UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALEXANDRE JUNIOR DA COSTA

Uso de plataformas *no-code* para desenvolvimento de um aplicativo móvel em formato *MVP* para carregamento de veículos elétricos

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dra. Érika Fernandes Cota

Porto Alegre
2023

“Any sufficiently advanced bug is indistinguishable from a feature.”

— RICK KULAWEIC

AGRADECIMENTOS

Agradeço à minha mãe por ter me bancado e apoiado durante o início da minha demorada, mas valiosa, graduação.

Agradeço às empresas que me proporcionaram as diversas oportunidades para crescimento profissional e pessoal durante o hiato sem estudos.

E agradeço, claro, à minha esposa, que me aguentou durante o final movimentado e ocupado dessa fase da minha vida acadêmica, mesmo que em meio a diversas queixas de falta de atenção da minha parte.

RESUMO

Este trabalho utiliza um estudo de caso complexo de uma startup do ramo de carregamento de veículos elétricos para entender o quanto as plataformas *no-code* da atualidade estão preparadas para atender às necessidades de desenvolvimento de projetos MVP. Diversas plataformas são analisadas preliminarmente de modo a filtrar aquelas que atendem a requisitos amplos de escolha, entre eles a necessidade de a plataforma oferecer meios de desenvolvimento e publicação de aplicativos móveis. Coletados os requisitos das funcionalidades e recursos do aplicativo, novos requisitos para a escolha da plataforma são elencados a partir deles, e a plataforma AppGyver, da SAP, acaba sendo escolhida para o estudo. Diversos desafios aparecem ao longo do caminho, em meio a mudanças de requisitos e escopo, bem como na utilização da plataforma, que apresentou algumas carências importantes.

Palavras-chave: Plataformas no-code. Aplicativos móveis. MVP. Veículos elétricos. AppGyver.

The use of no-code platforms for the development of an MVP mobile app for electric vehicle charging

ABSTRACT

This work uses a complex case study of a startup in the field of electric vehicles charging to understand how much the current no-code platforms are prepared to meet the needs of developing MVP projects. Several platforms are preliminarily analyzed in order to filter out those that meet broad choice requirements, including the need for the platform to offer means of developing and publishing mobile applications. Having collected the requirements for the application's functionalities and features, new requirements for choosing the platform are listed from them, and the AppGyver platform, from SAP, ends up being chosen for the study. Several challenges appear along the way, amidst changes in requirements and scope, as well as in the use of the platform, which presented some important shortcomings.

Keywords: No-code platforms. Mobile app. MVP. Electric vehicles. AppGyver.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CPMS	Charge Point Management System
CSS	Cascading Style Sheets
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
MVP	Minimum Viable Product
OCPP	Open Charge Point Protocol
REST	Representational State Transfer
SAVE	South America Vehicle Electrification (empresa)
SDK	Software Development Kit
VE	Veículo Elétrico
URL	Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO

O empreendedorismo tem se mostrado uma força motriz para a inovação e o crescimento econômico em todo o mundo. O MVP (*Minimum Viable Product*), ou Produto Mínimo Viável, é uma abordagem de desenvolvimento de produtos que enfoca a criação e lançamento de uma versão inicial simplificada de um produto ou serviço, com o objetivo de obter retorno dos usuários e validar hipóteses antes de investir recursos significativos no desenvolvimento completo.

A principal ideia por trás do MVP é oferecer apenas as funcionalidades básicas e essenciais que atendam às necessidades dos usuários iniciais. Isso permite que os empreendedores testem a aceitação e o interesse do mercado em relação à sua ideia, além de identificar pontos de melhoria e ajustes necessários. É construído de forma ágil e rápida, visando uma versão mínima, porém viável, do produto (RIES, 2011).

A importância dessa abordagem para novos empreendedores é significativa. Primeiro, ele ajuda a reduzir o risco e a incerteza associados ao desenvolvimento de um novo produto ou serviço. Ao lançar uma versão inicial com investimentos mínimos, os empreendedores podem testar suas suposições e entender melhor as necessidades e preferências dos clientes antes de se comprometerem com um desenvolvimento completo.

Além disso, o MVP permite uma interação direta com os usuários, permitindo a coleta de retorno valioso. Essas informações são essenciais para aprimorar o produto ou serviço, identificar lacunas e agregar valor real aos clientes. Com base no retorno, os empreendedores podem tomar decisões informadas sobre os recursos a serem adicionados, removidos ou ajustados, direcionando seus esforços para as áreas que realmente importam para os usuários. Por isso, ajuda a economizar tempo e recursos financeiros preciosos. Ao invés de gastar meses ou anos desenvolvendo um produto completo, os empreendedores podem obter uma versão inicial no mercado rapidamente, testando a viabilidade e a aceitação antes de se comprometerem totalmente. Isso reduz o risco de lançar um produto sem demanda e permite uma iteração mais ágil e eficiente.

Apesar da importância do MVP, muitos empreendedores enfrentam desafios significativos ao tentar implementá-lo. Muitas vezes, esses empreendedores possuem conhecimento especializado em um determinado setor de negócios, mas carecem de habilidades técnicas para desenvolver um aplicativo ou produto digital por conta própria. A contratação de desenvolvedores qualificados pode ser uma opção, mas os custos envolvidos podem ser proibitivos para empreendedores com recursos financeiros limitados.

Para esses empreendedores, as plataformas *no-code* surgem como uma solução promissora. Essas plataformas prometem habilitar pessoas sem conhecimento de programação a desenvolver aplicativos e produtos digitais sem escrever código.

As plataformas *no-code* oferecem interfaces visuais e ferramentas de arrastar e soltar, permitindo que cidadãos-desenvolvedores, pessoas sem formação em programação, construam protótipos funcionais e MVPs de forma rápida e acessível. Com essas ferramentas, é possível criar interfaces interativas, definir fluxos de trabalho, integrar serviços e até mesmo implementar lógica personalizada, tudo sem a necessidade de escrever em uma linguagem de programação específica.

Ao usar plataformas *no-code*, os empreendedores podem economizar tempo e dinheiro, já que não precisam contratar desenvolvedores ou depender de terceiros para a implementação do MVP. Isso permite que eles assumam o controle do processo de desenvolvimento e realizem iterações rápidas, conforme necessário, para validar sua ideia de negócio.

Além disso, as plataformas *no-code* prometem ter uma curva de aprendizado mais suave, permitindo que os cidadãos-desenvolvedores foquem em sua expertise de negócios em vez de investir tempo e esforço no aprendizado de linguagens de programação complexas. Isso é especialmente valioso para empreendedores com conhecimento especializado em um setor específico, mas que não possuem experiência em desenvolvimento de software.

Existem centenas de opções de plataformas *no-code* disponíveis (G2, 2023), entre elas, algumas famosas, como a Bubble, OutSystems, Zapier, Salesforce, Appy Pie, Shopify e Wix. Cada uma delas tem suas próprias características e recursos, oferecendo flexibilidade para atender às necessidades específicas de diferentes tipos ou classes de aplicações.

Em resumo, as plataformas *no-code* prometem uma solução acessível e eficaz para empreendedores que desejam implementar um MVP, mas que enfrentam desafios técnicos ou financeiros. Essas ferramentas visam capacitar os empreendedores a criar protótipos e produtos digitais funcionalmente viáveis, permitindo que eles testem e validem suas ideias no mercado de forma ágil e econômica.

O presente trabalho tem como objetivo compreender o alcance e as limitações do uso de plataformas *no-code* como ferramentas de criação de MVPs, por meio de um estudo de caso realizado com empreendedores da empresa SAVE (*South America Vehicle Electrification*, ou, pelo nome fantasia, Serviços de Abastecimento de Veículos Elétricos).

A SAVE é uma empresa incubada no Centro de Empreendimentos em Informática (CEI) da UFRGS e precisa implementar um aplicativo móvel que permita aos clientes utilizar sua infraestrutura de carregadores veiculares para efetuar a carga de seus automóveis e realizar o pagamento. O alinhamento do desenvolvimento de software com a estratégia da empresa foi um dos critérios da seleção da startup para incubação no CEI.

Considerando os desafios técnicos enfrentados pelos empreendedores, a utilização de plataformas *no-code* foi cogitada como uma solução viável para o desenvolvimento do MVP desejado.

Nesse contexto, o estudo de caso realizado com a SAVE busca analisar a eficácia e a praticidade do uso de uma plataforma *no-code* específica na criação do MVP do aplicativo de carregamento de veículos elétricos. Serão avaliados aspectos como a facilidade de criação das funcionalidades essenciais do aplicativo, a adaptabilidade às necessidades específicas da empresa e a agilidade no desenvolvimento.

Ao compreender o alcance e as limitações das plataformas *no-code* para o desenvolvimento de MVPs, este estudo de caso visa fornecer *insights* valiosos para os empreendedores da SAVE e para outros empreendedores que estejam enfrentando desafios semelhantes. Pretende-se, com este trabalho, contribuir-se para a tomada de decisões informadas sobre a utilização dessas plataformas como uma alternativa viável e acessível para o desenvolvimento de produtos digitais iniciais

Este trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta uma fundamentação teórica dos conceitos utilizados ao longo do estudo. O capítulo 3 apresenta uma análise inicial para a escolha de uma plataforma *no-code* para desenvolvimento do MVP, uma análise do estudo de caso da SAVE e a efetiva escolha e análise da plataforma de estudo. O Capítulo 4 consolida os requisitos do aplicativo móvel e define a arquitetura da solução, ajustando o escopo para se adaptar a desafios que surgiram durante o estudo. No Capítulo 5, são apresentados os resultados dos tempos de aprendizado e desenvolvimento dos diversos módulos da plataforma e páginas do aplicativo. No Capítulo 6, considerações finais são feitas sobre o estudo e a plataforma escolhida.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, adentramos em um universo de conhecimento essencial para a compreensão e análise do tema central deste trabalho. Aqui, exploraremos as bases conceituais e teóricas que sustentam nossa pesquisa, fornecendo um alicerce sólido para a investigação que se segue. Este capítulo é fundamental para estabelecer o contexto, definir os principais conceitos, revisar a literatura relevante e delinear os principais argumentos que sustentarão nosso estudo, permitindo aos leitores uma compreensão mais profunda e embasada sobre o assunto em questão.

2.1 O MVP e seu papel na criação de negócios

A indústria de software tem observado uma tendência crescente de produtos de software sendo construídos por startups de software. As startups de software são muitas vezes organizações recém-fundadas com pouca ou nenhuma experiência operacional que visam desenvolver produtos de software de alto crescimento. Ao contrário de empresas mais estabelecidas, as startups muitas vezes concentram seus esforços no desenvolvimento e lançamento de um produto que forneça valor real para seu público-alvo (PATERNOSTER et al., 2014). As abordagens metodológicas recentes para o desenvolvimento de produtos de startup, como o *Lean Startup* (RIES, 2011) ou os processos de desenvolvimento de novos produtos (COLEMAN; O'CONNOR, 2008), enfatizam a capacidade de aprender sobre os problemas reais dos primeiros clientes e a taxa de aprendizagem.

