UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Departamento de Computação

Trabalho de conclusão de curso - TCC

Guilherme de Oliveira

O uso de CMSs na construção de MVPs

São Carlos - São Paulo

2023

Guilherme de Oliveira

O uso de CMSs na construção de MVPs

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação como parte dos requisitos para a conclusão da graduação em Engenharia de Computação.

Orientação Prof. Dr. Daniel Lucrédio

São Carlos - São Paulo

2023

Dedico esta monografia à minha querida irmã Nicoly Karoline de Oliveira (in memoriam), que sempre esteve ao meu lado e me apoiou a correr atrás dos meus sonhos.

Agradecimentos

Aos meus pais e irmãs, que me incentivaram nos momentos difíceis e compreenderam a minha ausência enquanto eu me dedicava à realização deste trabalho. Também, aos meus colegas de curso, cm quem convivi intensamente durante os últimos anos, pelo companheirismo e pela troca de experiências que me permitiram crescer não só como pessoa, mas também como formando.

"Educação é uma descoberta progressiva de nossa própria ignorância.
(Voltaire)

Resumo

Startups em fase inicial necessitam coletar opiniões de possíveis clientes sobre a solução ofertada no menor tempo possível, para que possam se adaptar a alta concorrência do mercado. Nesse contexto, MVPs, ou Produto Mínimo Viável, são utilizados para apresentar a solução proposta e coletar opiniões do mercado consumidor de forma prática. Este trabalho defende a viabilidade em construir MVPs de software com a utilização de CMSs (Sistemas de Gerenciamento de Conteúdo). Como estudo de caso, o Directus, um CMS de código aberto, foi escolhido como ferramenta para a construção de uma aplicação de automação de *backup* para banco de dados. Ao longo do desenvolvimento, é demonstrado como a ferramenta foi adaptada a fim de construir a aplicação, aproveitando-se do alto grau de customização que os CMSs oferecem. Como discussão, o trabalho sugere o uso dessas ferramentas apenas para aplicações focadas em análise e gerenciamento de dados e eventos, citando a impossibilidade de utilizar CMSs na construção de aplicações de jogos ou *streaming*, por exemplo. Assim, o trabalho conclui que CMSs podem ser opções levadas em consideração para facilitar o processo de desenvolvimento de software.

Palavras-chave: MVP, CMS, Startup, Software, Directus, Desenvolvimento Web

Abstract

Early-stage startups need to collect costumer opinions about the offered solution in a short time stamp, so they can adapt to the high competitiveness of the market. In that context, Minimum Valuable Products (MVP's) are created to present the proposed solution and collect opinions from the consumer market in a practical way. This article proposes the use of CMS's (Content Management Systems) to build software MVP's. As a case-of-study, the Directus, an open-source CMS, was picked as the tool to build an application that automates database backups. Over the development phase, it is demonstrated how the tool was adapted in order to build the application, taking advantage of the high level of customization from CMS's. As a discussion, the work suggests the use of these tools only for data-driven applications, focused on analysis and management of data and events, citing the impossibility of using CMSs in the construction of game or streaming applications, for example. Then, this work concludes that CMS's can be considered options to facilitate the software development process.

Keywords: MVP, CMS, Startup, Software, Directus, Web Development

Lista de ilustrações

Sumário

1 Introdução

1.1 Contexto

Com o crescimento acelerado de *startups* na área de tecnologia, empresas em estado inicial (*early-staged*) dependem da velocidade com que conseguem entregar valor ao cliente final. Essas *startups* precisam validar seu modelo de negócio e ideia de produto coletando *feedbacks* contínuos de seus clientes e *stakeholders* no menor tempo possível, em prol da saúde do negócio (NILSSON; PETERSSON, 2013). Assim, surge a necessidade da criação de um protótipo ou produto inicial, conhecido como Produto Mínimo Viável, ou simplesmente MVP (Minimum Viable Product), que permite ao usuário interagir e opinar sobre a solução ofertada.

O MVP auxilia o empresário a aprender sobre seu modelo de negócio o mais rápido possível, podendo transformar hipóteses em fatos e validar ideias com o público consumidor (RIES, 2011). Dentro deste conceito de aprendizagem e experimentação, assume-se que qualquer método que colete informações de clientes possa ser considerado um MVP, como um simples questionário de pesquisa. Porém, de acordo com a pesquisa de Melegati et al. (2020), na prática, *startups* consideram a ideia de um MVP como um protótipo de funcionalidade única (ou mínimas) que transmitam o principal valor do produto final ao cliente, permitindo então a validação do software como um produto. Assim, este artigo irá considerar MVP como um protótipo mínimo de software.

Um MVP de sucesso não necessariamente entrega as funcionalidades que o usuário espera da aplicação, mas permite levantar requisitos para a construção do produto final, seja pela detecção da falta de funcionalidades ou pela aprovação ou desaprovação de alguma funcionalidade existente. O MVP precisa ser desenvolvido de maneira rápida, onde apenas um desenvolvedor possa conseguir implementar a funcionalidade alvo em poucos dias ou semanas. Além disso, o MVP deve ser construído com baixo custo de investimento, assim, ferramentas de acesso livre para construção de softwares devem ter prioridade na escolha das tecnologias.

Nesse cenário, destaca-se também o uso de sistemas CMSs (Content Management System). Esse tipo de sistema é conhecido por sua capacidade em gerenciar conteúdos no sentido literal da palavra: usuários utilizam primariamente para gerenciar imagens, vídeos e textos de aplicações (SINGH; CHAUDHARY; CHAUDHARY, 2023). Porém, também pode-se enxergar esses sistemas como plataformas para construção de softwares. De acordo com Sobri, Abas e Yassin (2023), CMSs podem ser utilizados para implementar aplicações de forma rápida e com baixo custo de desenvolvimento, além de serem plataformas com

baixa curva de aprendizagem para iniciantes no desenvolvimento de software. A utilidade de CMSs para a construção de aplicações é ainda maior quando se trata de ferramentas *open-source*, devido ao alto grau de customização e controle da plataforma (SINGH; CHAUDHARY; CHAUDHARY, 2023). Assim, pode-se argumentar que sistemas CMS são ferramentas com possibilidade de adaptação para criação de aplicações customizadas.

1.2 Objetivos

Este trabalho propõe a implementação de um MVP utilizando uma ferramenta CMS *open-source*, o Directus (DIRECTUS. . . , 2023). Essa ferramenta foi escolhida devido ao seu alto grau de customização, oferecendo uma plataforma de extensões onde o desenvolvedor pode alterar o comportamento da aplicação e tomar controle de grande parte do sistema, permitindo construir aplicações com diferentes propósitos. O trabalho busca demonstrar que é possível utilizar ferramentas CMS para:

- Construir aplicações de forma rápida e com baixo custo de desenvolvimento.
- Apresentar funcionalidades de maneira clara aos usuários.
- Permitir ao usuário entender as vantagens e desvantagens da solução apresentada.
- Fornecer uma interface amigável para usuários com baixo nível de conhecimento técnico.

