Authenticator>

No entanto, as aplicações, ao menos as mais robustas, estabelecem controle de tentativas de autenticação e, dado um número de tentativas inválidas, bloqueiam futuras requisições. Dessa forma, além de não ser possível tentar, por força bruta, diversas senhas, os profissionais responsáveis pelo monitoramento podem perceber o comportamento suspeito e ficar alertas sobre eventuais invasões ao sistema.

Assim, ciber-criminosos passaram a tentar outro tipo de ataque conhecido como *password spray*. Nele, o agente malicioso escolhe uma senha provável e tenta logar usando uma lista de nomes de usuários vazados. Apenas uma tentativa de autenticação será realizada por conta de usuário e, visto que a senha é propensa a ter sido escolhida por algum usuário, as chances de sucesso aumentam.

3.1.10.0.2 Roubo de credenciais

Além das tentativas de adivinhação de senhas, ciber-criminosos podem tentar roubar segredos, normalmente nomes de usuário e senhas, e a maneira de consegui-los é através das pessoas.

No contexto da segurança da informação, a engenharia social trata da manipulação de pessoas para *hackear* os sistemas. São usadas técnicas de persuasão e conhecimentos de psicologia para enganar as vítimas e fazê-las fornecer credenciais ou mesmo realizar ações que não fariam normalmente.

Um dos exemplos mais clássico são ataques de *phishing* (SHANKAR; SHETTY; K, 2019). O cerne deles é um e-mail, muitas vezes de caráter genérico e enviado em massa, que busca fazer a vítima baixar *malware* para sua máquina ou vazar credenciais, por exemplo, através de link que parece confiável ou que intercepta uma página real (ataque conhecido como *man-in-the-middle*).

Um tipo mais específico de *phishing* é conhecido como *spear phishing*. Enquanto a palavra *phishing* faz referência a *fishing*, no sentido de "pescar vítimas", visto que o ataque é enviado em massa e colhe apenas algumas vítimas por vez, *spear phishing* faz referência a "pesca com lança". Isso significa que os ciber-criminosos miram especificamente em uma vítima antes de enviar o e-mail, cujo conteúdo foi previamente pensado para fazer aquele usuário em especial cair no ataque.

3.1.11 Melhores Práticas

Abaixo, estão listadas medidas que foram discutidas ao longo desta seção e que podem ser tomadas como melhores práticas de segurança no que diz respeito à gestão da identidade e do acesso.

1. Criar contas individuais para cada usuário.

2. Limitar o uso da conta administrativa.
3. Usar Grupos para designar permissões aos usuários.
4. Usar Papéis para criar conjuntos de permissões.
5. Sempre aderir ao Princípio do Privilégio Mínimo ao atribuir permissões a usuários.
6. Utilizar uma política de senhas forte para os usuários, utilizando um *benchmark* de referência na indústria.
7. Não permitir senhas comuns.
8. Não permitir reuso de senhas antigas.
9. Habilitar Autenticação Multifator.
10. Rotacionar credenciais regularmente.
11. Remover credenciais e usuários desnecessários.
12. Monitorar atividade das contas de usuários.
13. Limitar número de tentativas para acesso à conta.
14. Não enviar segredos para repositórios públicos.
15. Remover segredos armazenados em repositórios públicos e limpar o histórico de versionamento.
16. Encriptar segredos.
17. Se possível, utilizar um gerenciador de segredos para realizar rotação e armazenamento de segredos.
18. Treinar usuários para reconhecer tentativas de falsificações de e-mails e notificações no geral.

3.2 Proteção dos Dados

Esta seção abordará conceitos relacionados ao segundo pilar do presente trabalho, ou seja, à proteção de dados do ponto de vista técnico e de *compliance*.

3.2.1 Tipos de dados

A LGPD define quatro tipos de dados (LGPD. . . , 2021):

- **Dados pessoais:** são dados que possibilitam a identificação, direta ou indireta, da pessoa natural, como nome e sobrenome, data e local de nascimento, RG e CPF.
- **Dados sensíveis:** são dados pessoais que exigem maior atenção no tratamento. Alguns exemplos de dados sensíveis: dados pessoais relacionados a crianças e adolescentes, dados que revelam origem racial ou étnica, questões genéticas, biométricas e sobre a saúde de uma pessoa.
- **Dados públicos:** inclui tanto dados que não são pessoais e/ou sensíveis e dados que os são, mas foram tornados públicos com consentimento do indivíduo.
- **Dados anonimizados:** são dados que passaram pelo processo de anonimização, ou seja, que tiveram dados removidos ou informações modificadas de forma a impossibilitar a identificação de uma pessoa. Nesse caso, a LGPD não se aplicará ao dado.

3.2.2 Dados em Repouso

Dados em repouso são aqueles armazenados em um computador, seja em discos rígidos (mais conhecidos como HDs, do inglês *Hard Drives*), SSDs (do inglês *Solid State Disk*), RAIDs (do inglês *Redudant Array of Independent Disks*), em servidores de banco de dados ou até mesmo em armazenamento em nuvem.

Isso significa que nenhum serviço, aplicação, usuário ou partes terceiras estão utilizando o dado no momento em que está armazenado e, no entanto, ainda assim trata-se de um alvo de ciber-criminoso porque, geralmente, os dados são armazenados seguindo uma estrutura lógica e organizada, com nomes de diretórios e arquivos significativos. Essa característica pode tornar o trabalho de um agente malicioso mais fácil.

Todo tipo de dados costumam ser armazenado, como credenciais de usuários, tokens de acesso, dados pessoais e sensíveis de clientes, entre outros. Assim, urge tomar precauções para protegê-lo e a mais notável delas é a encriptação de dados em repouso.

Existem alguns tipos diferentes de encriptação de dados em repouso, como descritos a seguir:

- **Encriptação a nível da aplicação:** a aplicação que trata os dados realiza a encriptação deles.
- **Encriptação a nível do banco de dados:** normalmente é o time responsável pela segurança dos banco de dados que encripta o banco de dados.

- **Encriptação do sistema de arquivos:** uso de ferramentas ou do próprio sistema operacional para encriptar o sistema de arquivos. O acesso aos dados, ou mesmo a montagem do sistema de arquivos, só é realizado quando a senha ou a chave é fornecida.
- **Encriptação de disco completo:** os dados da unidade de armazenamento ficam encriptados quando o dispositivo está desligado e a única forma de ligá-lo é fornecer a senha ou chave.

Existem diversas vantagens para a prática de encriptação de dados em repouso, como prevenir acesso não-autorizado, mencionado previamente. Além disso, ela limita o impacto de vazamento de conteúdo sensível, já que os dados são armazenados em um formato não legível para humanos. Por último, reduz o risco de dispositivos perdidos ou roubados.