Cada novo empreendimento de negócios deve inicialmente se concentrar no desenvolvimento de um "Produto Mínimo Viável", ou MVP, e depois usar este produto para testar sua hipótese sobre os requisitos de seu mercado-alvo. O desenvolvimento de MVPs, que são produtos que só têm recursos suficientes para reunir conhecimento validado sobre as mercadorias, é um foco primário nas fases iniciais do desenvolvimento do produto. Ele não só desempenha uma função significativa para a equipe que trabalha no negócio, mas também tem um papel importante para as partes interessadas externas que trabalham com a startup, como potenciais usuários, investidores e mentores. Hoje em dia, o MVP é um artefato crucial que deve ser mostrado durante uma reunião com um investidor. Existem vários tipos distintos de MVPs, cada um dos quais varia em termos de esforços de desenvolvimento envolvidos, as razões pelas quais servem e as etapas em que ocorrem com frequência (RIES, 2011).

O conceito enfatiza, portanto, a criação de produtos com o mínimo de recursos necessários para validar hipóteses, aprender com os clientes e obter retorno relevante do mercado. A ideia é evitar o desenvolvimento excessivo de um produto antes de saber se ele realmente atende às necessidades do público-alvo.

Há várias razões pelas quais o desenvolvimento de MVPs se tornou um foco primário nas fases iniciais do desenvolvimento de produtos para startups:

- **Aprendizado rápido**: ao criar um MVP, as startups podem lançar uma versão inicial do produto no mercado mais rapidamente. Isso permite que elas obtenham retorno real dos primeiros clientes e aprendam sobre os problemas e necessidades reais que o produto deve resolver.
- **Redução de riscos**: desenvolver um produto completo pode ser caro e demorado. Com um MVP, as startups podem minimizar os riscos financeiros, pois investem menos recursos no início do processo de desenvolvimento.
- **Validação de ideias**: o MVP é uma ferramenta para validar as ideias e hipóteses da startup sobre o mercado e os clientes. Se o MVP for bem recebido, isso pode indicar que a startup está no caminho certo; caso contrário, ela pode pivotar ou ajustar sua estratégia.
- **Engajamento com investidores e *stakeholders***: mostrar um MVP funcional é uma maneira eficaz de envolver investidores, parceiros e outros *stakeholders*. Ele permite que eles visualizem a proposta de valor do produto e o potencial do negócio.
- **Economia de recursos**: ao evitar o desenvolvimento de recursos desnecessários no estágio inicial, as startups economizam tempo e dinheiro. Isso é especialmente valioso, considerando que muitas startups têm recursos limitados.

Os MVPs podem assumir diferentes formas e níveis de complexidade, dependendo do produto e do mercado em questão. Alguns exemplos de MVPs de produtos de software incluem (ALTAF, 2022):

- ***Landing Page***: Uma página simples na web que descreve o produto ou serviço e coleta informações de interesse dos potenciais clientes. Isso ajuda a medir o interesse inicial no produto.
- **Protótipo funcional de recurso único**: Um MVP que possui apenas uma funcionalidade essencial, demonstrando o núcleo do valor do produto sem desenvolver todos os recursos planejados.
- ***Mockups* interativo**: Uma representação visual do produto que simula sua funcio-

nalidade, mas sem a implementação completa.

- **Bastidor oculto**: Uma versão do produto com foco em utilizar o tempo desenvolvendo-se um a interface sofisticada, mas que nos bastidores funciona de forma manual. Por exemplo, um bonito aplicativo de reserva de estadias que nada mais faz que enviar um e-mail à pessoa responsável pelas reservas, que, então, a processará manualmente.

A escolha do tipo de MVP a ser desenvolvido dependerá das metas específicas da startup, das restrições de recursos e do estágio de desenvolvimento do produto. Independentemente disso, o objetivo principal é obter o valioso retorno dos usuários para iterar e melhorar o produto, garantindo que ele seja bem-sucedido no mercado.

2.2 Plataformas *No-code/Low-code*

A Gartner tem acompanhado, ano a ano, a evolução do mercado de tecnologias de desenvolvimento *low-code*. Suas projeções indicam um crescimento de 25% para as plataformas de desenvolvimento de aplicações *low-code* (*Low-Code Application Platforms*) para o ano de 2023, alcançando quase US$ 10 bilhões em valor de mercado (STAMFORD, 2022).

As plataformas de desenvolvimento de aplicações *low-code* e *no-code* são um tipo de plataforma de desenvolvimento de software que prometem que os usuários possam criar aplicativos escrevendo pouco (*low-code*) ou nenhum código (*no-code*). Ambas se utilizam de ferramentas como interfaces gráficas com recursos de arrastar e soltar para possibilitar que o usuário consiga criar o leiaute e a lógica de seus aplicativos. As plataformas de *low-code*, além disso, oferecem meios de ampliar as possibilidades previstas pela plataforma com componentes que podem ser modificados com alguma linguagem de programação. Já as plataformas *no-code* não oferecem essa possibilidade, tendo menos possibilidades de personalização. No entanto, apesar de serem menos personalizáveis, as plataformas *no-code* acabam por ser mais simples de utilizar (BOBOYOROVA, 2023).

A promessa, em geral, é de uma vasta gama de benefícios, incluindo tempos de desenvolvimento mais curtos, custos mais baixos e maior agilidade. Essas plataformas procuram eliminar a necessidade de codificação, a fim de tornar mais fácil para as empresas criar aplicativos rapidamente e sem dificuldade. Isso pode permitir que as empresas respondam mais rapidamente às necessidades em constante mudança dos seus clientes.

As plataformas podem ser usadas para construir uma ampla gama de aplicações, desde aplicações baseadas na web até aplicações móveis. Além disso, essas plataformas podem ser utilizadas para automatizar operações corporativas, como análise de dados, gestão de documentos, integração de sistemas e gestão de relacionamento com o cliente.

Quanto mais genérica a plataforma, mais complexa acaba ficando sua utilização, devido à gama de funções e recursos que devem atender a diversos escopos. Por isso, é comum encontrar plataformas especializadas em uma área específica. Reduzindo o escopo, é possível oferecer ferramentas abrangentes para o nicho, com menor complexidade de uso da plataforma.

Nem tudo são flores, entretanto. Embora essas plataformas tornem o projeto de aplicativos mais simples, eles podem não ser tão personalizáveis ou escaláveis como outras opções. Além disso, como são feitas para desenvolvimento rápido, pode não haver garantia de qualidade suficiente ou precauções de segurança. Portanto, é crucial tomar as precauções de segurança necessárias para proteger seus aplicativos. Isso inclui a configuração de autenticação e autorização do usuário, bem como a criação de regras e procedimentos de segurança para proteger dados sensíveis. Além disso, é crucial examinar e atualizar rotineiramente seus aplicativos para garantir que eles estejam atualizados com as mais recentes proteções de segurança (HUGHES, 2022).

2.3 Conceitos de arquitetura e desenvolvimento de software

Nesta subseção, iremos explorar os fundamentos essenciais que permeiam a arquitetura de aplicações baseadas na web e aplicativos móveis, bem como as estratégias de comunicação entre o *frontend* e o *backend* dessas soluções. Além disso, abordaremos a peculiaridade da comunicação com carregadores veiculares, um elemento crucial em nossa pesquisa. Esta subseção servirá como um embasamento conceitual sólido para a compreensão e análise posterior das tecnologias e abordagens específicas que serão empregadas em nosso estudo, estabelecendo um alicerce robusto para a avaliação crítica das soluções propostas.

2.3.1 Arquitetura de aplicações baseadas na web

As aplicações web modernas são complexas e multifacetadas, com funcionalidades e interações altamente sofisticadas destinadas a fornecer uma experiência de usuário atraente e eficiente. O padrão *Backend For Frontend* (BFF) divide as responsabilidades das aplicações em dois componentes de alto nível: o *frontend* e o *backend* (WICKRAMARACHCHI, 2021).

O *frontend* refere-se à interface com a qual os usuários interagem diretamente. É a face visível da aplicação, onde os elementos gráficos e interativos são exibidos para o usuário final. O HTML (do inglês, *HyperText Markup Language*), o CSS (do inglês, *Cascading Style Sheets*) e o JavaScript são as principais tecnologias utilizadas no *frontend* para estruturar e estilizar a página web, bem como adicionar dinamismo e interatividade (CIAT, 2022).

As principais responsabilidades do *frontend* incluem a apresentação adequada do conteúdo, a coleta de entradas do usuário e a comunicação com o *backend* para troca de dados por meio de Interfaces de Programação de Aplicativos (do inglês, *Application Programming Interface* — API). Além disso, é responsabilidade do *frontend* garantir a experiência do usuário, com ênfase na responsividade, usabilidade e carregamento eficiente da página.

O *backend*, por outro lado, é a parte do sistema que opera nos bastidores, sem ser visível para o usuário. Ele lida com o processamento dos dados, a lógica de negócio e a interação com bancos de dados e outras infraestruturas de suporte. As tecnologias de *backend* são tipicamente linguagens de programação do lado do servidor, como Python, Java, PHP, Ruby, entre outras.

As principais responsabilidades do *backend* incluem o processamento e armazenamento de dados da aplicação, a aplicação de regras de negócio, a autenticação e autorização de usuários, e o gerenciamento da segurança e proteção contra vulnerabilidades. O *backend* também é responsável por se comunicar com serviços externos, quando necessário, como serviços de pagamento ou integrações com mídias sociais.

Olhando para o lado dos aplicativos móveis, essa diferença fica explicitada naqueles que são desenvolvidos com a finalidade de serem pontos de acesso a algum serviço baseado na web. O aplicativo móvel atua como o *frontend* desse serviço, e é necessário haver algum meio de comunicação com o *backend*, que fica nos servidores da entidade responsável pela manutenção do serviço.

2.3.2 Arquitetura de aplicativos móveis

Aplicativos móveis são, geralmente, desenvolvidos de 3 formas: nativa, web e híbrida (GRIFFITH, 2023). O aplicativo móvel nativo é aquele que realmente é executado no dispositivo, e deve ser modificado para operar em diferentes dispositivos-alvo. Por ser uma instalação nativa, obtém-se melhor performance, fácil acesso aos recursos de hardware do dispositivo, e fica disponível para download em lojas de aplicativos específicas da plataforma. O desenvolvimento de um aplicativo nativo para várias plataformas requer conhecimento das linguagens de programação, APIs e Kits de Desenvolvimento de Software (do inglês, *Software Development Kits* — SDKs) de cada plataforma de destino. Isso requer um esforço de desenvolvimento maior, pois exige equipe capacitada para diversas linguagens de desenvolvimento e, basicamente, tempo para desenvolver o mesmo aplicativo várias vezes. Outro ponto a se considerar é a adequação do aplicativo às diversas e diferentes regras de cada loja de aplicativos onde se deseja disponibilizá-lo.

A segunda abordagem é um aplicativo da web para dispositivos móveis, que deve ser hospedado em servidores da web e acessado por meio de navegadores da web para dispositivos móveis. São, portanto, altamente portáveis para uma ampla variedade de sistemas operacionais móveis. Essa estratégia permite que o programa seja executado em várias plataformas, o que acaba reduzindo o tempo e o custo necessários para o desenvolvimento. Contudo, os aplicativos móveis web muitas vezes não podem utilizar recursos de hardware específicos do dispositivo, como câmera ou acelerômetro. São comparáveis a um *website* que foi projetado especificamente para uso em dispositivos móveis e não é preciso se preocupar com regras externas ao próprio projeto.