Assim, o objetivo principal do trabalho é demonstrar que é possível utilizar um CMS para construir MVPs.

1.3 Organização

Para atingir os objetivos esperados e demonstrar seus resultados, este trabalho está organizado da seguinte forma: A seção 2 discorre sobre a literatura existente e mostra outras possíveis abordagens para implementação de MVPs, como também outros possíveis casos de uso para ferramentas de CMS. A seção 3 demonstra o desenvolvimento da aplicação de forma técnica, passando por detalhes sobre a plataforma do Directus e como as extensões da ferramenta foram utilizadas para atender aos requisitos levantados. Na seção 4 é discutida se a solução apresentada cumpre os objetivos deste trabalho. Por fim, a seção 5 exibe as conclusões obtidas a partir do estudo de caso realizado.

2 Trabalhos Correlatos

Alguns trabalhos foram trazidos da literatura para dar maior embasamento na defesa do uso de CMSs para a criação de aplicações. Em uma publicação na revista IJACSA, Hussein e Al-Kaddo (2014) propõem a utilização de um CMS, conhecido como Joomla, na construção de uma ferramenta de aprendizagem online. Após levantar os requisitos esperados para a aplicação, os autores argumentam o quanto a modularidade de ferramentas CMS permite adaptar a aplicação para satisfazer os objetivos propostos. Assim, utilizando extensões criadas pela comunidade, foi possível adicionar funcionalidades em um CMS a fim de criar uma plataforma de aprendizagem online. Foram utilizadas extensões que adicionam gerenciamento de arquivos, fórum de discussões e mensagens privadas. Os autores concluem que CMSs podem ser facilmente alterados a fim de se construir aplicações com alto grau de conectividade e troca de informações entre os usuários.

O estudo de Sobri, Abas e Yassin (2023) compara três ferramentas que auxiliam desenvolvedores no desenvolvimento de aplicações, duas delas são CMSs amplamente conhecidos: Directus e Strapi. A terceira é um produto em formato *Backend-as-a-Service* (BaaS) chamado Supabase. Ao longo do trabalho, os autores propõem a construção de uma aplicação fictícia e analisam as três ferramentas dentro de algumas categorias de requisitos: Serviço de autenticação, documentação, modularidade do sistema, integrações e popularidade. Após levantar os pontos positivos e negativos para cada uma das opções estudadas, concluem que o Directus é a ferramenta mais apropriada para o desenvolvimento do software escolhido, por possuir uma arquitetura totalmente extensível e de código aberto, além de oferecer, de forma nativa, gerenciamento de cargos e permissões e apresentar uma interface gráfica amigável para usuários.

Já no trabalho realizado por Wan (2016) é apresentada uma proposta para o uso de CMSs no campo industrial. O autor propõe o uso de um CMS chamado de Dupral para centralizar as informações e resultados obtidos durante o ciclo de vida de produtos e processos internos da empresa. Além de centralizar os antigos sistemas em um único, o autor cita que o CMS permitiu oferecer um alto nível de customização através de módulos e, também, controle de acesso de usuários de acordo com cargos.

Os trabalhos estudados trazem evidências de que ferramentas CMS podem ser adaptadas para construir aplicações web. Quando se trata de ferramentas *open-source*, é possível utilizar módulos customizados para alterar o comportamento da aplicação em diferentes âmbitos.

Este trabalho reforça os estudos apresentados, propondo a construção de um MVP utilizando um CMS como ferramenta base. Além das vantagens trazidas no estudo de Sobri,

Abas e Yassin (2023), este trabalho apresenta um estudo de caso onde as extensões do Directus são os atores principais no uso da ferramenta para a construção de uma aplicação.

3 O estudo de caso

Como estudo de caso, o Directus foi escolhido como o CMS para construção de um MVP. Os fatores que motivaram a escolha dessa ferramenta foram, primeiramente, a apresentação de uma documentação com exemplos e ilustrações que auxiliam o leitor a conhecer a plataforma. A ferramenta também possui um sistema de extensões completo, o que facilita adaptar e pivotar o propósito do CMS para construir aplicações diversas. Além disso, o fato de ser uma ferramenta *open-source* permite que a comunidade compartilhe conteúdo e extensões entre si.

Para exemplificar o uso do Directus na construção de um MVP, foi simulado um cenário onde uma *startup* fictícia pretende averiguar a validade de uma aplicação para automação de *backups* como produto. A fim de simplificar o estudo de caso, o MVP construído permite que apenas bancos de dados do tipo Postgres (GROUP,) sejam cadastrados. Os seguintes requisitos foram levantados como necessários para apresentar o MVP proposto:

1. O usuário deve ser capaz de cadastrar, editar e excluir conexões com banco de dados.
2. O usuário deve poder configurar o período em que os *backups* acontecerão para cada conexão.
3. O usuário deve conseguir executar um *backup* manual sempre que necessário.
4. O usuário deve ser capaz de pausar o agendamento de *backups* para determinada conexão.
5. O sistema deve realizar *backups* mesmo que a aplicação não esteja em uso.
6. O sistema deve recuperar o agendamento de *backups* quando reiniciado.
7. O sistema deve excluir agendamentos de *backup* para uma conexão deletada pelo usuário.

Ao longo deste trabalho é demonstrado como um desenvolvedor com conhecimento em *JavaScript* e *Vue.js* pode customizar o comportamento do Directus para implementar o MVP proposto.

3.1 Sobre o Directus

O Directus, como um sistema de gerenciamento de conteúdo, cria uma abstração na construção de banco de dados relacionais a fim de possibilitar que usuários não técnicos

também possam alterar estruturas e customizar o sistema. Para isso, a ferramenta utiliza de um paralelo conceitual nos termos de banco de dados relacionais a fim de facilitar o entendimento do usuário. Os principais conceitos para o gerenciamento de dados dentro do Directus são: Coleções, Campos, Itens e Fluxos.

Uma Coleção tem um paralelo 1:1 com tabelas de banco de dados relacionais. Assim como tabelas, Coleções são utilizadas para categorizar um conjunto específico de dados do sistema. Por padrão, o Directus já possui Coleções administrativas, por exemplo:

- Usuários - Coleção para registrar informações sobre cada usuário cadastrado na aplicação.
- Cargos - Coleção que lista os diferentes cargos de usuários e relaciona com as permissões que cada cargo possui dentro da ferramenta.
- Arquivos - Coleção para controlar todos os arquivos que passaram pela aplicação, como fotos, vídeos, arquivos de texto e outros formatos.

Dentro do Directus, Campos representam as colunas de uma tabela de banco de dados. Além de ajudar a organizar os dados da tabela em sub-categorias, no Directus os Campos são os principais meios de interação do usuário com a aplicação.