3.2.3 Dados em Trânsito

Dados em trânsito são pacotes que transitam por meio de um canal de comunicação, como a internet. As aplicações baseadas em microsserviços, como descrito ao longo deste trabalho, são compostas por diversas entidades - os serviços - que comunicam-se entre si pela requisição e envio de informações. Nesse contexto, é essencial garantir que os dados em trânsito estejam devidamente protegidos contra dois objetivos de um agente malicioso performando um ataque *man-in-the-middle*.

No primeiro, a intenção é ler ou roubar a informação contida nos dados, enquanto, no segundo, é a adulteração do conteúdo dos dados em trânsito. Em ambos os casos, o ciber-criminoso se insere no meio do canal e faz com que as partes comunicantes acreditem que estão em uma comunicação fidedigna.

Existem formas de mitigar essa ameaça, como o uso do protocolo de segurança *Transport Layer Security* (TLS). O TLS é um protocolo criptográfico usado para prover uma comunicação segura em uma rede não-confiável, como quando é usado em conjunto com o protocolo *Hypertext Transfer Protocol* (HTTP) na internet, resultando no protocolo *Hypertext Transfer Protocol Secure* (HTTPS).

Além disso, é possível utilizar certificados digitais para impedir a utilização de *proxies* entre o canal TLS e o sistema.

3.2.4 Disponibilidade

A disponibilidade de dados também é uma questão importante a ser levada em consideração ao arquitetar uma solução. Urge garantir que a aplicação esteja acessível para os usuários e, para isso, faz-se necessário planejar a quantidade de requisições aos recursos

ao longo do tempo. Isso ocorre pois em determinadas épocas do ano pode haver maior número de acessos à aplicação. Um exemplo típico é o acesso à plataformas de *e-commerce* durante o período de festas no final do ano e, se os sistemas não estiverem preparados para lidar com a demanda recrudescente, as perdas financeiras e de reputação podem ser catastróficas.

Para casos de aumento de demanda, a arquitetura baseada em microsserviços se mostra bastante versátil, uma vez que é possível escalar os microsserviços necessários separadamente. Sobretudo para sistemas que usam soluções em nuvem, híbridas ou não, é possível escalar tanto verticalmente - aumento de recursos, como memória RAM ou CPU do hardware responsável pelas funcionalidades - quanto horizontalmente, por meio de adição de máquinas.

Outra questão diretamente relacionada à disponibilidade dos serviços prestados por uma aplicação são os ataques do tipo *Denial of Service* (DoS) ou *Distributed Denial of Service* (DDoS). Neles, o objetivo é cortar acesso dos usuários ao sistema ou aos serviços oferecidos por ele.

3.2.5 Compliance

Existem diversas políticas de *compliance* que devem ser seguidas, dependendo do escopo da aplicação, do tipo de dados que ela manipula, do país ou estado em que atua, entre outros fatores.

No Brasil, talvez a mais conhecida das políticas que devem ser seguidas por aplicações que manipulam dados é a LGPD, abordada previamente. No entanto, diversas outras políticas e leis podem ser aplicadas e é de suma importância que o desenvolvimento da aplicação seja alinhado a elas.

3.2.6 Melhores Práticas

A seguir, foram listadas as medidas discutidas ao longo desta seção e que podem ser tomadas como melhores práticas de segurança no que diz respeito à proteção de dados.

1. Encriptar dados em repouso.

2. Encriptar dados em trânsito.

3. Estabelecer meios confiáveis de trânsito de dados através de certificados digitais.

4. Seguir leis e políticas de *compliance*.

3.3 Segurança da Arquitetura baseada em Microsserviços

Nesta seção, serão discutidos aspectos do terceiro pilar deste trabalho - a segurança da arquitetura baseada em microsserviços.

3.3.1 Ciclo de vida do desenvolvimento de software seguro

A construção dos sistemas normalmente é realizada por meio de etapas que, a depender da metodologia adotada, podem ser diferentes. Por exemplo, no caso da metodologia ágil, o projeto é dividido em ciclos, ao fim dos quais os clientes podem verificar os resultados.

Existem frameworks especializados em aumentar a segurança da aplicação em todas as fases da construção da aplicação. Dois deles são: NIST Special Publication 800-160 volume 2, que foca em resiliência cibernética dos sistemas (ROSS et al., 2021), e OWASP *Software Assurance Maturity Model* (SAMM)[18], que provê uma técnica mensurável para análise e melhoria do ciclo de vida do desenvolvimento seguro de aplicações. A Figura 9 ilustra o modelo SAMM.

Figura 9 – Modelo OWASP SAMM. Fonte:(OWASP. . . , a).

São cinco funções de negócio: governança, projeto, implementação, verificação e operações. Para cada uma delas, estão associadas 3 melhores práticas de segurança, cada uma com 3 níveis de maturidade.

- **Governança:**
  - **Estratégia e métricas:** visa criar um plano efetivo para realização dos objetivos relacionados à segurança da aplicação.

[18] <https://owasp.org/www-project-samm/>

  - **Políticas e *compliance*:** focam no entendimento e na implementação de requisitos legais e regulamentares, ao mesmo tempo que buscam alinhar os padrões internos de segurança com os objetivos de negócio da organização.
  - **Educação e orientação:** busca orientar e educar os indivíduos envolvidos nas etapas do ciclo de vida do desenvolvimento de software em boas práticas de segurança.

- **Projeto:**
  - **Avaliação de ameaças:** objetiva identificar e entender riscos a nível de projeto.
  - **Requisitos de segurança:** foca nos requisitos de segurança relacionados ao software seguro.
  - **Arquitetura segura:** busca identificar pontos de segurança relacionados aos componentes e à tecnologia utilizados durante o projeto arquitetural do software.

- **Implementação:**
  - **Construção segura:** enfatiza a importância de construir software de maneira padronizada, utilizando boas práticas como revisão de código, e utilizando componentes seguros.
  - **Implantação segura:** assegurar que a segurança e a integridade das aplicações desenvolvidas não foram comprometidas durante a fase de implantação.
  - **Gestão de defeitos:** visa coletar, armazenar e analisar defeitos de segurança do software.

- **Verificação:**
  - **Avaliação da arquitetura:** certificar-se que a arquitetura da infraestrutura e a aplicação cumprem todos os requisitos de segurança e de *compliance*.
  - **Testes orientados por requisitos:** garantir que os controles de segurança adotados funcionam como esperado.
  - **Testes de segurança:** utilizar testes de segurança automatizados e testes em profundidade manuais baseados no bom conhecimento das funcionalidades da aplicação.