Os aplicativos móveis híbridos, a terceira e última técnica, são um misto das soluções supracitadas. O aplicativo é desenvolvido em linguagem web (HTML, CSS e JavaScript), e, ao invés de rodar dentro do navegador padrão do usuário, uma instância invisível do navegador é encapsulada em um aplicativo nativo. Essas instâncias invisíveis são componentes disponíveis nos SDKs de cada plataforma. Para uma aplicação Android, por exemplo, o elemento WebView desempenha essa função, enquanto no iOS seria o WKWebView. Adiante no texto, o termo *webview* será usado para referir-se a todas as semelhantes tecnologias de encapsulamento. Embora seja necessário seguir as regras das lojas de aplicativos e o desempenho ainda seja limitado por estar rodando dentro de um navegador, é possível reutilizar o mesmo código para compilar o aplicativo para várias plataformas e ainda assim usufruir dos recursos de hardware do dispositivo. Outra vanta-

gem é que existem tecnologias que permitem embutir e renderizar o código da interface diretamente no aparelho, não sendo preciso manter um servidor de hospedagem web para ela, como é o caso dos aplicativos móveis web.

Em todos os casos, os aplicativos que necessitam trabalhar com informações disponíveis via Internet estão usando arquitetura BFF em seu mais alto nível.

2.3.3 Comunicação entre o *frontend* e o *backend*

O desenvolvimento de sistemas distribuídos tornou-se uma prática comum para atender às crescentes demandas da era digital. Nesse contexto, as APIs desempenham um papel fundamental, possibilitando a interação entre diferentes aplicações e serviços. Uma das abordagens mais adotadas para projetar APIs é o REST (*Representational State Transfer*), comumente em conjunto com o formato JSON (*JavaScript Object Notation*), que utiliza o protocolo HTTP (*Hypertext Transfer Protocol*) para enviar e receber requisições.

2.3.3.1 REST

O REST é um estilo de arquitetura de software que visa fornecer padrões entre sistemas de computador na web, facilitando a comunicação entre eles. Os sistemas compatíveis com REST, geralmente chamados de sistemas *RESTful*, são caracterizados por serem *stateless* (sem estado) e por separar as preocupações do cliente e do servidor (CODECADEMY, 2023).

2.3.3.2 JSON

O JSON é um formato de dados amplamente usado para representar e transmitir informações em aplicações e sistemas (CROCKFORD; MORNINGSTAR, 2017). Ele é baseado em pares chave-valor, proporcionando uma sintaxe concisa e eficiente, resultando em arquivos menores e processamento mais rápido, tornando-o ideal para aplicações web e APIs REST. Sua popularidade crescente é devido à sua simplicidade e compatibilidade com diversas linguagens de programação.

2.3.3.3 HTTP

O HTTP é um protocolo de camada de aplicação no modelo de conjunto de protocolos da Internet para sistemas de informação de hipermídia distribuídos e colaborativos (IETF, 2022). HTTP é a base da comunicação de dados para a *World Wide Web*, onde os documentos de hipertexto incluem *hyperlinks* para outros recursos que o usuário pode acessar facilmente, por exemplo, por um clique do mouse ou tocando na tela em um navegador da web.

O HTTP usa "campos" para fornecer dados na forma de pares nome/valor extensíveis com um *namespace* de chave registrado. Os campos são enviados e recebidos nas seções de cabeçalho e trailer das mensagens. Os campos que são enviados ou recebidos antes do conteúdo são chamados de "campos de cabeçalho" (ou apenas "cabeçalhos", coloquialmente).

A "seção de cabeçalho" de uma mensagem consiste em uma sequência de linhas de campo de cabeçalho. Cada campo de cabeçalho pode modificar ou estender a semântica da mensagem, descrever o remetente, definir o conteúdo ou fornecer contexto adicional.

O campo de cabeçalho "*Authorization*" permite que um agente de usuário se autentique com um servidor de origem. Seu valor consiste em credenciais contendo as informações de autenticação do agente do usuário para o domínio do recurso que está sendo solicitado.

2.3.3.4 URL

Uma URL (do inglês, *Uniform Resource Locator*), coloquialmente chamado de endereço da web, é uma referência a um recurso da web que especifica sua localização em uma rede de computadores e um mecanismo para recuperá-lo. Uma URL é um subconjunto de URI (do inglês, *Uniform Resource Identifier*) — uma sequência compacta de caracteres que identificam um recurso abstrato ou físico (IETF, 2005) — embora muitas pessoas usem os dois termos de forma intercambiável.

2.3.3.5 API REST

As APIs REST são baseadas no conceito de arquitetura REST, que se concentra na simplicidade e no uso de recursos padrão da web, como URLs e métodos HTTP (REDHAT, 2023). A finalidade principal das APIs REST é fornecer uma maneira padro-

nizada de interagir com serviços web de forma eficiente e escalável. Algumas características essenciais das APIs REST incluem:

- ***Stateless***: cada requisição de cliente para servidor deve conter todas as informações necessárias para compreender e processar a solicitação, tornando o servidor independente das requisições anteriores.
- **Utilização de Métodos HTTP**: as operações básicas do protocolo HTTP, como GET, POST, PUT e DELETE, são mapeadas para as ações que a API deve realizar nos recursos.
- **Recursos Identificados por URLs**: cada recurso da API é acessado por um URL exclusivo, permitindo que os clientes interajam com esses recursos usando os métodos HTTP apropriados.

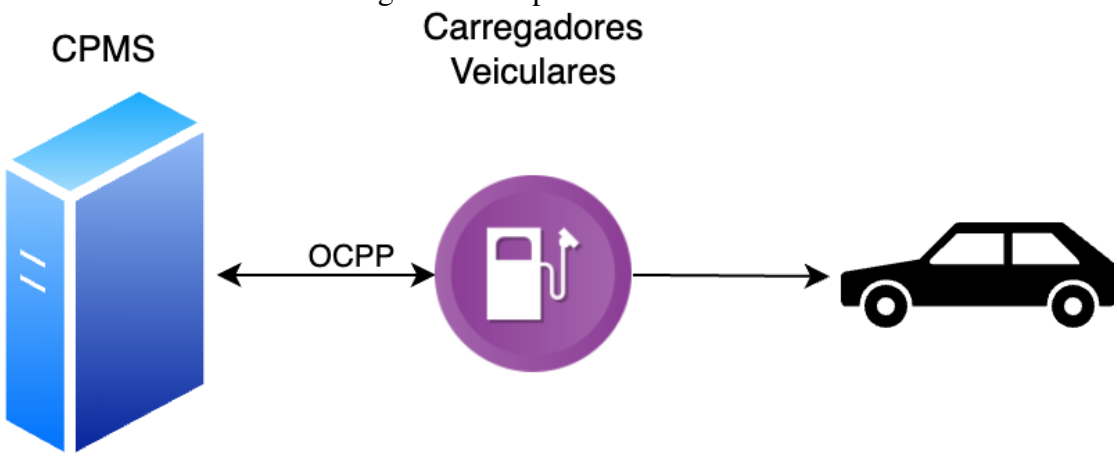
APIs REST geralmente implementam táticas de negação de serviço, como *rate limiting* e *throttling*, para controlar o tráfego, prevenir abusos e garantir uma distribuição equitativa dos recursos. Isso ajuda a evitar sobrecargas nos servidores, assegurar uma experiência estável para os usuários, impedir que um único usuário monopolize os recursos da API e cumprir contratos de nível de serviço. Essas táticas também contribuem para a previsibilidade do desempenho da API, a conservação de recursos e a redução de custos operacionais (TIBCO, 2023).

2.3.4 Comunicação com carregadores veiculares

O OCPP (*Open Charge Point Protocol*) é um protocolo de comunicação aberto e padronizado, desenvolvido especificamente para permitir a comunicação entre sistemas de gerenciamento de carga de veículos elétricos e pontos de carga (*charge points*), que são os carregadores em si. Ele foi projetado para facilitar a interoperabilidade e a comunicação eficiente entre diferentes equipamentos e fornecedores envolvidos no carregamento de veículos elétricos (WEVO, 2022).

A relação entre o OCPP e o uso de CPMS (*Charge Point Management System*), às vezes também referenciado como CSMS (*Charging Station Management System*) é fundamental para o funcionamento e a gestão das redes de carregamento de veículos elétricos. O CPMS é o software centralizado responsável por controlar, monitorar e gerenciar todas as estações de carregamento em uma determinada rede, que podem ter múltiplos pontos de carga em cada uma. Ele atua como um sistema de gerenciamento central, fornecendo

Figura 2.1: O protocolo OCPP



Fonte: O Autor

aos operadores uma visão completa do status e desempenho de todas as estações de carregamento conectadas e os estados atuais de seus pontos de carga (CURRENT, 2022).

O protocolo OCPP em pontos de carga permite que eles se comuniquem com o CPMS de forma bidirecional e troquem informações vitais, como status de carregamento, autenticação de usuários, gerenciamento de tarifas, diagnóstico de falhas e relatórios de utilização. Ele define a estrutura e os procedimentos para a troca dessas informações, permitindo que diferentes fabricantes de estações de carregamento e fornecedores de software de gerenciamento trabalhem em conjunto de forma padronizada.

Dessa forma, o OCPP desempenha um papel crítico na interoperabilidade de sistemas e na criação de uma infraestrutura de carregamento de veículos elétricos mais unificada e eficiente. Ele garante que as estações de carregamento sejam compatíveis com o CPMS e outros sistemas relacionados, independentemente do fornecedor, o que é essencial para o desenvolvimento e expansão bem-sucedidos da mobilidade elétrica. Além disso, o uso do OCPP também contribui para a segurança, eficiência e facilidade de uso de toda a rede de carregamento, oferecendo uma experiência consistente aos usuários finais de veículos elétricos.

Existem diversos fornecedores de plataformas que integram um CMPS a outras funcionalidades e recursos relacionados ao carregamento dos veículos, como formas de pagamento ao usuário final, sistema de autenticação e cadastro, relatórios gerenciais e reserva de equipamentos. No entanto, como o OCPP é aberto, qualquer entidade que necessitar utilizá-lo está livre para desenvolver toda a infraestrutura por conta própria.

3 ANÁLISE PRELIMINAR

A escolha de uma plataforma *no-code* pode ter um impacto significativo para um cidadão-desenvolvedor e para o projeto em que ele está trabalhando. Plataformas diferentes oferecem conjuntos variados de recursos e capacidades. Escolher uma plataforma que não atenda às necessidades específicas do projeto pode resultar em limitações funcionais, levando a compromissos na funcionalidade desejada.

Se a plataforma escolhida não for capaz de lidar com o crescimento do projeto, podem surgir problemas de escalabilidade. Conforme o produto ganha visibilidade e aumenta seu número de usuários, isso pode levar a lentidão, falhas ou até mesmo a necessidade de reescrever o projeto em uma plataforma mais robusta, o que consome tempo e recursos.

Algumas plataformas podem não permitir uma manutenção adequada ou a evolução contínua do projeto. Isso pode levar a problemas de longo prazo, especialmente quando atualizações ou novos recursos são necessários.

Outras podem utilizar formatos de dados ou tecnologias proprietárias, tornando difícil ou caro migrar para outras plataformas no futuro. Isso pode deixar o projeto preso à plataforma original, limitando a flexibilidade e a escolha futura.