Dependendo de como são configurados, Campos permitem diferentes maneiras de interação e exibição de seus dados para o usuário. Por exemplo, se um Campo é configurado para armazenar data e horário, o desenvolvedor pode configurar o Directus para exibir um componente de calendário para escolha da data, melhorando a experiência do usuário.

Os Itens são como registros em tabelas de banco de dados relacionais. Cada linha em uma tabela é mapeada para um Item dentro do Directus. Ao listar os Itens de uma Coleção, o Directus exibe cada um dos Campos do item de acordo com o tipo do Campo. A interface facilita a experiência do usuário ao exibir Campos relacionais em uma listagem única, possibilitando visualizar o Item atual e o Item da Coleção que possui relacionamento com o Campo.

Os Fluxos são ferramentas poderosas dentro do Directus. São funções baseadas em eventos e altamente configuráveis, possibilitando processamento customizado dos dados e automação de tarefas. Um Fluxo pode ser criado para enviar notificações sempre que um novo Item é adicionado em uma Coleção, ou até para executar um *script* customizado em um determinado horário do dia. Cada Fluxo é composto por um gatilho e uma série de operações. Possíveis gatilhos para ativação de um Fluxo são:

- Eventos - Fluxos podem ser ativados via eventos de modificação em alguma Coleção, como criação, alteração ou exclusão de um item.

- Cron - Gatilhos configurados para executar conforme determinado período, especificado via expressão conhecida como *cron expression* (CRON. . . , 2023).
- Outro Fluxo - Fluxos também podem engatilhar outros Fluxos, criando uma cadeia de ações customizadas.
- Manual - Gatilho que executa Fluxos de acordo com a interação do usuário com um botão de ativação.

Além disso, por padrão, o Directus oferece uma lista de operações que podem ser encadeadas dentro de um Fluxo e ativadas de acordo com o gatilho configurado, alguma delas são:

- CRUD de dados - Operações podem criar, ler, atualizar ou deletar itens de qualquer Coleção na qual o usuário tenha permissão de acesso.
- Log - Operação que cria mensagens de log dentro do servidor do Directus, permitindo que o adminstrador obtenha informações sobre a execução do Fluxo.
- Notificação - Exibe notificações em tempo real para usuários dentro da plataforma do Directus.
- Execução de *script* - Operação que permite a execução de código *JavaScript* diretamente dentro da ferrramenta.

Além dos conceitos principais para gerenciamento de dados, a ferramenta também oferece um ecossistema de extensões, permitindo que desenvolvedores de software customizem vários aspectos da aplicação. Assim, a experiência do usuário pode ser adaptada a fim de criar novas funcionalidades e atingir os objetivos esperados da aplicação.

3.2 Construindo o MVP

O primeiro passo para a construcão da ferramenta foi configurar um ambiente de desenvolvimento para o Directus. Um banco de dados previamente configurado foi necessário para armazenar as configurações e dados da aplicação. A documentação do Directus possui um guia para configurar um ambiente de desenvolvimento utilizando Docker (QUICKSTART. . . ,).

3.2.1 Operação de Backup

Após a configuração inicial da aplicação, a primeira customização necessária foi uma extensão de operação customizada, responsável por construir a funcionalidade principal da

aplicação proposta: o agendamento de *backups*. A operação contém o código *JavaScript* necessário para executar o *backup* de uma conexão com banco de dados previamente cadastrada, utilizando a *pg_dump*, uma ferramenta criada pela comunidade para realizar *backups* de bancos de dados *Postgres* (PG_DUMP...,). O Directus fornece uma seção em sua documentação que explica como criar operações customizadas, construindo a interface de usuário e o código que será executado dentro do servidor (CUSTOM...,).

O pseudocódigo apresentado no Algoritmo 1 foi utilizado como base para a construção da lógica realizada pela operação. O código em *JavaScript* que implementa o algoritmo está exibido no Apêndice A.

Algoritmo 1 Operação customizada de backup

```
1:  procedure OPERACAOBACKUP(dadosConexao, pastaDestino)
2:      nomeArquivo ← gerarNomeArquivo(dadosConexao)
3:      statusBackup ← executarPgDump(dadosConexao, nomeArquivo)
4:      if statusBackup = sucesso then
5:          tamanhoBackup ← calcularTamanhoBackup()
6:          tempoOperacao ← calcularTempoOperacao()
7:          return tamanhoBackup, tempoOperacao
8:      else
9:          return mensagemErro
10:     end if
11: end procedure
```

Na construção da interface, algumas opções foram utilizadas para definir quais entradas a operação criada necessita para ser executada. A Figura 1 mostra as opções de configuração da operação criada, onde é possível selecionar a pasta onde os *backups* deverão ser armazenados.

Agora que a operação customizada de backup foi criada e adicionada ao Directus, um Fluxo foi configurado para executar a operação. A Figura 2 mostra a configuração final do Fluxo criado. O Fluxo é engatilhado através de um segundo Fluxo, que é configurado na sessão 3.2.3 e está representado no primeiro bloco da imagem. O primeiro bloco após o gatilho é a operação de *backups* seguida de uma segunda operação de *log* que foi encadeada para registrar possíveis erros de execução.

3.2.2 Modelo de Dados

Algumas abstrações foram necessárias para aproveitar do poder do modelo de dados do Directus e criar adaptações que possibilitem atingir os objetivos propostos. Para cumprir o requisito 1 do MVP, foi necessária a criação de uma Coleção chamada de "Conexões". O intuito da Coleção é controlar as diferentes conexões com banco de dados que o usuário queira que o sistema faça operações de *backup*. Assim, ao adicionar um novo Item (uma nova conexão) na Coleção, o sistema deverá agendar o *backup* automático

Figura 1 – Opções de configuração da operação de *backup*

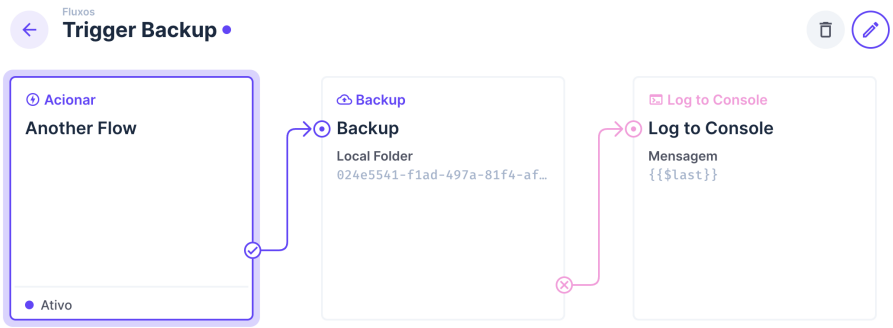

Figura 2 – Configuração do Fluxo com a operação de backup customizada

daquela instância de acordo com as opções configuradas pelo usuário. Da mesma forma, ao atualizar um Item, o agendamento do *backup* também deverá ser atualizado. E, ao deletar, o agendamento de *backup* para aquela conexão deverá ser cancelado. O Directus deve ser configurado para apresentar a nova Coleção contendo os seguintes Campos:

- *name* - Campo para armazenar o apelido para a conexão, facilitando a identificação por parte do usuário.
- *active* - Campo que determina se o *backup* automático está ativado para a conexão.
- *host* - Endereço de IP do servidor onde o banco de dados está hospedado.
- *database* - Nome do banco de dados alvo da operação de *backup*.
- *port* - Porta do servidor onde a conexão com o banco de dados está exposta.