- **Operações:**
  - **Gestão de incidentes:** busca lidar com incidentes de segurança, como DoS e escalação de privilégios por parte de um usuário.

- **Gestão do ambiente:** foca em manter o ambiente seguro e limpo através da operação segura da pilha de tecnologias.
- **Gestão operacional:** foca em atividades para garantir que a segurança está sendo mantida através de funções de apoio operacional.

3.3.2 Contêineres

Como foi mencionado na seção 2.1.2, contêineres permitem empacotar aplicações configuradas para executar em qualquer tecnologia, desde que apoiada por contêineres. Isso é possível através do uso de imagens de contêiner, que podem ser armazenadas tanto localmente quanto em repositórios.

As imagens podem ser fornecidas por terceiros e armazenadas em repositórios de imagens de contêineres, como o *Docker Hub*. Normalmente, essas imagens são construídas sobre a imagem de um sistema operacional e modificadas por meio de instruções de instalação e de configuração em um arquivo como o *Dockerfile*.

O uso de imagens públicas armazenadas em repositórios pode representar um risco de segurança. Isso ocorre porque, além da possibilidade de haver *malwares* ou *backdoors* nessas imagens, elas podem conter vulnerabilidades críticas. Uma pesquisa de 2020 realizada pela Prevasio[19](PREVASIO. . . ,), empresa que oferece uma ferramenta de gerenciamento da postura de segurança na nuvem (CSPM, da sigla em inglês), descobriu que 51% de cerca de 4 milhões de imagens públicas armazenadas no *Docker Hub* continham vulnerabilidades críticas.

Além disso, 6 mil contêineres continham mineradores de criptomoedas e *backdoors* o que, embora não signifique necessariamente que as imagens estão comprometidas, pode indicar que atividades maliciosas estão em curso. Por último, a análise de contêineres maliciosos mostrou imagens que, em tempo de execução, poderiam fazer download de fontes de mineração de criptomoedas através de *scripts*.

Assim, imagens de contêineres devem ser constantemente avaliadas de acordo com as melhores práticas de segurança. Isso inclui garantir que o software instalado com a imagem está atualizado e que terá acesso a atualizações futuras, além de executar a imagem de contêiner como um usuário com os mínimos privilégios necessários.

Uma ferramenta interessante é a *Docker Bench for Security*[20]. Trata-se de um *script* que busca por melhores práticas de segurança em torno da implantação de contêineres *Docker* em produção. Outro ponto a ser levado em consideração é que muitos dados de *log* são perdidos quando o contêiner para de executar. Para isso, podem ser usados *plugins*, oferecidos pelo *Docker* por exemplo, que realizam a função de coletar *logs*.

[19] <https://www.prevasio.io/>
[20] <https://hub.docker.com/r/docker/docker-bench-security>

3.3.3 Segurança das camadas 3 e 4

O modelo *Open Systems Interconnect* (OSI) é uma arquitetura de rede utilizada como referência para sistemas abertos à conexão com outros sistemas. Ele é composto por 7 camadas, ilustradas e descritas na Figura 10.

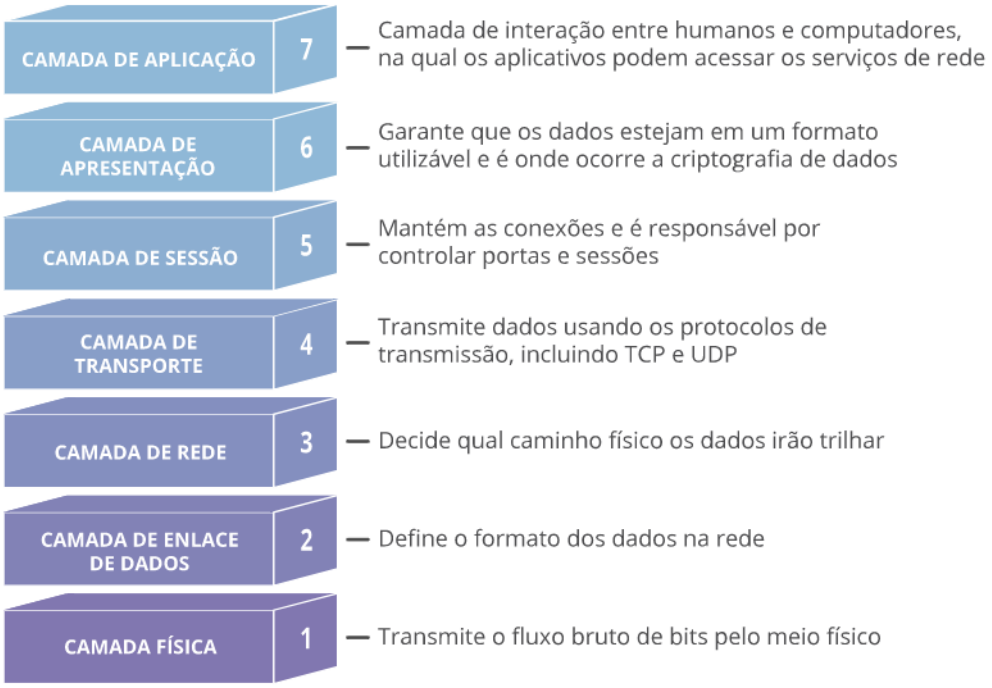

Figura 10 – Modelo OSI. Fonte:(CLOUDFLARE. . . ,).

A segurança das camadas 3 e 4 refere-se, portanto, às camadas de rede e de transporte. É através delas que são definidos quais endereços de IP e quais portas e protocolos podem se comunicar com os serviços, respectivamente.

3.3.3.1 Exposição de serviços

É de suma importância estabelecer os filtros corretos para ambas as camadas, visto que serviços sensíveis expostos na internet podem resultar em comprometimentos sérios às aplicações. Shodan[21] é uma ferramenta de busca que permite pesquisar por diversos tipos de servidores conectados à internet. Ele coleta dados sobretudo de servidores web (HTTP/HTTPS, que usam as portas 80 e 443, por exemplo), mas também recolhe informações sobre *File Transfer Protocol* (FTP), *Secure Shell* (SSH) e *Simple Mail Transfer Protocol* (SMTP), por exemplo.

Ferramentas como Shodan também são utilizadas por agentes maliciosos para identificar possíveis alvos, por isso é muito importante verificar quais serviços estão desnecessariamente - e perigosamente - expostos na internet. O exemplo mais clássico é de

[21] <https://www.shodan.io/>

exposição do banco de dados. Um banco de dados não deve jamais ser acessível diretamente pela internet e alguma camada deve ser implementada para buscar seus recursos, como uma API.