Os aplicativos resultantes podem ser menos eficientes em termos de desempenho, comparadas a desenvolvimento tradicional. Isso pode resultar em aplicativos mais lentos e menos responsivos.

Personalização do aplicativo é oferecida até certo ponto. Se o projeto exigir customizações mais complexas ou específicas, a plataforma pode não ser capaz de atender às demandas.

Nem todas as plataformas são igualmente robustas em termos de segurança. Escolher uma plataforma com vulnerabilidades de segurança pode resultar em violações de dados ou exposição a ataques.

Algumas plataformas podem parecer acessíveis no início, mas à medida que o projeto cresce e mais recursos são necessários, os custos podem aumentar substancialmente. Isso pode pegar desprevenidos os desenvolvedores que não anteciparam esses custos escalonados.

Se a plataforma escolhida exigir uma curva de aprendizado íngreme, pode levar mais tempo para o cidadão-desenvolvedor se tornar produtivo. Além disso, se a plataforma não se adaptar bem às habilidades e experiência do desenvolvedor, isso pode levar

a frustração e atrasos.

Certas plataformas podem ter melhor reputação do que outras, juntamente com um suporte mais sólido da comunidade ou da empresa. Escolher uma plataforma menos conhecida ou com suporte insuficiente pode resultar em dificuldades ao enfrentar problemas ou desafios.

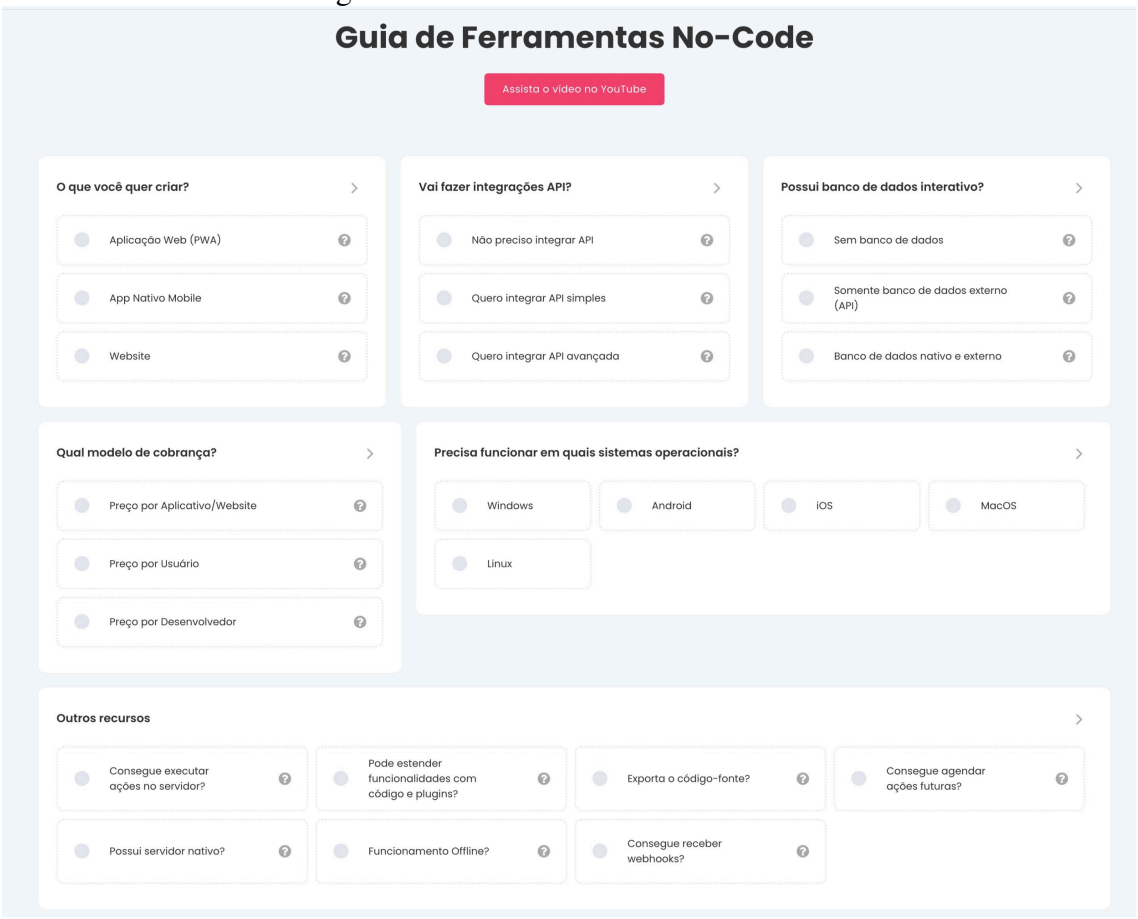
Ou seja, a escolha equivocada de uma plataforma *low-code* ou *no-code* pode resultar em limitações funcionais, problemas de escalabilidade, dificuldades de manutenção, bloqueio de plataforma, questões de desempenho, falta de personalização, problemas de segurança, custos inesperados e desafios de aprendizado. Portanto, é importante realizar uma pesquisa detalhada, avaliar as necessidades do projeto e considerar cuidadosamente os prós e contras de cada plataforma antes de tomar uma decisão.

Isso mostra que a escolha de uma plataforma pode não ser trivial para o cidadão-desenvolvedor. A base de conhecimento para essa tomada de decisão é composta por publicações em blogs e reportagens que, muitas vezes, listam requisitos técnicos das plataformas e/ou não oferecem informações suficientes para diferenciar uma da outra. A simples pesquisa por plataformas que permitem, especificamente, criar aplicativos nativos para Android e iOS pode ser difícil e confusa. A informação, em alguns casos, é omitida ou parcial. Por exemplo, algumas plataformas podem informar que permitem desenvolver aplicativos compatíveis com Android e iOS, mas ocultam o detalhe de que são, na verdade, aplicativos da web, ao invés de nativos ou híbridos. Um tempo considerável pode ser perdido do início da utilização da plataforma até a realização de que não é possível gerar o aplicativo para publicação em *marketplaces*.

Portanto, ao escolher-se uma plataforma *no-code* para criar um aplicativo móvel para um MVP, deve-se analisar algumas características básicas importantes para garantir que necessidades específicas sejam atendidas, de acordo com a realidade do produto e do negócio e com os objetivos do MVP.

- **Facilidade de Uso**: a plataforma é intuitiva e fácil de usar, com uma interface amigável que permita aos fundadores criar o aplicativo com facilidade, mesmo sem experiência de programação?
- **Publicação**: a plataforma permite o desenvolvimento de aplicativos móveis para iOS e Android? É possível que a startup publique-o nas lojas de aplicativos, de forma que seja possível alcançar uma base de usuários mais ampla?
- **Suporte e Comunidade**: a plataforma oferece suporte ao usuário, tutoriais, documentação e uma comunidade ativa, onde os fundadores podem buscar ajuda e

Figura 3.1: Guia de ferramentas *no-code*



Fonte: (CODAR, 2023)

compartilhar experiências?

- **Identidade visual**: a plataforma permite a customização do aplicativo de forma a integrá-lo à identidade visual da empresa, como paletas de cores e logotipo?
- **Custo**: os planos de preços oferecidos pela plataforma que cobrem todas as características desejadas se encaixa no orçamento inicial da startup?

Essa primeira análise deve levar em conta os objetivos do MVP em questão, mas também um planejamento de curto a médio prazo do negócio, uma vez que se espera uma sequência de MVPs à medida que o negócio evolui.

A busca de plataformas que satisfaçam todas as necessidades de um projeto pode ser bastante trabalhosa, como analisado no início deste capítulo. Algumas fontes relevantes de plataformas, como (GARTNER, 2023) e (G2, 2023), apresentam listas com centenas de opções, mas sem opções de filtragem efetivas e úteis, o que complica a busca, visto que essas listas incluem plataformas de todos os escopos possíveis.

Felizmente, em pesquisa na web, foi encontrado um guia de plataformas *no-code*

em (CODAR, 2023), mostrado na Figura 3.1 . O guia possibilita marcar as funções e recursos que são desejadas em uma plataforma, e mostra aquelas que atendem a todos os requisitos marcados.

Para o caso da SAVE e a partir da análise dos critérios preliminares, os seguintes filtros foram marcados: "App Nativo Mobile", "Quero integrar API simples"e filtros de sistema operacionais "Android"e "iOS".

Na época da consulta nesse guia, as seguintes plataformas foram sugeridas: Adalo, AppGyver, FlutterFlow, Google AppSheet, Mendix, Microsoft Power Apps, Outsystems, Thunkable e Zoho Creator.

A partir desse ponto, uma análise de cada plataforma foi realizada para elucidar os demais requisitos de seleção. Os resultados obtidos estão apresentados na Tabela 3.1.

Os custos mensais estão representados em dólares americanos, e consideram o valor do plano de cada plataforma que atende, minimamente, aos critérios estabelecidos.

Os critério para classificação da facilidade de uso de cada plataforma engloba a familiaridade dos elementos gráficos da interface com outros elementos comuns em sistemas operacionais e outros aplicativos, recursos intuitivos de arrastar e soltar para criação da lógica e da interface, necessidade ou não de instalar aplicativos de computador para desenvolvimento do projeto e a facilidade com a qual se pode criar um aplicativo simples para testes.

Para todas as plataformas, foi feito, quando aplicável, um cadastro inicial e a criação de um aplicativo bastante básico que somente mostrasse uma tela com texto. Nos casos em que isso não foi possível completamente devido a falta de recurso nos planos gratuitos, foram utilizados vídeos disponíveis na web.

Abaixo, são detalhados os níveis da coluna Facilidade de Uso na Tabela 3.1:

- **Excelente**: A plataforma possui uma interface limpa, com termos simples e de fácil entendimento e utilização. É possível, facilmente, criar um aplicativo de testes simples sem necessidade de consulta à documentação. Não necessita instalação de aplicativos, funcionando de maneira completa via navegador.
- **Ótima**: A plataforma possui uma interface limpa e de fácil utilização. Pode ser necessário consultar a documentação para entender alguns termos e ferramentas. É possível criar um aplicativo de testes simples com pouca ou nenhuma consulta à documentação. Não necessita a instalação de aplicativos de computador, funcionando de maneira completa via navegador.
- **Aceitável**: A plataforma possui uma interface de fácil utilização, mas a quantidade

de ferramentas, menus e submenus pode torná-la confusa, necessitando consultas à documentação para que se encontre as funções desejadas. A criação de um aplicativo simples de teste pode necessitar consulta à documentação e tutoriais. Pode ser necessário instalar aplicativos de computador proprietários para possibilitar o uso completo da plataforma.

- **Ruim**: A plataforma possui interface com algumas familiaridades que podem facilitar o seu uso, mas a complexidade dos termos utilizados e a quantidade de caminhos a se percorrer para adicionar funções pode tornar seu uso inviável a pessoas leigas. A criação de um aplicativo simples para teste requer consultas à documentação e a tutoriais, mesmo para usuários acostumados com os termos complexos. Pode ser necessário instalar aplicativos de computador proprietários para possibilitar o uso completo da plataforma.