- *user* - Nome de usuário com acesso ao banco de dados.
- *password* - Senha do usuário com acesso ao banco de dados.
- *cron* - Expressão no formato *cron* que define o período em que as operações de backup serão executadas.

Uma segunda Coleção chamada “Backups” foi criada, com o intuito de registrar todas as operações de *backup* executadas pela ferramenta e, no futuro, poder exibir métricas sobre a performance das operações. Essa nova Coleção foi relacionada de forma *M2O* (muitos pra um) com Itens da Coleção de "Conexões", pois cada operação de *backup* faz referência a alguma conexão de banco de dados específica.

Os Campos mapeados para a nova Coleção de registro de *backups* são:

- *size_in_bytes* - Campo que armazena o tamanho final do *backup* gerado, em *bytes*.
- *elapsed_time* - Tempo total de execução da operação de *backup*, em milissegundos.
- *status* - Campo que determina o estado atual do *backup*, variando entre: pendente, sucesso ou erro.
- *error_message* - Armazena a mensagem de erro que ocorreu durante a operação, se existir.
- *connection* - Identifica com qual conexão de banco de dados o *backup* está atrelado.

A Figura 3 exibe o modelo de dados final para a Coleção de conexões. Cada uma das caixas na figura representa um dos Campos listados. Para facilitar a visualização de coleções relacionadas, o Directus também adiciona a lista de *backups* como um Campo dentro de uma conexão, como mostra o último item da lista.

A fim de oferecer uma melhor experiência de usuário no gerenciamento das conexões de banco de dados, foram criadas duas extensões de interface para a Coleção. As extensões de interface permitem alterar o modo com que o usuário interage com o Campo, a depender do tipo de dado que o Campo armazena. A primeira interface customizada foi uma que permite ao usuário criar uma expressão *cron* utilizando linguagem natural. A segunda foi uma interface que lista os *backups* atrelados com determinada conexão, exibindo estatísticas como tamanho e tempo decorrido em uma tabela, a fim de facilitar a leitura. O código responsável por cada uma das interfaces pode ser encontrado no Apêndice B e C, respectivamente. A Figura 4 mostra como as interfaces aparecem ao editar uma conexão que já possui *backups* listados. O Campo *cron* apresenta um componente que possibilita a escolha do período através de linguagem natural e, o Campo *backups* serve apenas para listagem das operações executadas. Os demais Campos apresentam componentes de entrada de acordo com o tipo de cada um.

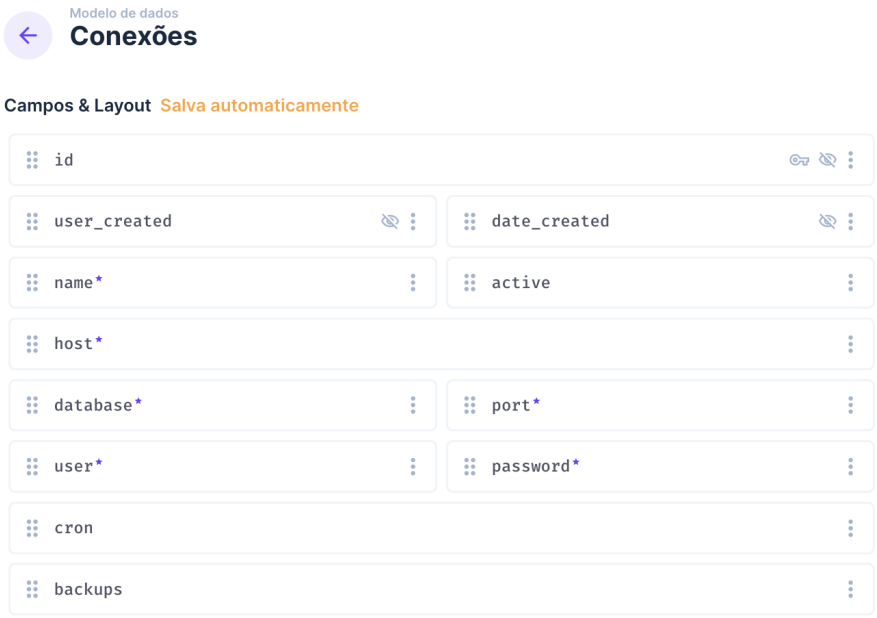

Figura 3 – Modelo de dados para a coleção de Conexões

3.2.3 Construção dos Fluxos

Com o intuito de fornecer uma opção de *backups* manuais para o usuário e cumprir com o requisito 3 do MVP, foi necessário configurar um segundo Fluxo. Este Fluxo é responsável por ativar o Fluxo que contém a operação de *backup*, apresentado na seção 3.2.1.

O novo Fluxo foi configurado com um gatilho manual, que permite ao usuário selecionar um ou mais Itens da Coleção de conexões e ativar ambos os Fluxos de forma encadeada, para executar uma operação de *backup* em cada conexão selecionada. O Fluxo é responsável por receber os identificadores das conexões selecionadas pelo usuário, ler os dados da Coleção e repassar os dados para o Fluxo de *backup*, executando uma operação *backup* para cada conexão selecionada. A Figura 5 apresenta a configuração responsável por esse comportamento. A primeira caixa representa o gatilho manual do Fluxo, seguido por uma operação que realiza a leitura dos dados para as conexões selecionadas e repassa para a segunda operação, responsável por ativar o Fluxo de *backup*.

Para realizar o registro dos *backups* realizados, novas operações foram adicionadas ao primeiro Fluxo. Agora, além de executar a operação de *backup*, o Fluxo também é responsável por criar Itens na Coleção "Backups"e, após a execução, atualizar o Item criado afim de registrar o tamanho do arquivo gerado e o tempo decorrido. Em caso de falha na operação de *backup*, o Fluxo também atualiza o Item para registrar a mensagem de erro. Ao final, o Fluxo de *backup* apresenta uma estrutura conforme exibe a Figura 6, onde, além da operação de *backup*, também possui as novas operações para criação e atualização de Itens na coleção "Backups".