Uma rápida busca no Shodan com o filtro "port:3306", que é a porta comumente utilizada para acesso aos servidores de bancos de dados MySQL, retornou mais de 4 milhões de resultados. A Figura 11 ilustra os resultados da busca.

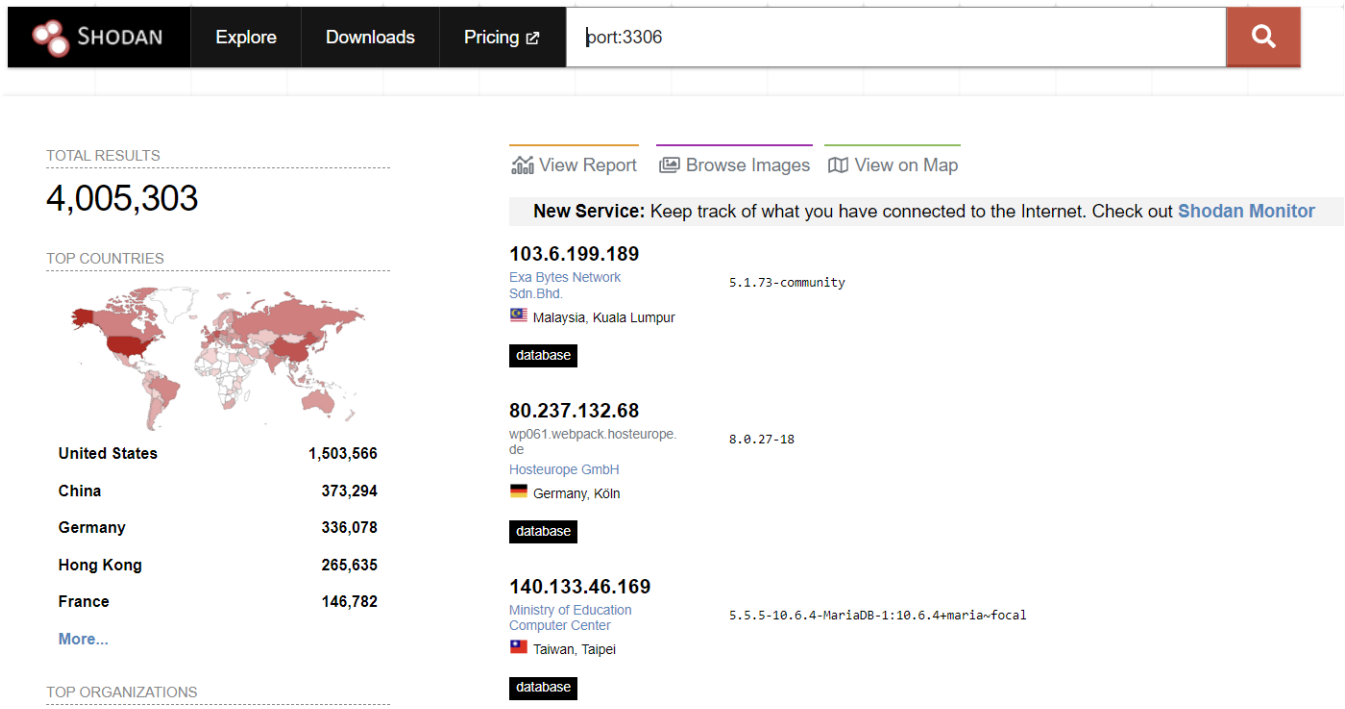

Figura 11 – Resultados da busca por "port:3306" na ferramente da pesquisa Shodan. Fonte: busca no Shodan.

A exposição de serviços sensíveis na internet é crítica sobretudo porque o agente malicioso passa a estar a um roubo ou adivinhação de credenciais para ter acesso a todo o serviço - no caso de banco de dados, a todo o banco e aos dados, sensíveis ou não, que ele armazena.

No caso de uma arquitetura baseada em microsserviços, apenas os serviços que precisam se comunicar com entidades externas pela rede devem estar expostos na internet. Os outros microsserviços devem permitir apenas comunicação interna com outros microsserviços que necessitam dos seus recursos, normalmente também com acesso à mesma rede virtual privada (VPN, da sigla em inglês).

3.3.3.2 Segmentação de redes

Uma técnica de segurança utilizada em redes é a segmentação, na qual uma VPN, por exemplo, é dividida em partes e apenas os serviços pertencentes às mesmas partes

podem se comunicar livremente. Para comunicação entre as partes, são estabelecidas regras mais rígidas e, normalmente apenas alguns serviços são permitidos.

A segmentação de redes reduz superfície de ataque de sistemas comprometidos, uma vez que a parte ameaçada está logicamente isolada das outras. Além disso, pode ser usada para criar uma camada extra de isolamento para sistemas sensíveis, já que acrescenta um ponto de estrangulamento (*chokepoint*, em inglês). Por último, também melhora a performance da rede, já que separa tráfegos não relacionados em segmentos diversos da rede. O processo de segmentar uma rede em partes menores é conhecido como *subnetting* e as redes menores derivadas do processo são chamadas de *subnets*.

Subnets públicas permitem acesso vindo da internet, desde que exista um endereço de IP público. *Subnets* privadas, por outro lado, não são acessíveis pela internet, visto que não fornecem serviços a usuários e seus serviços devem ser acessados exclusivamente por serviços internos. Caso haja a necessidade de permitir tráfego no sentido estrito *subnet* privada para a internet, é possível utilizar *gateways Network Address Translation* (*gateways* NAT).

3.3.3.3 Segregação de serviços sensíveis

Para serviços muito sensíveis, como banco de dados, pode-se utilizar servidores *jumpers* ou bastiões para criar mais uma camada de isolamento. Servidores *jumper* são tratados como a porta única de entrada para um grupo de servidores dentro de uma zona de segurança e podem funcionar como uma ponte entre duas redes confiáveis, como duas *subnets*.

Bastiões, por sua vez, ficam fora da zona de segurança da rede da aplicação. O objetivo dos bastiões é de prover acesso a uma rede privada a partir de redes externas, como a própria internet. Em ambas as técnicas, contudo, o servidor que funciona como porta de entrada é tratado como um ponto de auditoria para acessar as *subnets*.

3.3.3.4 Gerenciamento do acesso remoto

O acesso remoto a serviços pode ser realizado através de dois protocolos: *Remote Desktop Protocol* (RDP) no caso do Windows e *Secure Shell* (SSH), para sistemas operacionais baseados em Linux.

Visto que eles permitem o gerenciamento remoto de sistemas, pode-se dizer que eles deixam um possível agente malicioso a um roubo ou adivinhação de credenciais de distância dos serviços. Por essa razão, eles devem ser utilizados somente se estritamente necessário.