Para análise do suporte e comunidade, foram considerados critérios como qualidade, complexidade e alcance da documentação, seja escrita ou por vídeos oficiais, movimentação dos fóruns ou canais de comunidade e volume de tutoriais não oficiais, escritos e em vídeo, encontrados em pesquisas na web. Os níveis da coluna Suporte e Comunidade são:

- **Excelente**: A plataforma possui uma documentação clara e completa, com vídeos e tutoriais úteis. O suporte oficial é de fácil acesso e ágil. O fórum oficial da comunidade possui uma grande e ativa base de usuários, que ajudam uns aos outros. Há uma miríade de fontes externas de conteúdo sobre a plataforma.
- **Ótima**: A plataforma possui uma documentação completa, com vídeos e tutoriais, mas que pode ser um pouco complexa em alguns casos. O suporte oficial é de fácil acesso. O fórum oficial da comunidade possui uma base de usuários grande e ativa. É possível encontrar diversas fontes externas de conteúdo sobre a plataforma.
- **Aceitável**: A plataforma possui uma documentação extensa mas incompleta, ou completa, porém complexa. O suporte oficial é de fácil acesso. O fórum da comunidade possui uma base de usuários razoável e ativa. Há algum conteúdo externo sobre a plataforma.

Tabela 3.1: Análise das plataformas selecionadas

# Plataforma	Facilidade de Uso	Suporte e Comunidade	Custo Mensal
Adalo	Excelente	Excelente	$65
AppGyver	Ótima	Ótima	$0
FlutterFlow	Excelente	Ótima	$70
Google AppSheet	Aceitável	Ótima	$10
Mendix	Aceitável	Ótima	$60
Microsoft Power Apps	Ruim	Ótima	$20
Outsystems	Aceitável	Excelente	$1513
Thunkable	Ótima	Ótima	$200
Zoho Creator	Aceitável	Aceitável	$30

Fonte: O Autor

3.1 Estudo de caso da SAVE

A transição para a mobilidade elétrica é uma tendência crescente em todo o mundo, impulsionada pela busca por soluções mais sustentáveis no setor automotivo. Nesse contexto, empresas inovadoras estão surgindo para enfrentar os desafios do abastecimento de veículos elétricos (VE) e garantir uma experiência confiável aos usuários. Um exemplo local é a SAVE — Serviços de Abastecimento de Veículos Elétricos — uma empresa sediada em Porto Alegre, Brasil, fundada em 2022 por dois engenheiros com vasta experiência na indústria automotiva nacional e internacional.

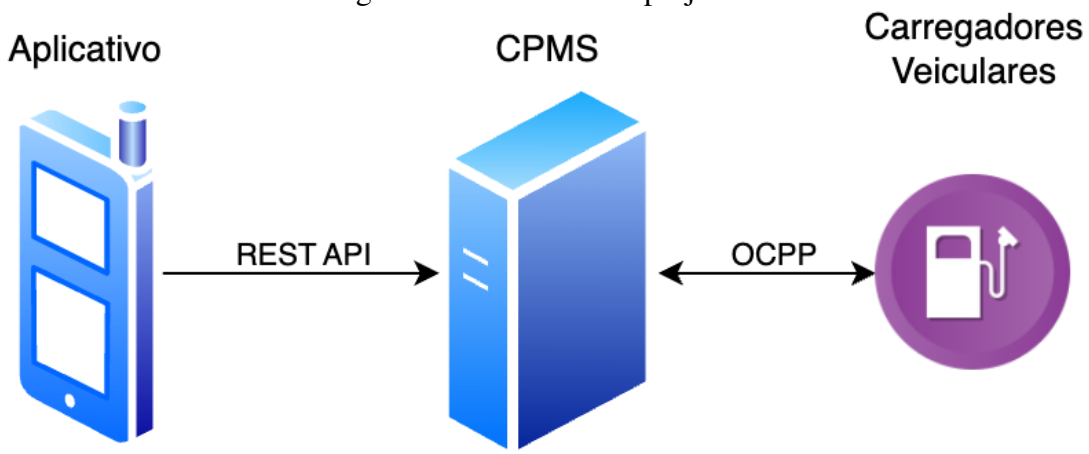
A SAVE propõe oferecer soluções abrangentes que se inserem no ecossistema do abastecimento de VE, focando em estabelecer uma “Rota Segura” para os usuários. Com uma visão que vai além do simples abastecimento, a empresa busca criar uma infraestrutura completa que proporcione autonomia aos usuários de VE em todo o território nacional e sul-americano (SAVE, 2022).

O diferencial da Rota Segura da SAVE é a sua capacidade de planejar a rota de forma inteligente, evitando o risco do VE ficar sem carga no trajeto. O sistema leva em consideração a autonomia do veículo, a distância a ser percorrida e a disponibilidade de pontos de recarga ao longo da rota, garantindo que os usuários tenham acesso a pontos de abastecimento confiáveis e estrategicamente posicionados.

Este estudo de caso tem como objetivo auxiliar a SAVE a desenvolver um aplicativo em formato de MVP para validar seus objetivos no mercado. O MVP será do tipo "produto com recursos mínimos", ou seja, com funcionalidades básicas, que permitirá testar a viabilidade do negócio e obter retorno dos usuários.

Após reunião inicial com os empreendedores, definiu-se o escopo do MVP. A necessidade principal seria a criação de um aplicativo móvel para possibilitar aos clientes

Figura 3.2: Estrutura do projeto



Fonte: O Autor

localizar um ponto de carga compatível disponível, dirigir até ele, carregar o veículo e pagar pela energia consumida.

Uma premissa para o desenvolvimento do aplicativo seria um CPMS disponível para comunicação com os pontos de carga e o gerenciamento das estações e recargas, bem como a referência e documentação de uma API REST para comunicação com o CPMS, como ilustrado na Figura 3.2.

O aplicativo terá as seguintes funcionalidades e recursos essenciais para validar o negócio:

- **Localização de carregadores**: os usuários poderão encontrar, em um mapa, os carregadores mais próximos de sua localização atual. Essa função permitirá que os usuários identifiquem facilmente os pontos de recarga disponíveis ao seu redor. Ao selecionar um carregador específico, o aplicativo permitirá ao usuário abrir a navegação até ele usando um aplicativo de navegação de sua preferência. Isso facilitará o acesso aos pontos de recarga escolhidos.
- **Carregamento do veículo**: para iniciar a recarga, o usuário poderá escanear um código QR impresso em um adesivo no carregador. Essa funcionalidade garantirá um processo rápido e conveniente para iniciar a recarga do veículo elétrico. Alternativas devem ser oferecidas para digitação manual do código, caso haja problemas de leitura.
- **Pagamento da energia consumida**: o aplicativo permitirá que os usuários efetuem o pagamento da energia consumida por meio de cartão de crédito. O usuário deve, antes de iniciar a recarga, escolher o valor dela, que deve ser reservado no limite

do cartão. Após finalizar, a transação é efetivada no valor consumido, que pode ser menor que o reservado. Essa função facilitará a transação financeira e proporcionará uma experiência sem complicações para os usuários.

- **Histórico de carregamentos**: os usuários poderão consultar um histórico de suas recargas por meio do aplicativo, bem como acompanhar a recarga atual. Essa funcionalidade permitirá que os usuários acompanhem e monitorem suas recargas anteriores, fornecendo um registro claro e útil de suas atividades de abastecimento.
- **Cadastro**: sistema simples de cadastro que solicite apenas nome, e-mail, senha e número de celular. O e-mail deverá ser verificado de alguma forma. Deve haver opções de alteração e recuperação de senha, bem como alteração de nome e número de celular.

Para que a startup alcance uma base de usuários mais ampla, a SAVE solicitou que o aplicativo possa ser publicado nas lojas de aplicativo móveis mais populares do mercado.

Esse MVP servirá como uma base sólida para futuras iterações e melhorias do aplicativo, à medida que a SAVE avança em direção a uma solução mais completa e abrangente.

3.2 Requisitos ampliados para escolha da plataforma

Após análise dos requisitos preliminares do projeto, novos requisitos para a escolha da plataforma devem ser considerados. Estes requisitos levam em conta a pouca experiência com desenvolvimento de software dos usuários que ficarão a cargo de continuar a manutenção do aplicativo após a entrega da versão MVP, bem como funções e recursos específicos para a integração com o CPMS.

- **Templates e Componentes Pré-Construídos**: a presença de templates e componentes pré-construídos pode acelerar o processo de desenvolvimento, permitindo que os fundadores utilizem leiautes e funcionalidades prontas, reduzindo o esforço e o tempo de criação do aplicativo.
- **Customização**: a plataforma deve oferecer opções de customização, permitindo que os fundadores personalizem o aplicativo de acordo com as necessidades e a identidade da startup, para que o aplicativo tenha uma aparência única e adequada ao mercado alvo.

- **Testes e Publicação**: verificar se a plataforma oferece ferramentas de teste e a capacidade de publicar o aplicativo nas lojas de aplicativos, como a App Store e o Google Play, de forma simples e direta.
- **Integração com APIs externas**: deve ser possível efetuar a integração com APIs REST externas à plataforma, de forma a tornar possível a comunicação com um CPMS.
- **Segurança**: a segurança é um fator crítico ao lidar com dados e comunicação entre o aplicativo e o CPMS. É preciso certificar-se de que a plataforma *no-code* ofereça recursos robustos de segurança, como autenticação de usuários e criptografia de dados.
- **Meios de Pagamento**: é preciso validar se a plataforma permite a integração de meios de pagamento via componentes prontos ou, pelo menos, via integração de API REST.

3.3 Escolha da plataforma

Levando em conta os requisitos ampliados e o escopo inicial do projeto MVP, foi escolhida a plataforma *no-code* AppGyver, da renomada empresa multinacional SAP. Entre os fatores decisivos, pode-se citar:

- **Componentes prontos**: a plataforma oferece uma ampla variedade de componentes prontos para uso, o que facilita o desenvolvimento rápido e eficiente.
- **Personalização avançada**: o AppGyver oferece um alto nível de personalização, permitindo a adaptação do aplicativo de acordo com necessidades específicas. É possível criar interfaces de usuário personalizadas, implementar lógica de negócios complexa e ajustar o projeto do aplicativo de acordo a marca e identidade visual da empresa.
- **Integração com APIs e fontes de dados**: a plataforma é especializada em desenvolvimento de aplicativos *frontend*, oferecendo integração com APIs REST e outras fontes de dados populares. Isso significa que é possível se conectar facilmente a diferentes sistemas e serviços, aproveitando dados e funcionalidades externas no aplicativo, essencial para o estudo de caso.
- **Compatibilidade multiplataforma**: com o AppGyver, é possível desenvolver aplicativos que funcionam tanto em dispositivos Android quanto iOS. A plataforma

permite compilar o aplicativo para as duas plataformas a partir do mesmo projeto, economizando tempo e esforço no desenvolvimento.