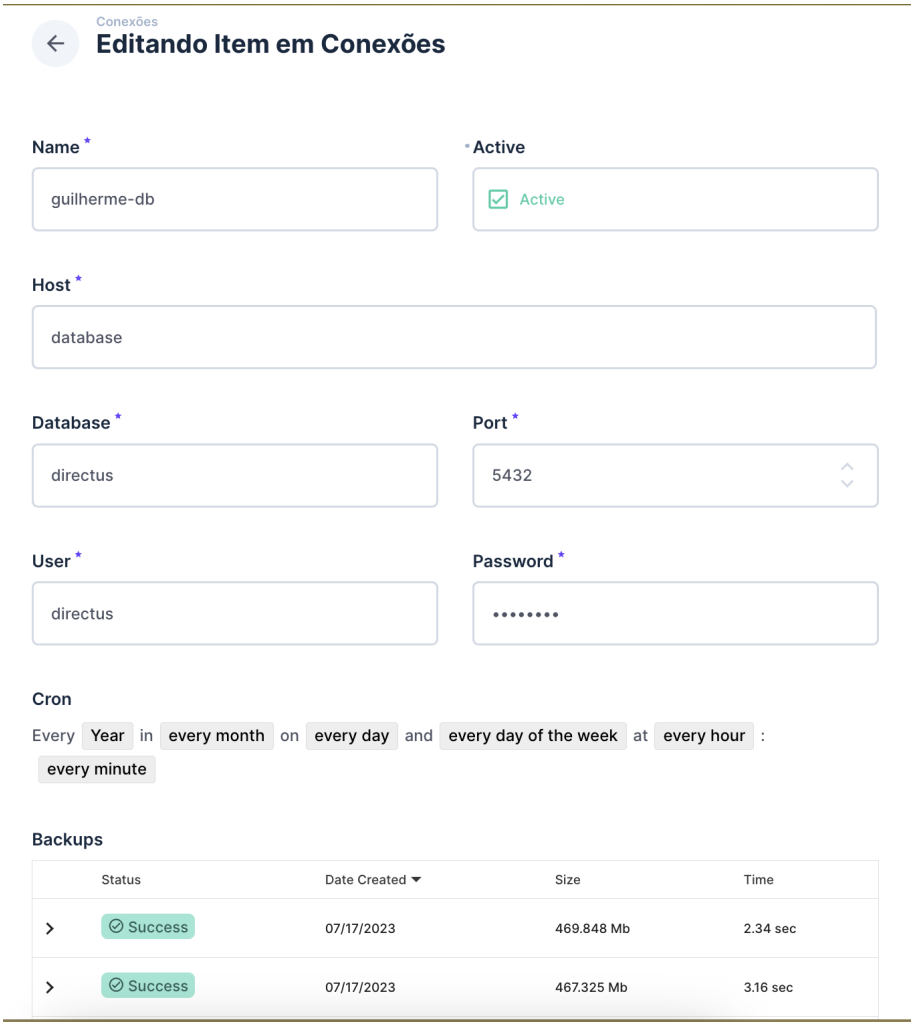

Figura 4 – Tela de interação com uma conexão, com extensões de interfaces customizadas

Após a configuração de ambos os Fluxos, ao visualizar a lista de conexões cadastradas, é possível visualizar o botão que ativa o Fluxo manual para os Itens selecionadas, cumprindo com o requisito 3 do MVP e oferecendo a possibilidade de execução arbitrária de *backups* por parte do usuário. A figura 7 apresenta o botão de gatilho do Fluxo, posicionado na barra lateral direita.

3.2.4 Automação de Backups

Com o modelo de dados e os Fluxos configurados no Directus, foi necessário realizar a automação dos *backups* em si, respeitando o período definido pelo usuário no cadastro da conexão e cumprindo com os requisitos 2 e 5 do MVP.

A estratégia adotada para a automação das operações foi criar uma extensão de *hook* customizado que, ao perceber que o servidor do Directus foi iniciado, lê as conexões que estão cadastradas e agenda um processo que ativará o Fluxo de *backup* programaticamente, respeitando o período definido na expressão *cron* de cada conexão ativa. Assim, o requisito 6 do MVP também é cumprido, pois permite que *backups* previamente agendados sejam

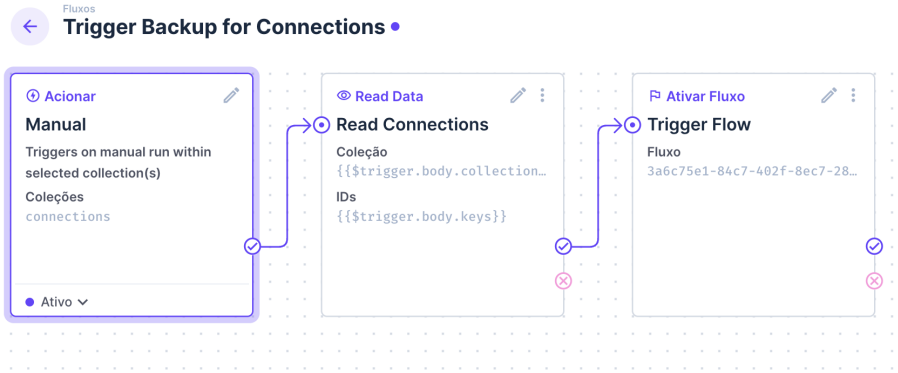

Figura 5 – Fluxo responsável por engatilhar o Fluxo de backup de acordo com as conexões selecionadas

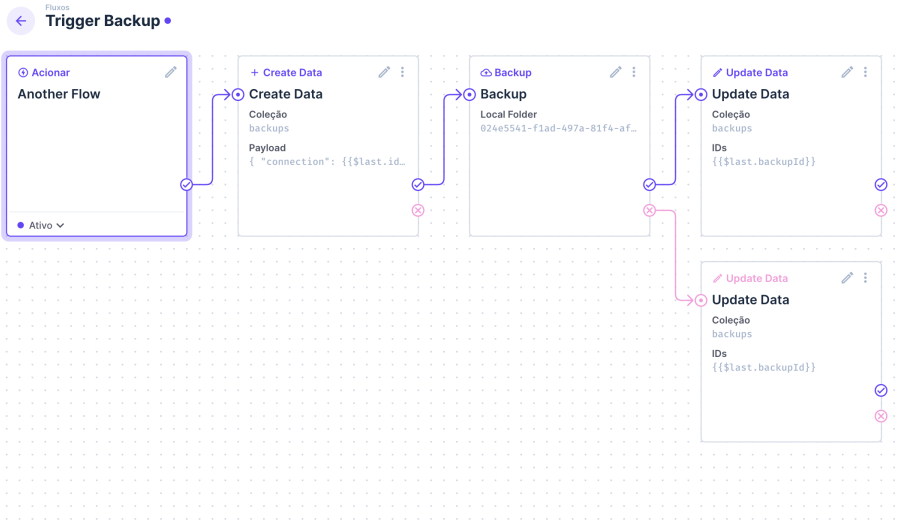

Figura 6 – Fluxo com operações para criação e atualização de registros de backups