3.3.4 Melhores Práticas

As medidas listadas abaixo foram discutidas ao longo desta seção e podem ser tomadas como melhores práticas no que concerne à segurança da arquitetura.

1. Oferecer treinamento às pessoas envolvidas no desenvolvimento e no gerenciamento do software.
2. Definir requisitos de segurança.
3. Definir métricas e relatórios de *compliance*.
4. Realizar modelagem de ameaças.
5. Definir e usar padrões de criptografia.
6. Usar ferramentas aprovadas por padrões rígidos de segurança.
7. Realizar teste de segurança de análise estática.
8. Realizar teste de segurança de análise dinâmica.
9. Realizar testes de invasão.
10. Estabelecer um processo padrão de resposta a incidentes.
11. Realizar testes de segurança em imagens de contêiner.
12. Adicionar camadas de segurança ao banco de dados ou a serviços sensíveis utilizando servidores *jumper* ou bastiões.
13. Segmentar a rede privada interna, separando os serviços que não comunicam-se entre si ou que apresentam níveis diferentes de sensibilidade.
14. Não permitir que os serviços sejam desnecessariamente acessíveis por redes externas.
15. Filtrar os serviços que devem ter acesso aos recursos de outro serviço através do uso de IPs.
16. Desativar acesso remoto a serviços caso não seja estritamente necessário.
17. Utilizar SSH apenas com chaves em vez de credenciais.

4 Proposta de arquitetura de sistema baseada em microsserviços

Com base nos conceitos estudados e apresentados ao longo dos capítulos anteriores, foi formulada uma arquitetura de sistema baseada em microsserviços que atendesse às necessidades do projeto Amive. A proposta, fundamentada sob um modelo constituído em camadas, foi ilustrada levando-se em consideração o modelo C4 para visualização de arquitetura de software.

A Figura 12 ilustra o diagrama de contexto do modelo C4 para o projeto Amive.

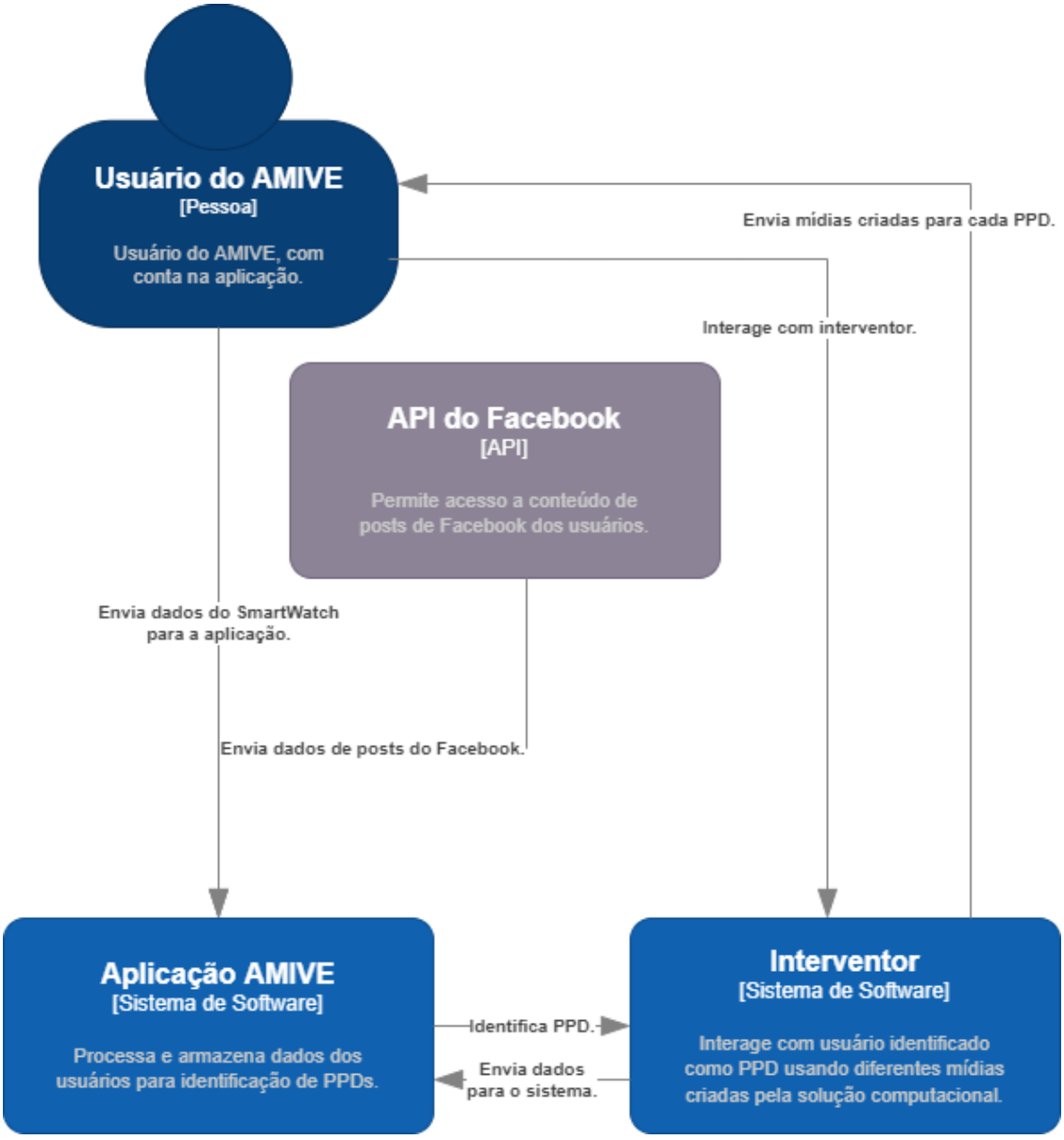

Figura 12 – Diagrama de contexto do projeto Amive. Fonte: da própria autora.

Por se tratar da camada mais abstrata do modelo C4, o diagrama de contexto busca representar a aplicação Amive como uma caixa preta que esconde os detalhes de funcionamento interno em prol de evidenciar as relações que existem entre ela e sistemas externos e usuários.

É possível perceber, pela Figura 12, que o aplicativo do usuário enviará dados fisiológicos coletados pelo *SmartWatch* para a aplicação. Além disso, a API do Facebook será utilizada para enviar conteúdo de postagens do Facebook dos usuários, que também serão processados pela aplicação. Caso o usuário seja identificado como PPD, uma das possibilidades consideradas no projeto é que a aplicação Amive enviará informações ao interventor, implementado sob o formato de ChatBot, que estabelecerá uma comunicação com o usuário através de mídias de diversos formatos, criadas especialmente para cada PPD. O usuário, por sua vez, também interagirá com o interventor.