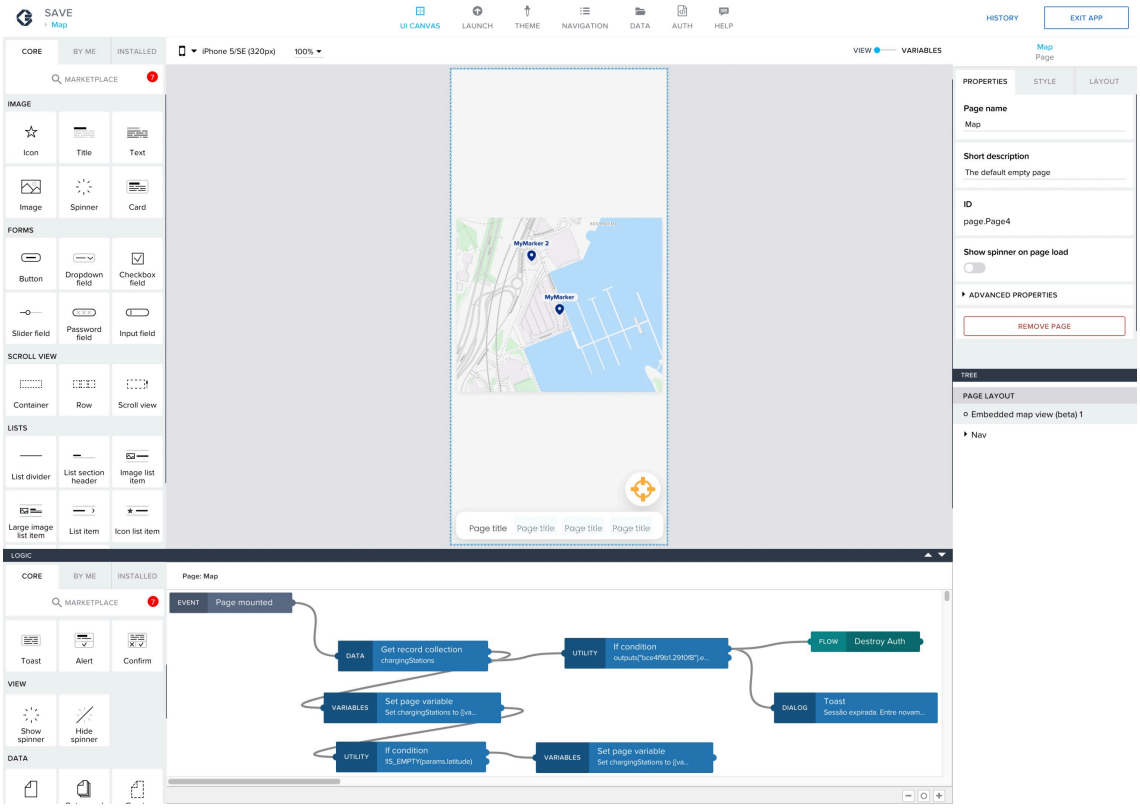
- **Interface intuitiva de desenvolvimento**: o AppGyver oferece uma interface de desenvolvimento visual e intuitiva com recursos de arrastar e soltar, o que torna a criação de leiautes e fluxos de ações muito mais fácil.
- **Suporte da SAP**: o AppGyver é desenvolvido pela SAP, uma empresa multinacional renomada no setor de tecnologia. Isso garante suporte técnico confiável, documentação razoável e confiabilidade da plataforma.
- **Comunidade ativa**: o AppGyver possui uma comunidade de desenvolvedores ativa, onde é possível obter suporte técnico, compartilhar conhecimentos e trocar ideias com outros usuários da plataforma. Isso facilita a resolução de problemas e o aprendizado contínuo.
- **Atualizações e novos recursos**: a plataforma é constantemente atualizada com novas funções e recursos, garantindo acesso às últimas tecnologias e tendências de desenvolvimento de aplicativos.
- **Precificação**: o AppGyver é uma versão de comunidade do SAP Build Apps, com menos recursos, porém gratuito. Levando em consideração que uma startup em início de operação não dispõe de tantos recursos, este é um grande diferencial.

A escolha da plataforma para o desenvolvimento do aplicativo foi fundamentada em uma série de motivos significativos. Primeiramente, a decisão foi influenciada pela experiência prévia do autor com a plataforma, o que permitiu uma curva de aprendizado mais eficiente e uma maior familiaridade com suas funcionalidades. Além disso, a gratuidade da plataforma aliada à sua compatibilidade com os requisitos do aplicativo desempenhou um papel crucial na decisão, garantindo uma gestão financeira eficiente do projeto. Embora a plataforma não tenha obtido a classificação mais elevada em comparação com outras alternativas, essas justificativas foram determinantes para a escolha do autor, uma vez que a decisão foi apresentada aos *stakeholders*, os quais expressaram seu consentimento.

3.4 Apresentação da plataforma

A interface de desenvolvimento do AppGyver, nomeada Composer Pro, tem como tela principal a *UI Canvas*, mostrada na Figura 3.3. A SAP conseguiu criar uma interface

Figura 3.3: Tela principal do Composer Pro — UI Canvas



Fonte: (SAP, 2023)

de desenvolvimento simples, porém completa. Não é necessário navegar por diversos menus e submenus para alcançar alguma ferramenta ou lógica desejada, por exemplo.

3.4.1 Leiaute

A plataforma trabalha com uma separação de conceitos bem definida para o desenvolvimento da aplicação. O desenvolvimento do leiaute conta com diversos componentes visuais predefinidos, como visto na barra lateral esquerda da Figura 3.3, abaixo de "*MARKETPLACE*". Um ponto muito interessante é a possibilidade de combinar os componentes básicos disponíveis para criar outros personalizados, que podem ser salvos — ficando disponíveis na aba "*BY ME*" — e, também, disponibilizados a terceiros por meio de publicação na *marketplace*. Lá, também é possível instalar componentes criados por outros usuários e, também, componentes mais específicos, disponibilizados pela própria equipe de desenvolvimento da plataforma, como, por exemplo, o componente de exibição de mapa.

Na barra lateral direita estão localizadas as abas para personalização das proprie-

dades, estilo e leiaute do componente selecionado. As propriedades servem de interface para a passagem de dados entre o contexto da página e do componente para serem utilizadas em sua lógica interna. Na aba de estilos, é possível escolher entre estilos predefinidos ou criar novos, que poderão ser facilmente reutilizados em outros componentes. Na aba de leiaute ficam opções de personalização do formato visual do componente, como tamanho, posição, alinhamento, entre outros.

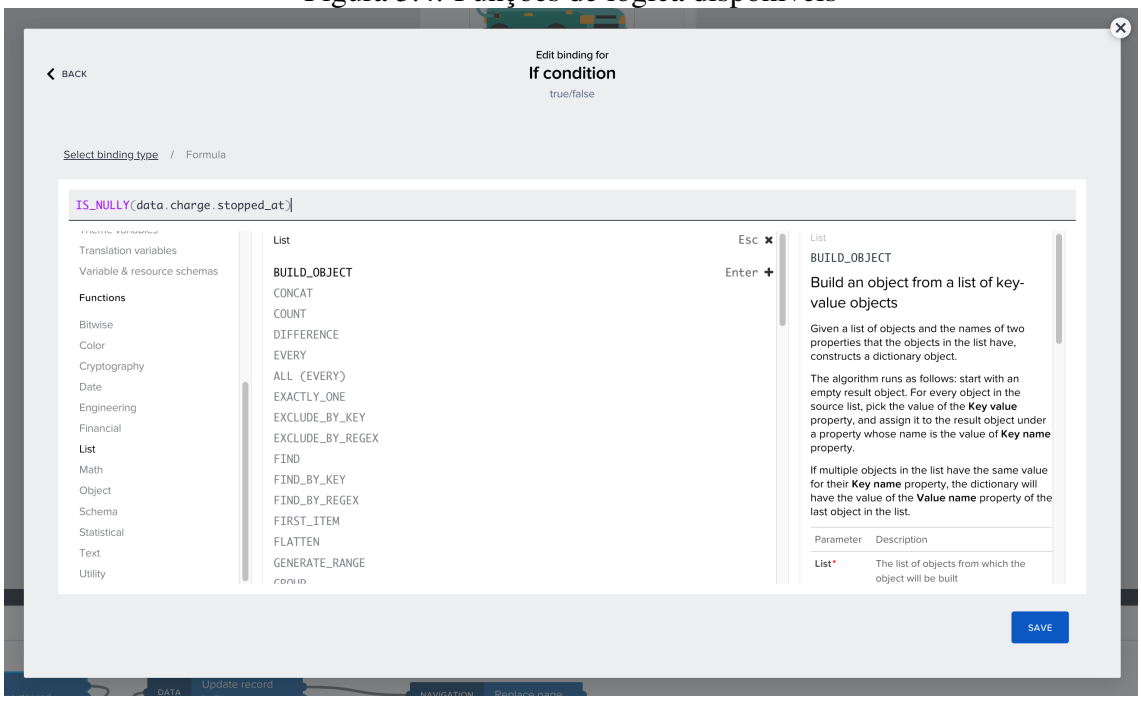
3.4.2 Lógica

Na parte inferior da tela, fica a área de implementação da lógica, que também funciona por meio de componentes de lógica. A mesma ideia de *marketplace* se aplica para componentes lógicos, e ainda há a possibilidade de criar lógica a partir de código JavaScript. A implementação de lógica no AppGyver é feita por meio de uma interface visual, onde você pode criar fluxos de trabalho e automações usando elementos de arrastar e soltar. Por exemplo, você pode configurar um fluxo de trabalho que ocorre quando um botão é pressionado ou uma página é aberta. Dentro desse fluxo de trabalho, você pode adicionar ações como fazer uma requisição de API, atualizar variáveis, exibir mensagens, navegar para outras páginas, entre outras. Existem diversos outros eventos que podem ser utilizados como gatilho de fluxos de trabalho, e inclusive é possível criar eventos personalizados dentro de um componente.

As fórmulas de lógica e manipulação mostradas na Figura 3.4 são usadas para criar regras e ações personalizadas nos fluxos de trabalho. Essas fórmulas permitem que você execute operações complexas, tome decisões com base em condições e manipule dados de várias maneiras. Embora possam parecer um tanto complexas à primeira vista, elas trabalham de forma muito semelhante ao que se pode encontrar em software de manipulação de planilhas, como o Microsoft Excel. Esse pode ser um ponto positivo, visto que, mesmo tendo pouca ou nenhuma experiência com desenvolvimento de software propriamente dito, os empreendedores de uma startup, em geral, possuem conhecimentos básicos dessas fórmulas.