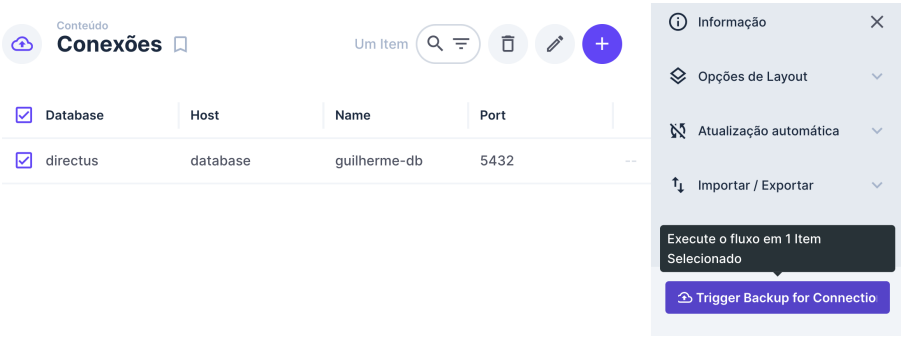

Figura 7 – Botão que ativa o Fluxo manual para as conexões selecionadas, na barra lateral direita

restaurados em caso de reinicialização do servidor. O *hook* também é responsável por deletar o processo de *backup* agendado para uma conexão se a mesma for excluída ou desativada pelo usuário, além de atualizar o período de agendamento se a expressão *cron* da conexão for alterada, cumprindo com os requisitos 7 e 4, respectivamente.

Agora, ao iniciar a aplicação, é possível observar nos *logs* do servidor que uma

operação de *backup* previamente cadastrada está sendo executada. A Figura 8 exibe as linhas de *log* com o início e fim de cada operação realizada. O código do *hook* que automatiza os *backups* na inicialização do servidor pode ser encontrado no Apêndice D.

```
[07:10:00.287] INFO: [directus-backup-operation] Backing up connection to database directus on host database
[07:10:00] POST /flows/trigger/f6e41f44-72fe-4ccb-a04b-3cef11e32ba0 204 21ms
[07:10:01.754] INFO: [directus-backup-operation] New database backup created: guilherme-db-2023.8.4.7.10.dump
[07:11:00] POST /flows/trigger/f6e41f44-72fe-4ccb-a04b-3cef11e32ba0 204 15ms
[07:11:00.485] INFO: [directus-backup-operation] Backing up connection to database directus on host database
[07:11:01.793] INFO: [directus-backup-operation] New database backup created: guilherme-db-2023.8.4.7.11.dump
[07:12:00.697] INFO: [directus-backup-operation] Backing up connection to database directus on host database
[07:12:00] POST /flows/trigger/f6e41f44-72fe-4ccb-a04b-3cef11e32ba0 204 35ms
[07:12:02.157] INFO: [directus-backup-operation] New database backup created: guilherme-db-2023.8.4.7.12.dump
[07:13:00.952] INFO: [directus-backup-operation] Backing up connection to database directus on host database
[07:13:00] POST /flows/trigger/f6e41f44-72fe-4ccb-a04b-3cef11e32ba0 204 15ms
[07:13:02.428] INFO: [directus-backup-operation] New database backup created: guilherme-db-2023.8.4.7.13.dump
```

Figura 8 – *Logs* do servidor do Directus com as chamadas para a operação de backup

4 Discussões e Resultados

Durante o processo de construção do MVP para uma aplicação de automação de backups, o Directus se mostrou capaz de ser adaptado a fim de construir uma aplicação com um propósito diferente daquele oferecido por um CMS.

Apesar do Directus possuir uma boa documentação e apresentar tutoriais, conhecimentos em *Vue.js* ainda são necessários na criação da maioria das extensões do Directus, o que pode ser um empecilho, visto que, de acordo com uma pesquisada realizada pelo site *Stack Overflow*, em Junho de 2023, apenas 16.38% dos desenvolvedores de aplicações web apresentam familiaridade com esse *framework* em específico (STACK. . . ,).

O MVP construído conseguiu oferecer ao usuário a experiência desejada e atingir os requisitos propostos. Isso foi possível devido ao uso correto das extensões do Directus e a modelagem da aplicação, o que pode não permitir com que desenvolvedores menos experientes cheguem nos mesmos resultados. Além disso, a capacidade de adaptação da ferramenta pode não ser válida para MVPs de aplicações que dependam de um grande nível de interação e complexidade, como aplicações de jogos ou *streaming*. Assim, este estudo defende o uso de CMSs na construção de aplicações focadas em análise de dados e gerenciamento de eventos, a fim de aproveitar ao máximo a customização de CMSs e suas extensões.

No final do estudo de caso, o Directus conseguiu cumprir todos os requisitos propostos para o MVP. A Figura 9 mostra a tela da aplicação que apresenta ao usuário o resumo dos *backups* realizados para uma conexão específica, bem como estatísticas para cada um deles e informações sobre possíveis erros que podem ter ocorrido durante a operação.

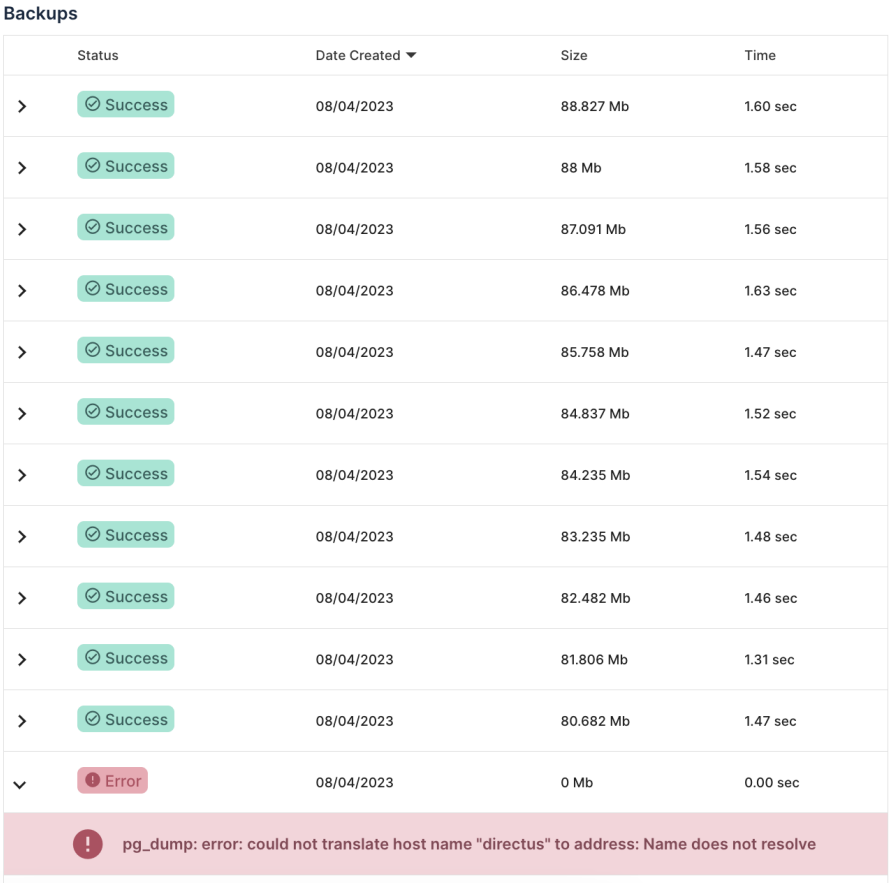

Figura 9 – Resumo dos backups realizados para um banco de dados, com estatísticas e mensagem de erro

5 Conclusão

Ao utilizar o Directus e seu ferramental de extensões, foi possível construir a ferramenta de automação de *backups* e cumprir todos os requisitos propostos para o MVP. O trabalho demonstrou que, ao conhecer os conceitos básicos de um CMS, desenvolvedores podem adaptar essas ferramentas a fim de construir aplicações baseadas em análise de dados e gerenciamento de eventos. Em um contexto de *startups* que precisam apresentar um MVP para o mercado, CMSs podem ser opções levadas em consideração para facilitar o processo de desenvolvimento.