A segunda camada do modelo C4, de contêiner, está ilustrada pela Figura 13. Nela, os detalhes do sistema de software Amive são mostrados e é possível perceber o funcionamento e a comunicação interna dos módulos funcionais.

O usuário do Amive fará download dos dados coletados pelo *SmartWatch* em um arquivo do sistema de arquivos local do celular. Em seguida, entrará no aplicativo Android do Amive e selecionará as opções de dados que deseja enviar à aplicação. O aplicativo é responsável por pegar o documento local com os dados coletados pelo *SmartWatch*, filtrar os dados de interesse e enviá-los à API via chamada de API. O conteúdo de posts do Facebook, por sua vez, será enviado diretamente para a API do Amive.

A API do Amive cumpre a função de enviar os dados ao banco NoSQL, visto que um banco de dados não deve ficar exposto para acesso direto pelos serviços. Foi escolhido um banco de dados NoSQL por causa do volume de dados que a aplicação tratará na etapa de treino dos modelos. A API também pode buscar dados armazenados no banco de dados e enviá-los para a fila de mensagens, que pode ser tanto do tipo *message broker* quanto *publish-subscribe*.

Uma medida interessante do ponto de vista da segurança seria segmentar os módulos em diferentes *subnets*, de modo que o banco de dados fique logicamente isolado em sua própria *subnet*. Isso evitaria comprometimento dos dados caso houvesse comprometimento da rede através da qual os outros serviços se comunicam. Um servidor *jumper* ou um bastião podem ser adotados para criar uma camada extra de proteção ao banco de dados.

Além disso, é importante ressaltar que os dados em repouso armazenados no banco de dados devem ser encriptados utilizando algum algoritmo de encriptação amplamente usado e aceito pela comunidade de segurança. Visto que a comunicação entre os serviços será realizada através do protocolo HTTPS, ou outros protocolos sobre o TLS, o canal de comunicação estará assegurado e os dados estarão encriptados utilizando um algoritmo

criptográfico de chave assimétrica, como o ECDSA.

A fila de mensagens é orientada a eventos e busca armazenar os dados enviados para os consumidores, de modo que eles consumam os dados aos quais estão inscritos quando for necessário. Os consumidores, nessa proposta, são o módulo de Processamento de Linguagem Natural (PLN) e o módulo de processamento de dados fisiológicos, que necessitam dos dados do usuário para realizar o treinamento e a classificação do usuário em PPD. A classificação, então, retorna à fila de mensagens, por onde o consumidor interventor pode pegá-la quando necessário. O interventor será futuramente implementado no formato de um *ChatBot* que estabelecerá comunicação com PPDs.

Por último, a fila de mensagens proposta como parte da aplicação poderá suprir a demanda extra caso haja necessidade futura de escalar a aplicação ou de adicionar novos módulos de processamento que serão consumidores dos dados de usuários.

4.1 Justificativa da arquitetura

A escolha pelo banco de dados NoSQL está diretamente relacionada à flexibilidade em relação à escalabilidade futura e à possibilidade de armazenar diferentes tipos de dados sem a necessidade de planejamento dos tipos de dados com antecedência.

A API do Amive foi adotada sobretudo pela necessidade de manter o banco de dados inacessível pelo usuário, pela aplicação e por outros serviços que precisam dos dados.

A fila de mensagens foi adotada com o propósito de tornar a aplicação mais escalável e permitir que os consumidores dos dados armazenados nela possam se inscrever e consumir os dados que são de seu interesse. Um dos consumidores, o módulo de PLN, por exemplo, pode se inscrever para os eventos nos quais há dados relacionados a conteúdos de posts de Facebook. Os dados estarão disponíveis na fila de mensagens por um tempo determinado, mas maior que o tempo previsto para o consumo dos dados por cada um dos consumidores.

Segmentar a rede da aplicação também é uma boa prática de segurança, sobretudo se se trata de proteger o banco de dados de possíveis ataques por movimentação entre serviços. Além disso, outras boas práticas de segurança são o uso do protocolo HTTPS para comunicação em rede e encriptação de dados em repouso.