Conforme se digita uma fórmula, o Composer Pro valida os tipos esperados de entrada e retorno, mostrando os erros de tipagem identificados ao usuário. A tipagem, porém, não é imposta e os erros podem ser ignorados. Obviamente, ignorar os tipos pode ser fonte de problemas, mas ocorrem casos em que se precisa usar algumas fontes de dados que simplesmente não permitem prever o tipo. Nesses casos, acaba sendo necessário

Figura 3.4: Funções de lógica disponíveis



Fonte: (SAP, 2023)

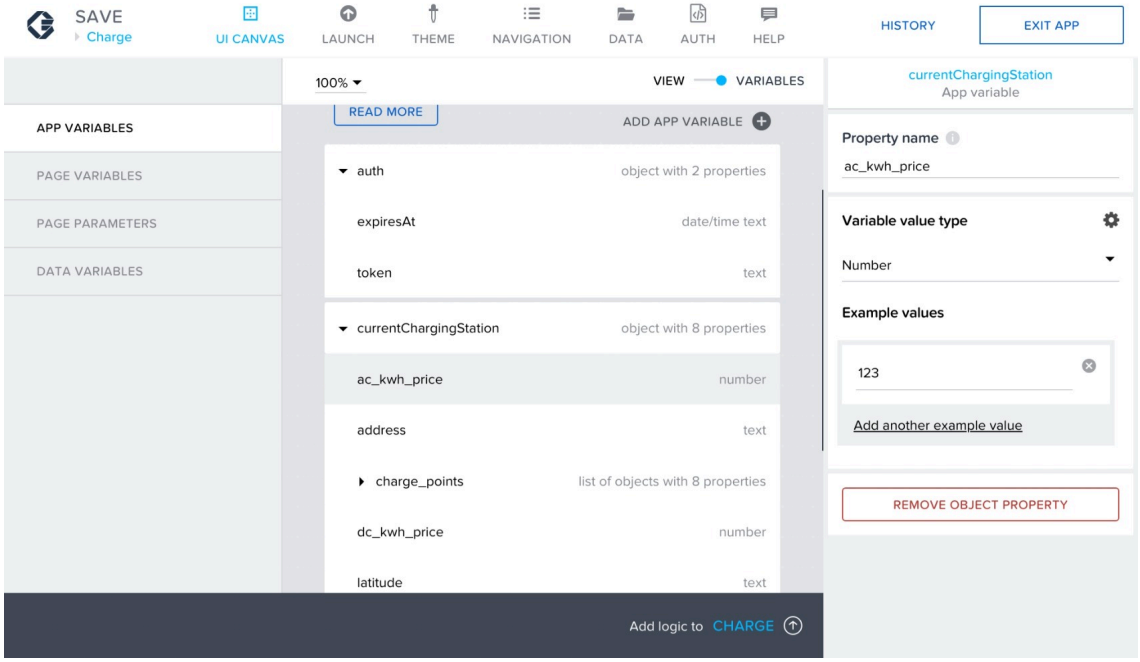
ignorar os avisos.

3.4.3 Variáveis

As variáveis utilizadas em uma página são dividas em 4 escopos diferentes: variáveis de aplicativo (*app variables*), de página (*page variables*), parâmetros de página (*page parameters*) e variáveis de dados (*data variables*). As variáveis de aplicativo seriam, basicamente, as variáveis globais do ambiente. Elas pode ser manipuladas a partir de qualquer página do aplicativo. Em geral, devem ser usadas para armazenamento de configurações, preferências e dados de autenticação. Como o nome indica, as variáveis de página são as variáveis locais, acessíveis apenas por componentes contidos dentro da mesma tela do aplicativo. Os parâmetros de página são aqueles que podem ser recebidos como argumentos em um redirecionamento de página, sendo uma forma de passar dados de uma página a outra sem ter que recorrer a variáveis globais. Por fim, as variáveis de dados são interfaces padrão de acesso às fontes de dados disponíveis — como uma API REST, por exemplo.

Componentes também podem ter suas próprias variáveis internas, que podem ser propriedades, já descritas anteriormente, ou variáveis internas, disponíveis somente den-

Figura 3.5: Visualização de variáveis da página



Fonte: (SAP, 2023)

tro do escopo interno. Elas existem pois é possível criar fluxos de trabalho a nível de componente, para manipular seu comportamento e, possivelmente, emitir eventos para o contexto externo. Isso é especialmente útil para a reutilização do mesmo em outras partes do aplicativo, bastando implementar a lógica somente uma vez e, depois, utilizar o mesmo componente onde for necessário.

A Figura 3.5 mostra a tela de visualização de variáveis da página em edição, que pode ser aberta pelo seletor deslizável no canto superior direito da *UI Canvas*.

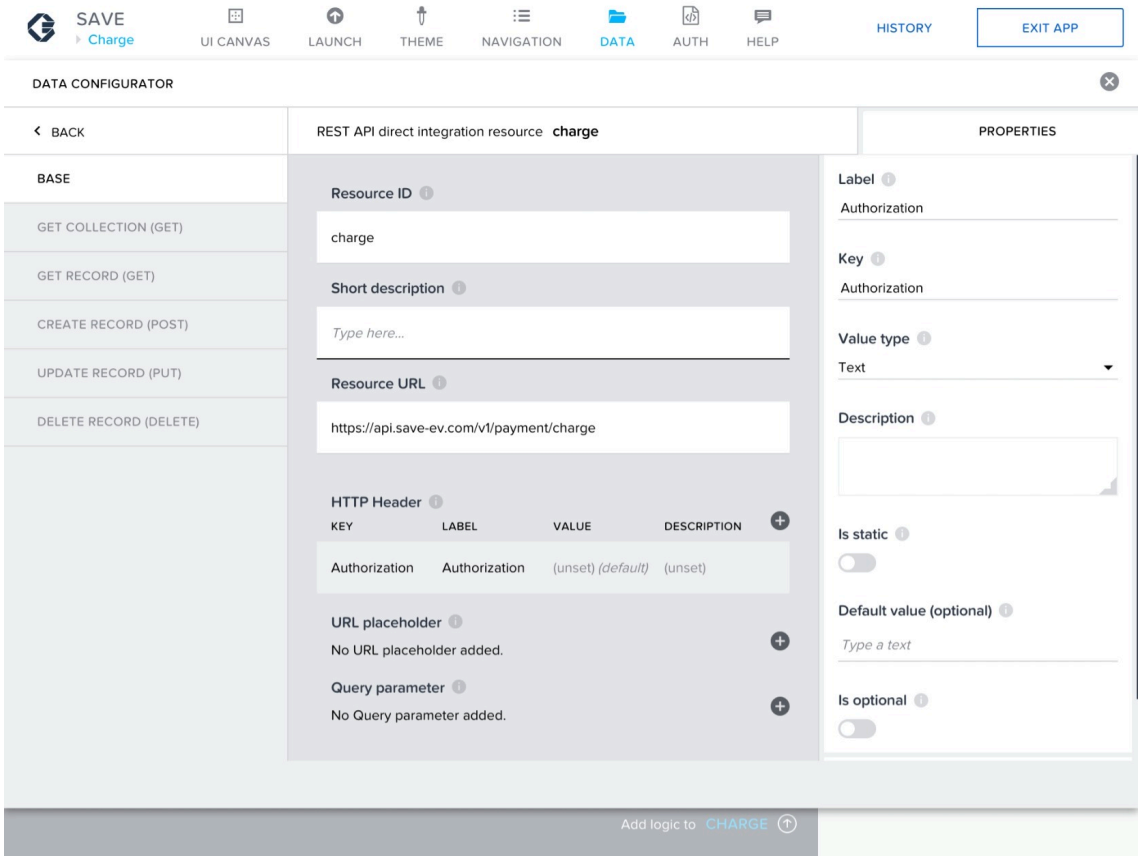
3.4.4 Fontes de dados

Acessando a página de fontes de dados pelo botão "*DATA*" na barra de navegação superior do Composer Pro, é possível criá-las a partir de alguns tipos de conexões externas, como APIs REST e Google Firebase, ou do armazenamento interno do aparelho, útil para salvar preferências que devem persistir mesmo com o aplicativo fora de execução.

A Figura 3.6 mostra a interface de criação de uma conexão com um recurso de API REST. Algo que não fica claro na interface é que essa funcionalidade funciona exclusivamente com APIs REST baseadas em formato JSON, não sendo possível utilizar outros formatos.

Para cada recurso, é possível utilizar 5 ações. As 5 ações são provenientes de

Figura 3.6: Criação de uma fonte de dados para uma API REST



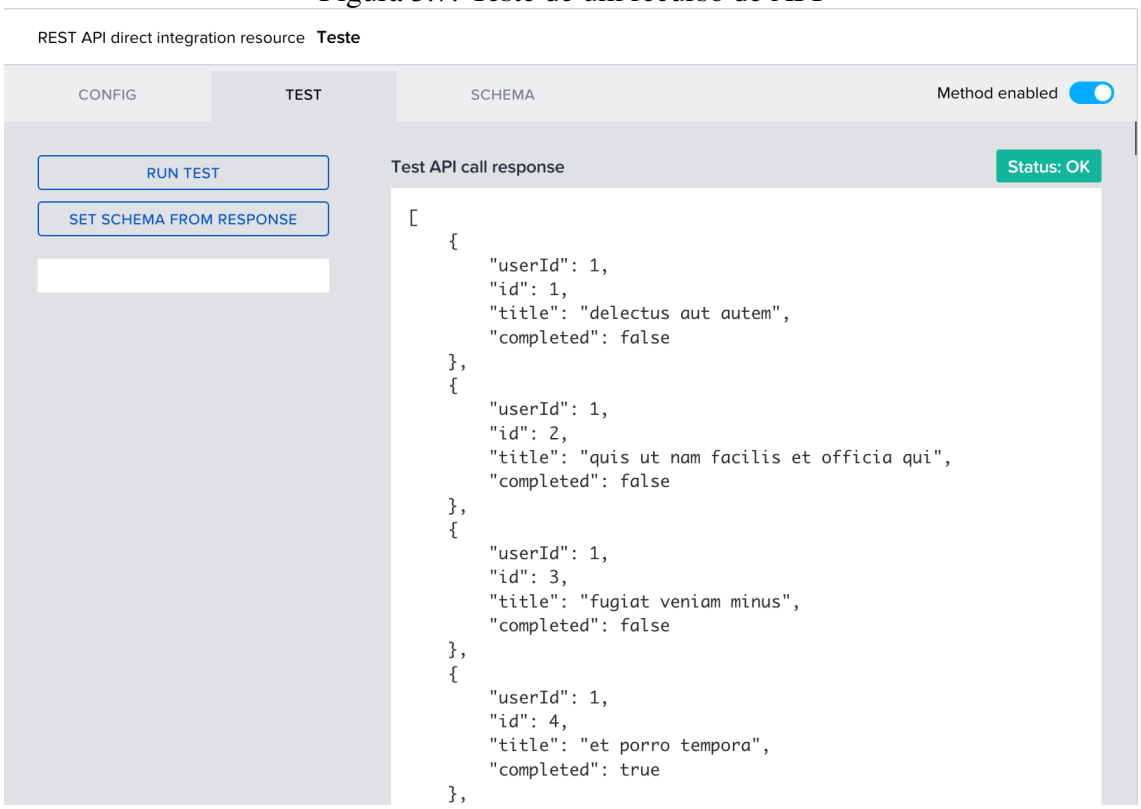
Fonte: (SAP, 2023)

um padrão de projeto comumente utilizado em APIs REST, que é o CRUD (do inglês, *Create-Read-Update-Delete*), sendo o método *GET* utilizado para consulta ou leitura de dados, tanto em lista quanto de forma unitária, o método *POST* para criação de dados, o *PUT* para modificação e o *DELETE* para exclusão.

3.4.4.1 Mapeamento de propriedades

Um excelente recurso do configurador de dados, responsável por uma das economias de tempo mais significativas na integração de APIs no AppGyver, é a possibilidade de gerar o esquema de dados a partir de uma resposta de requisição que pode ser efetuada na aba de testes (*test*), representada na Figura 3.7. Nela, após configurar o URL do recurso na aba de configurações (*config*), é possível efetuar uma requisição teste e receber o corpo da resposta na visualização ao lado. Caso houvesse cabeçalhos necessários para a autenticação, como o *Authorization*, ele aparecia nesta tela para preenchimento manual. Na tela de configurações do recurso, também é possível indicar o caminho dentro do corpo da resposta a serem buscados os dados. Isso é útil em APIs que retornam os dados dentro

Figura 3.7: Teste de um recurso de API



Fonte: (SAP, 2023)

de um objeto *data*, por exemplo.