Referências

CRON Expressions — docs.oracle.com. 2023. <https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm>. [Accessed 16-08-2023]. Citado na página 15.

CUSTOM Operations | Directus Docs — docs.directus.io. <https://docs.directus.io/extensions/operations.html>. [Accessed 17-08-2023]. Citado na página 16.

DIRECTUS: The Modern Data Stack, democratized. 2023. Acessado em 10 de Julho de 2023. Disponível em: <https://directus.io/>. Citado na página 10.

GROUP, P. G. D. *PostgreSQL — postgresql.org*. <https://www.postgresql.org/>. [Accessed 17-08-2023]. Citado na página 13.

HUSSEIN, R. R. A.; AL-KADDO, A. B. E-learning by using content management system (cms). *International Journal of Advanced Computer Science and Applications*, The Science and Information Organization, v. 5, n. 10, 2014. Disponível em: <http://dx.doi.org/10.14569/IJACSA.2014.051015>. Citado na página 11.

LENARDUZZI, V.; TAIBI, D. Mvp explained: A systematic mapping study on the definitions of minimal viable product. In: *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. [S.l.: s.n.], 2016. p. 112–119. Nenhuma citação no texto.

MELEGATI, J. et al. Mvp and experimentation in software startups: a qualitative survey. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. [S.l.: s.n.], 2020. p. 322–325. Citado na página 9.

NILSSON, H.; PETERSSON, L. How to manage technical debt in a lean startup. In: . [S.l.: s.n.], 2013. Citado na página 9.

PG_DUMP - postgresql.org. <https://www.postgresql.org/docs/current/app-pgdump.html>. [Accessed 17-08-2023]. Citado na página 16.

QUICKSTART | Directus Docs — docs.directus.io. <https://docs.directus.io/self-hosted/quickstart.html>. [Accessed 17-08-2023]. Citado na página 15.

RIES, E. *Lean Startup: How Today's Entrepreneurs Use Continous Innovation to Create Radically Successful Businesses*. New York: Crown Business, 2011. ISBN 978-0-307-88789-4. Citado na página 9.

SAADATMAND, M. Assessment of minimum viable product techniques: A literature review. In: . [S.l.: s.n.], 2017. Nenhuma citação no texto.

SINGH, A.; CHAUDHARY, A.; CHAUDHARY, K. Content management system. *Global Journal of Enterprise Information System*, v. 15, n. 1, p. 87–92, May 2023. Disponível em: <https://www.gjeis.com/index.php/GJEIS/article/view/713>. Citado 2 vezes nas páginas 9 e 10.

SOBRI, N. A. N.; ABAS, M. A. H.; YASSIN, A. I. M. Comparison between headless cms and backend-as-a-service products for e-suripreneur backend. *Mathematical Statistician and Engineering Applications*, v. 71, n. 1, p. 928–938, May 2023. Citado 3 vezes nas páginas 9, 11 e 12.

STACK Overflow Developer Survey 2023 — survey.stackoverflow.co. <https://survey.stackoverflow.co/2023/>. [Accessed 24-08-2023]. Citado na página 23.

WAN, S. Exploring the advantages of content management systems for managing engineering knowledge in product-service systems. In: . [S.l.: s.n.], 2016. v. 56. Citado na página 11.

A – Operação de Backup

```
export default defineOperationApi({
    id: "backup",
    handler: async ({
        storage,
        folder
    }, {
        database,
        services,
        getSchema,
        logger,
        data
    }) => {
        const start = performance.now()

        const schema = await getSchema()

        const filesService: FilesService = new services.FilesService({
            schema,
            knex: database,
        })

        const connection: IConnection = database.client.connectionSettings

        const fileName = getFileName(connection)

        try {
            logger.info(
                `Backing up connection to database ${connection.database} on host ${
                    connection.host}`
            )

            const {
                stdout,
                stderr
            } = childProccess.exec(
                `PGHOST=${connection.host} PGPORT=${connection.port} PGDATABASE=${
                    connection.database} PGUSER=${connection.user} PGPASSWORD=${
                    connection.password} pg_dump --format=c`
            )

            const upload = filesService.uploadOne(stdout!, {
                title: fileName,
                type: "application/octet-stream",
                filename_download: fileName,
                storage: storage ?? "local",
                folder: folder ?? undefined,
            })
            const errors = checkStreamError(stderr)

            await Promise.race([upload, errors])

            logger.info(
                `[${pkg.name}] New database backup created: ${fileName}`
            )
```

```
52
53              const time = (performance.now() - start) / 1000
54              const size = (await filesService.readOne(await upload)).filesize
55
56              return {
57                  connection,
58                  time,
59                  size,
60                  backupId,
61              }
62          } catch (e) {
63              await filesService.deleteByQuery({
64                  filter: {
65                      title: {
66                          _eq: fileName,
67                      },
68                  },
69              })
70
71              logger.error(
72                  ` Error on database backup: ${
73                      (e as Error).message
74                  }`
75              )
76
77              throw {
78                  connection,
79                  backupId,
80                  error: message,
81              }
82          }
83      },
84  })
```

B – Interface para expressão cron

```
<template>
  <cron-light
    :locale="locale"
    :model-value="value"
    @update:model-value="handleChange"
  />
</template>

<script lang="ts">
import cronLight from "@vue-js-cron/light"
import "@vue-js-cron/light/dist/light.css"

import { defineComponent } from "vue"

export default defineComponent({
    props: {
        value: {
            type: String,
            default: null,
        },
    },
    emits: ["input"],
    components: { "cron-light": cronLight.component },
    setup(props, { emit }) {
        const locale = navigator.language.split("-")[0]

        function handleChange(value: string): void {
            emit("input", value)
        }

        return { locale, handleChange }
    },
})
</script>
```