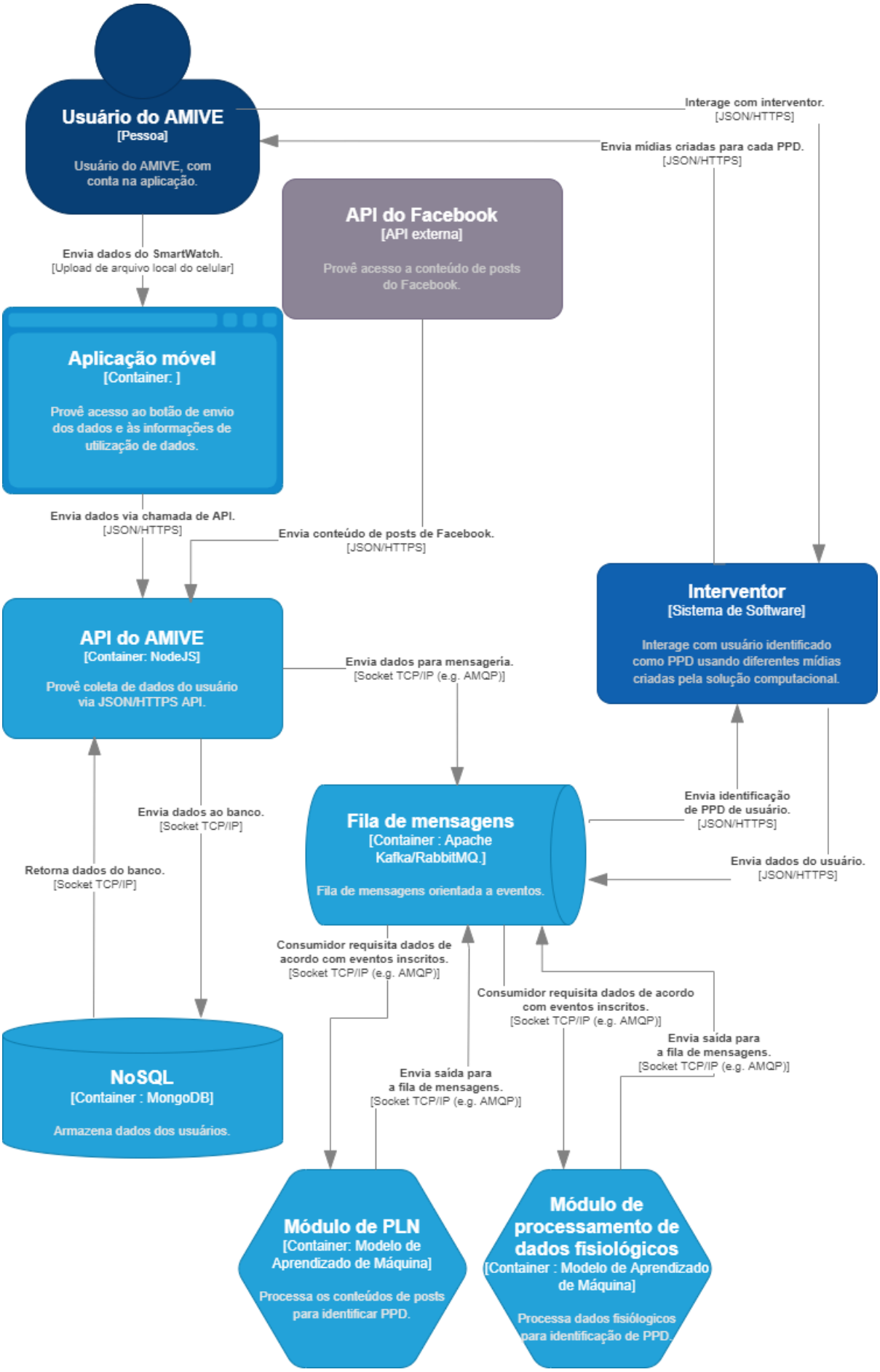

Figura 13 – Diagrama de contêiner do projeto Amive. Fonte: da própria autora.

5 Conclusão

O presente trabalho foi desenvolvido com o objetivo de abordar conceitos relacionados a três pilares: gestão da identidade e do acesso, proteção dos dados e segurança da arquitetura. Eles formam base para a proposta de arquitetura baseada em microsserviços do projeto Amive.

Embora exista muito trabalho a ser feito no sentido de aprofundar o conteúdo discutido no capítulo 3 para o contexto de implementação da solução do Amive, esta pesquisa serviu como base para a arquitetura e para as considerações de segurança da aplicação.

A arquitetura proposta considera aspectos voltados tanto à segurança quanto à escalabilidade da aplicação, questões de suma importância ao se lidar com uma solução que trata dados sensíveis e que poderá incorporar módulos futuros não planejados previamente.

Por último, esta pesquisa deixa em aberto alguns pontos de discussão e trabalhos futuros que, devido ao tempo limitado do trabalho de conclusão de curso, podem ser explorados posteriormente.

5.1 Pontos de verificação

A proposta cumpriu com os seguintes pontos:

- Adoção da arquitetura baseada em microsserviços, que permite escalabilidade e uso de conteinerização.
- Isolamento do banco de dados em relação a acesso direto por usuários e outros microsserviços, por meio de API planejada com este propósito.
- Comunicações por rede entre os serviços utilizando o protocolo HTTPS ou outro protocolo sobre o TLS.
- Adoção de uma fila de mensagens designada para prover os dados aos consumidores inscritos em eventos de seu interesse, que pode ser tanto do tipo *message broker* quanto *publish-subscribe*.

Visto que o projeto Amive encontra-se em andamento e a implementação da arquitetura será realizada por diferentes pesquisadores envolvidos, além do tempo limitado para desenvolvimento do presente trabalho de conclusão de curso, apenas os pontos

referidos acima foram escolhidos. Eles são essenciais para fundamentar o restante do desenvolvimento e devem ser seguidos.

Os outros pontos discutidos nas seções de “Melhores práticas” do capítulo 3 podem e devem ser incluídos no refinamento da arquitetura para o projeto Amive em trabalhos futuros. Alguns exemplos de pesquisas que podem derivar deste trabalho serão citados na seção 5.4.

5.2 Pontos de discussão

Uma das questões importantes e que devem ser levantadas em trabalhos futuros diz respeito ao acesso de dados por parte da empresa Meta[1] através do uso da API do Facebook. É necessário verificar quais dados da aplicação Amive e de seus usuários são tratados pela Meta e se há necessidade de restringir os privilégios da API.

Além disso, é importante verificar se o tratamento dos dados dentro da aplicação estará em *compliance* com a LGPD. Outras etapas que podem interferir e devem ser levadas em consideração são: implementação do *ChatBot* que, se utilizada uma solução de terceiros, pode ferir alguma política de proteção de dados pessoais e o acesso aos dados pela Samsung, no sentido de entender qual o tempo de retenção deles pela empresa através do aplicativo Samsung Health e se a empresa pode fornecê-los a terceiros.

Por último, a implementação atual da solução Amive, em estágio inicial, não realiza controle de acesso para proteger princípios de segurança como autenticação, autorização e confidencialidade. A Figura 14 ilustra a resposta, em formato JSON, dada pela API Amive à requisição do usuário, sem realização de controle de acesso. O *endpoint* utilizado para acessar a API do Amive foi ofuscado por motivos de confidencialidade.

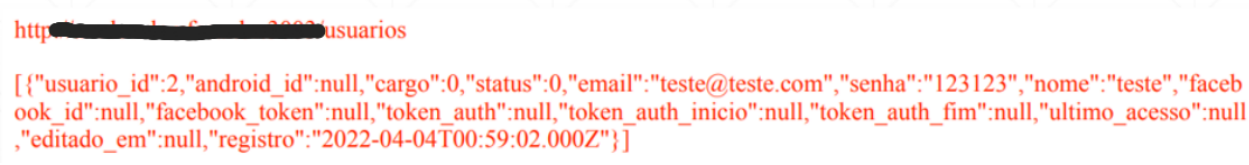

Figura 14 – Resposta da requisição à API do Amive, sem controle de acesso.

É de suma importância, portanto, implementar um controle de autenticação e de autorização, possivelmente utilizando um framework de código que contém ferramentas de autenticação consolidadas e testadas, de acordo com a linguagem escolhida para a implementação.

[1] <https://about.facebook.com/meta/>

5.3 Diretrizes em *compliance* com a LGPD

- Especificar quais os dados coletados e motivos da coleta.
- Ressaltar que a aplicação coleta dados sensíveis.
- Especificar quando os dados são coletados.
- Informar compartilhamento de dados com terceiros.
- Informar sobre tempo de retenção dos dados.
- Apresentar fundamento jurídico que autorize o tratamento dos dados.
- Ressaltar que dados não previstos podem ser coletados, desde que fornecidos com o consentimento do usuário.
- Informar direitos do usuário, como acesso e correção de dados.
- Empregar técnicas de segurança para proteger os dados.
- Permitir ao usuário reclamação a uma autoridade de controle.
- Notificar usuários quando a política de privacidade for modificada.
- Apresentar um contato para que os usuários tirem dúvidas.