C – Interface para listar backups

```
<template>
  <DataTable
    v-model:server-options="serverOptions"
    :server-itens-length="totalCount ?? 0"
    :loading="loading"
    :headers="headers"
    :itens="itens"
    table-class-name="customize-table"
    must-sort
    no-hover
  >
    <template #expand="item">
      <expandable-area
        :status="item.status"
        :error-message="item.error_message"
      />
    </template>

    <template #item-status="{ status }">
      <status-chip :status="status" />
    </template>
    <template #item-date_created="{ date_created }">
      {{ formatRelative(new Date(date_created), Date.now(), { locale }) }}
    </template>
    <template #item-size_in_bytes="{ size_in_bytes }">
      {{ Number(size_in_bytes) / 1000 }} Mb
    </template>
    <template #item-elapsed_time="{ elapsed_time }">
      {{ Number(elapsed_time).toFixed(2) }} sec
    </template>
  </DataTable>
</template>

<script lang="ts">
import { useitens } from "@directus/extensions-sdk"
import Vue3EasyDataTable from "vue3-easy-data-table"
import type { Header, ServerOptions } from "vue3-easy-data-table"
import "vue3-easy-data-table/dist/style.css"
import { formatRelative } from "date-fns"
import { ptBR, enUS } from "date-fns/locale"

import { defineComponent, ref, computed, watch } from "vue"

import StatusChip from "./components/status-chip.vue"
import ExpandableArea from "./components/expandable-area.vue"

export default defineComponent({
    props: {
        value: {
            type: String,
            default: null,
        },
        field: {
            type: String,
```

```
55              default: null,
56          },
57          primaryKey: {
58              type: String,
59              default: null,
60          },
61          limit: {
62              type: Number,
63              default: 15,
64          },
65      },
66      emits: ["input"],
67      components: { DataTable: Vue3EasyDataTable, StatusChip, ExpandableArea },
68      setup(props, { emit }) {
69          const serverOptions = ref<ServerOptions>({
70              page: 1,
71              rowsPerPage: props.limit,
72              sortBy: "date_created",
73              sortType: "desc",
74          })
75
76          const { itens, loading, totalCount } = useitens(ref(props.field), {
77              page: computed(() => serverOptions.value.page),
78              limit: computed(() => serverOptions.value.rowsPerPage),
79              sort: computed(() => {
80                  const type = serverOptions.value.sortType === "desc" ? "-" : ""
81                  return serverOptions.value.sortBy
82                      ? [type + serverOptions.value.sortBy]
83                      : null
84              }),
85              fields: ref([
86                  "status",
87                  "date_created",
88                  "size_in_bytes",
89                  "elapsed_time",
90                  "error_message",
91              ]),
92              filter: ref({
93                  connection: {
94                      id: {
95                          _eq: props.primaryKey === "+" ? -1 : props.primaryKey,
96                      },
97                  },
98              }),
99              search: ref(null),
100         })
101
102         const headers: Header[] = [
103             { text: "Status", value: "status", sortable: true },
104             { text: "Date Created", value: "date_created", sortable: true },
105             { text: "Size", value: "size_in_bytes", sortable: true },
106             { text: "Time", value: "elapsed_time", sortable: true },
107         ]
108
109         const locale = window.navigator.language === "pt-BR" ? ptBR : enUS
110
111         return {
112             serverOptions,
113             loading,
114             totalCount,
```

```
115             headers,
116             itens,
117             locale,
118             handleChange,
119             formatRelative,
120         }
121
122         function handleChange(value: string): void {
123             emit("input", value)
124         }
125     },
126 })
127 </script>
```

D – *Hook* para agendamento de backups

```
const tasks: Record<string, ScheduledTask> = {}

export default defineHook(
    async ({ action }, { services, logger, getSchema }) => {
        const schema = await getSchema()

        const FlowsService: FlowsService = new services.FlowsService({
            schema,
        })

        const ConnectionsService: itensService = new services.itensService(
            "connections",
            { schema }
        )

        const [{ id: flowId }]: any[] = await FlowsService.readByQuery({
            filter: { name: { _eq: "Trigger Backup for Connections" } },
        })

        const scheduleBackup = async (
            cron: string,
            directusUrl: string,
            flowId: string,
            collection: string,
            key: string
        ) => {
            if (key in tasks) {
                tasks[key]!.stop()
                delete tasks[key]
            }

            const data = await ConnectionsService.readOne(key)
            if (!data.active) {
                return
            }

            logger.info(
                `[schedule-backup-hook]: Scheduling backup for connection ${key} with
                    cron ${cron}`
            )

            tasks[key] = schedule(cron, async () => {
                const data = await ConnectionsService.readOne(key)
                if (!data.active) {
                    return
                }

                await axios.post(`${directusUrl}/flows/trigger/${flowId}`, {
                    collection,
                    keys: [key],
                })
            })
        }

```

```
54          const configureCron: ActionHandler = async (meta, context) => {
55              if (!meta.payload?.cron && !meta.payload?.active) {
56                  return
57              }
58
59              const { accountability } = context
60              const { collection, keys, payload } = meta
61
62              if (!payload?.cron) {
63                  // when toggle active status, cron payload is undefined
64                  const data = await ConnectionsService.readOne(keys[0])
65                  payload.cron = data.cron
66              }
67
68              const directusUrl = accountability?.origin!
69
70              scheduleBackup(
71                  payload.cron,
72                  directusUrl,
73                  flowId,
74                  collection,
75                  keys[0]
76              )
77          }
78
79          action("connections.itens.update", async (meta, context) => {
80              if (!("active" in meta.payload)) {
81                  meta.payload.active = (
82                      await ConnectionsService.readOne(meta.keys[0])
83                  ).active
84              }
85              if (!("cron" in meta.payload)) {
86                  meta.payload.active = (
87                      await ConnectionsService.readOne(meta.keys[0])
88                  ).cron
89              }
90
91              configureCron(meta, context)
92          })
93
94          action("connections.itens.create", (meta, context) => {
95              if (!("active" in meta.payload)) {
96                  meta.payload.active = true
97              }
98              if (!("cron" in meta.payload)) {
99                  meta.payload.cron = "* * * * *"
100             }
101             meta.keys = [meta.key]
102             configureCron(meta, context)
103         })
104
105         action("connections.itens.delete", (meta) => {
106             const key = meta.keys[0]
107             if ("key" in tasks) {
108                 tasks[key]!.stop()
109                 delete tasks[key]
110             }
111         })
112
113         action("server.start", async ({ server }) => {
```

```
114             const serverAddress = (
115                 server as http.Server
116             ).address() as AddressInfo
117             const directusUrl = `http://${serverAddress.address}:${serverAddress.port}`
118 
119             const connections = await ConnectionsService.readByQuery({
120                 filter: {
121                     cron: {
122                         _nnull: true,
123                     },
124                     active: {
125                         _neq: false,
126                     },
127                 },
128             })
129 
130             for (const { id, cron } of connections) {
131                 scheduleBackup(cron, directusUrl, flowId, "connections", id)
132             }
133         })
134     }
135 )
```