5.4 Trabalhos futuros

Para continuação deste trabalho, estão definidas algumas possibilidades que poderiam contribuir com a melhoria do estudo e com sua implementação no contexto do projeto Amive:

- Proposta de arquitetura mais minuciosa para o projeto Amive, a nível dos diagramas de componentes e de código do modelo C4.
- Proposta de modelo de gestão robusta da identidade e do acesso para o projeto Amive, seguindo e adicionando às melhores práticas discutidas na seção 3.1.
- Implementação da arquitetura seguindo princípios do ciclo de vida do desenvolvimento de software seguro, através da adoção de algum framework, como o OWASP SAMM.
- Busca por vulnerabilidades existentes no software, com o auxílio de frameworks como o STRIDE e de ferramentas como o MITRE ATT&CK e através de realização de testes de segurança de análise estática e dinâmica e de testes de invasão.

- Validação do tratamento dos dados realizado pela aplicação seguindo políticas de *compliance* aplicáveis, sobretudo a LGPD.

Referências

AL-DEBAGY, O.; MARTINEK, P. A comparative review of microservices and monolithic architectures. *18th IEEE International Symposium on Computational Intelligence and Informatics*, 2018. Citado na página 23.

AUTH0 – Add Facebook Login to Native Apps. <https://auth0.com/docs/authenticate/identity-providers/social-identity-providers/facebook-native>. Último acesso em: 08/03/2022. Citado na página 40.

AUTH0 – Protocols. <https://auth0.com/docs/authenticate/protocols>. Último acesso em: 08/03/2022. Citado na página 40.

AWAD, M. et al. Password security: Password behavior analysis at a small university. *5th International Conference on Electronic Devices, Systems, and Applications (ICEDSA)*, 2016. Citado na página 43.

AWS – Pub/Sub Messaging. <https://aws.amazon.com/pt/pub-sub-messaging/>. Último acesso em: 14/03/2022. Citado 2 vezes nas páginas 13 e 25.

BENCHMARKS, C. Cis password policy guide. *Center for Internet Security*, 2021. Citado na página 42.

C4 Model – The C4 model for visualising software architecture. <https://c4model.com/>. Último acesso em: 26/03/2022. Citado 2 vezes nas páginas 13 e 31.

CHANDRAMOULI, R. Security strategies for microservices-based application systems. *National Institute of Standards and Technology*, 2019. Citado 3 vezes nas páginas 21, 23 e 24.

CLOUDFLARE – O que é o modelo OSI? <https://www.cloudflare.com/pt-br/learning/ddos/glossary/open-systems-interconnection-model-osi/>. Último acesso em: 28/03/2022. Citado 2 vezes nas páginas 13 e 53.

DEEPAK, G. C. *A Critical Comparison of NoSQL Databases in the Context of ACID and BASE*. Tese (Doutorado) — St Cloud State University, 2016. Citado na página 27.

DOCKER – Use containers to Build, Share and Run your applications. <https://www.docker.com/resources/what-container/>. Último acesso em: 28/03/2022. Citado 3 vezes nas páginas 13, 26 e 27.

IAM – AWS Identity and Access Management. <https://aws.amazon.com/pt/iam/>. Último acesso em: 08/03/2022. Citado na página 21.

KRUIDENBERG, D. *From monoliths to microservices*. [S.l.]: Universiteit van Amsterdam, 2018. Citado na página 23.

LGPD – Classificação dos Dados. 2021. <https://www.gov.br/cidadania/pt-br/acesso-a-informacao/lgpd/classificacao-dos-dados>. Último acesso em: 07/03/2022. Citado na página 47.

LGPD – Proteção de Dados. <https://www.gov.br/defesa/pt-br/acesso-a-informacao/lei-geral-de-protecao-de-dados-pessoais-lgpd>. Último acesso em: 08/03/2022. Citado na página 29.

MELI, M.; MCNIECE, M. R.; REAVES, B. How bad can it git? characterizing secret leakage in public github repositories. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019. Citado na página 41.

MITRE – ATTCK Matrix for Enterprise. <https://attack.mitre.org/>. Último acesso em: 28/03/2022. Citado 3 vezes nas páginas 13, 33 e 34.

O'REILLY – O'Reilly's Microservices Adoption in 2020 Report Finds that 92% of Organizations are Experiencing Success with Microservices. <https://www.oreilly.com/pub/pr/3307>. Último acesso em: 08/03/2022. Citado 3 vezes nas páginas 9, 11 e 19.

OWASP – OWASP SAMM. <https://owasp.org/www-project-samm/>. Último acesso em: 29/03/2022. Citado 2 vezes nas páginas 13 e 50.

OWASP – Threat Modeling. <https://owasp.org/www-community/Threat_Modeling>. Último acesso em: 08/03/2022. Citado na página 29.

PREVASIO – Operation "Red Kangaroo": Industry's First Dynamic Analysis of 4M Public Docker Container Images. <https://www.prevasio.io/blog/operation-red-kangaroo-industrys-first-dynamic-analysis-of-4m-public-docker-container-images>. Último acesso em: 28/03/2022. Citado na página 52.

RICHARDSON, C.; SMITH, F. *Microservices From Design to Deployment*. [S.l.]: NGINX, Inc., 2016. Citado 2 vezes nas páginas 23 e 24.

RJAIBI, N.; RABAI, L. Developing a novel holistic taxonomy of security requirements. *The International Conference on Soft COmputing and Software Engineering (SCSE)*, 2015. Citado na página 32.

ROSS, R. et al. Developing cyber-resilient systems: A systems security engineering approach. *NIST Special Publication 800-160 volume 2*, 2021. Citado na página 50.

SCHEEPERS, M. J. Virtualization and containerization of application infrastructure: A comparison. *21st Twente Student Conference on IT*, 2014. Citado na página 25.

SHANKAR, A.; SHETTY, R.; K, B. N. A review on phishing attacks. *International Journal of Applied Engineering Research*, 2019. Citado na página 45.

THE SECURITY FACTOR – Password cracking speed. <https://www.thesecurityfactory.be/password-cracking-speed/>. Último acesso em: 10/03/2022. Citado 2 vezes nas páginas 13 e 43.

THREAT MODELER – STRIDE, VAST, TRIKE, MORE: Which threat modeling methodology is right for your organization? <https://threatmodeler.com/threat-modeling-methodologies-overview-for-your-business/>. Último acesso em: 08/03/2022. Citado na página 31.

WANG, R.; YANG, Z. Sql vs nosql: A performance comparison. 2017. Citado na página 28.