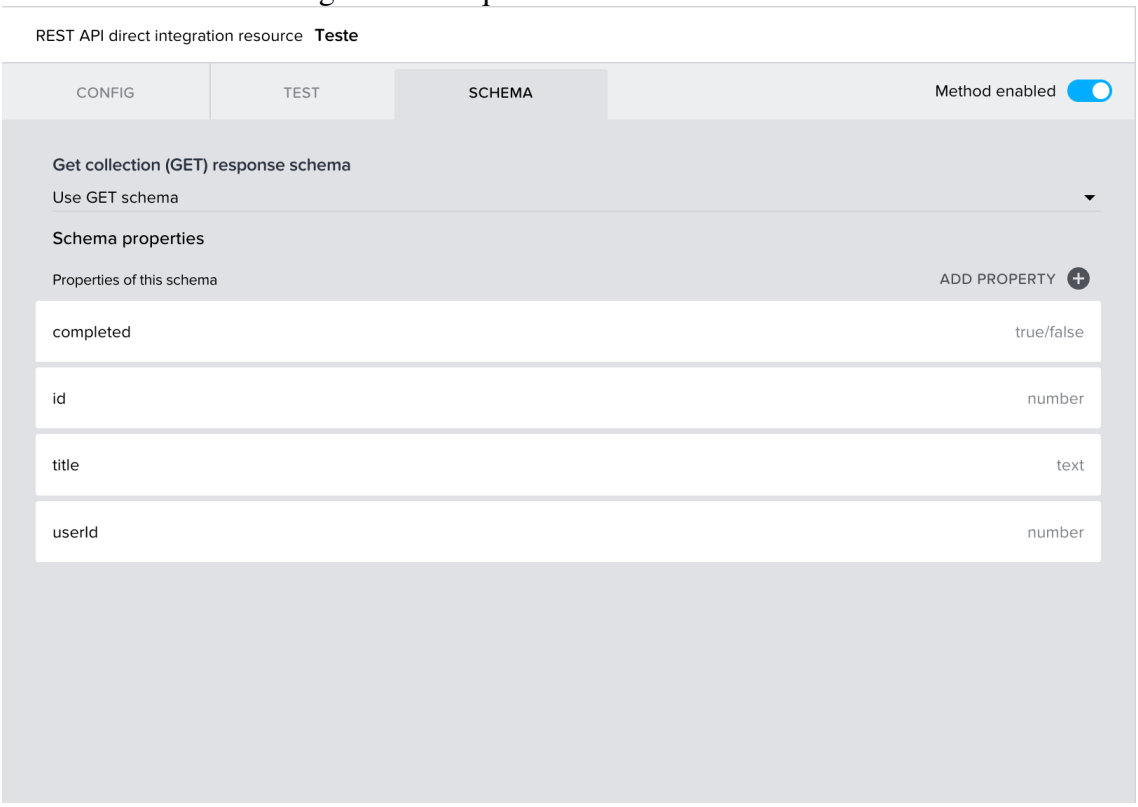
Com a resposta esperada, é possível utilizar o botão "*SET SCHEMA FROM RESPONSE*" para gerar, automaticamente, o esquema de tipos do objeto retornado. O resultado para a requisição utilizada na Figura 3.7 é mostrado na Figura 3.8. Note que os dados foram corretamente tipados. É possível alterá-los ou fazer o processo manualmente, caso necessário.

O efeito disso é que as variáveis de dados irão utilizar objetos com essas propriedades tipadas, e o Composer Pro as sugere sempre que se está utilizando essas variáveis, de modo semelhante às ferramentas de auto completar código. Não só isso, como na interface de criação de fórmulas o tipo de cada propriedade do objeto é verificado e inconsistências são informadas normalmente.

3.4.4.2 Possível problema de escalabilidade

Para utilizar os recursos de API criados dentro de uma página, é preciso criar uma variável de dados para a página, que fica atrelada à ação do recurso de API desejado. Por padrão, nos casos de leitura dos dados, o AppGyver cria um fluxo de trabalho que

Figura 3.8: Esquema de um recurso de API



Fonte: (SAP, 2023)

fica fazendo requisições à API a cada 5 segundos, de modo a implementar uma espécie de sincronização em tempo real da variável de dados com a fonte de dados. Entretanto, esse pode rapidamente se tornar um causador de negação de serviços por parte da API, na medida em que o usuário começar a utilizar muitas variáveis de dados que são atualizadas constantemente. Por exemplo, caso o usuário utilize uma API com limitação de requisições diárias, ele pode se ver em uma situação em que o aplicativo para de funcionar após atingir o limite, devido à recusa da API. Esse é um problema grave em relação à escalabilidade do aplicativo. É possível facilmente contornar isso removendo o fluxo de dados criado automaticamente, visto que, na maioria dos casos, não é necessário ter o dado em tempo real, e sim somente coletar o valor atual uma vez. Porém, usuários leigos podem notar esse tipo de problema em um ponto muito avançado do desenvolvimento, quando o aplicativo escalar em número de usuários e o problema começar a ocorrer. Nesse ponto, pode ser custoso remover os fluxos e alterar a lógica.

3.4.5 Estilos e Fontes

O Composer Pro possui uma página onde se pode criar paletas de cores padrão para serem utilizadas dentro do aplicativo. Isso é de grande utilidade na medida em que evita a preocupação com especificação manual de cores para os componentes. Na hora de escolher uma cor seja para texto ou leiaute, as paletas padrão são mostradas e basta selecionar aquela desejada. Com isso, alterações de cor a nível de aplicativo são tão fáceis quanto alterar a paleta uma única vez e ver o resultado ser refletido em todas as páginas.

Para fontes, a mesma ideia é possível. É possível criar tipos de fonte, como primária ou secundária, e vincular a classe a um componente. Se for preciso alterar a fonte, basta alterar a fonte do tipo, e isso será refletido em todo o aplicativo.

Por fim, até mesmo estilos de componente podem ser salvos e reutilizados no aplicativo. Os estilos incluem informações como cores, fonte, corpo da fonte, sombras, bordas, cantos arredondados, cor de fundo, entre outros. Um bom exemplo seria a criação de um botão. Um botão nada mais é que um *container* com um certo estilo. Aplique um fundo colorido, bordas, sombra e cantos arredondados nele e pronto, se tem um botão. Se precisar fazer botões de diferentes tamanhos, basta salvar o mesmo estilo e aplicar neles.

3.4.6 Autenticação

O AppGyver utiliza uma forma bem simples de implementar a autenticação no aplicativo. Basicamente, existem dois contextos de aplicação disponíveis: não autenticado e autenticado. A única forma de trocar de um contexto ao outro é utilizando um componente de lógica específico que faz a troca de contexto. Páginas de um contexto não podem acessar páginas de outro contexto via redirecionamento simples. Com isso, é possível criar as páginas de cadastro e recuperação de senha, por exemplo, no contexto sem autenticação. Após autenticar o usuário utilizando algum meio, como usuário e senha validados em uma fonte de dados de API, basta utilizar o componente de troca de contexto para autorizar o usuário.

3.4.7 Testes

Para possibilitar testes dos aplicativos em tempo real, a SAP disponibiliza nas lojas de aplicativos da Apple e da Google o aplicativo SAP Build Apps, que, apesar do nome, também é compatível com o AppGyver. Para iniciar os testes, o processo é tão simples quanto baixar o aplicativo e ler um código QR na aba "*LAUNCH*" do AppGyver. O aplicativo é baixado e simulado no celular em poucos segundos. Esse é um grande diferencial da plataforma, pois permite testar os recursos em um dispositivo real e ver os resultados.

3.4.8 Compilação e publicação

O AppGyver não oferece integração com as lojas de aplicativos para possibilitar publicação automática. Porém, é possível compilar e gerar os arquivos para publicação manual. O processo, no entanto, não é trivial, envolvendo necessidade de geração de certificados e chaves de segurança que podem ser impeditivos para que um usuário leigo consiga fazer o processo.

Ademais, não foi encontrado nenhum tutorial completo para o processo especificamente no AppGyver. Após algum estudo sobre compilação de aplicativos para Android, foi possível gerar o arquivo e instalá-lo manualmente em um dispositivo Android. Tudo funcionou conforme esperado. Para iOS, infelizmente não foi possível efetuar os testes pois é necessária a criação de uma conta de desenvolvedor com a Apple, o que geraria custos.

Esse poderia ser um impeditivo importante para a escolha de uma plataforma, e foi identificado tardiamente. Porém, em breve análise das 2 plataformas melhor qualificadas na Tabela 3.1, Adalo e FlutterFlow, conclui-se que esse processo também não seria trivial, embora essas plataformas tenha documentação e tutoriais mais completos.

Dessa forma, levando em consideração a gratuidade do AppGyver, manteve-se a escolha.

4 REQUISITOS E ARQUITETURA

Neste capítulo, inicialmente, examinaremos em detalhes os requisitos detalhados para o aplicativo, consolidando todas as necessidades dos usuários e partes interessadas em um conjunto coerente e completo. Em seguida, apresentamos uma arquiterura preliminar com base nesses requisitos.

No entanto, ao longo do processo, percebemos que uma análise mais aprofundada era necessária, devido a desafios inesperados que surgiram. Assim, na terceira subseção, exploraremos as adaptações realizadas aos requisitos de forma a resolver esses desafios, assegurando que o aplicativo permaneça alinhado com os objetivos e as expectativas. Por fim, apresentaremos a arquitetura definitiva do aplicativo, após todas as iterações e aperfeiçoamentos, oferecendo uma visão completa e detalhada da estrutura que guiará o desenvolvimento e a implementação do aplicativo.

4.1 Consolidação dos requisitos do aplicativo

Com os requisitos básicos definidos e a plataforma escolhida, foram agendadas algumas reuniões para definir-se a lista completa e detalhada dos requisitos que atendessem ao projeto de MVP do aplicativo. Após algumas iterações, chegou-se ao detalhamento descrito na sequência:

4.1.1 Localização de carregadores

A tela principal do aplicativo, após efetuada autenticação, deve ser um mapa que ocupa toda a tela e exibe um pino na localização atual do aparelho, bem como marcadores de cada estação de carregamento disponível.

Para encontrar uma estação o usuário deverá arrastar o dedo para movimentar a região visualizada no mapa e usar movimentos de pinça para controlar a escala.

Ao tocar numa estação de carga, uma tela com detalhes da mesma deve ser exibida, e deve exibir as seguintes informações:

- Endereço da estação, que, ao ser tocado, encaminha as coordenadas ao aplicativo de navegação padrão do dispositivo do usuário, para iniciar percurso.
- Tarifas do kWh (quilowatt-hora) que será cobrado no carregamento, sendo divididas

por tipo de corrente elétrica do carregador: corrente alternada (AC) ou digital (DC).

- Lista dos pontos de carga (carregadores) disponíveis, contemplando detalhes de cada um:
- Tipo de corrente
- Tipo de conector
- Potência de carga em quilowatts (kW)
- Disponibilidade
- Ocupado: quando está sendo utilizado em um carregamento ativo
- Disponível: quando está livre e apto para iniciar um carregamento
- Indisponível: quando está desativado por qualquer motivo técnico
- Ao tocar em um carregador na lista, deve-se apresentar o código identificador para possibilitar a comparação com o adesivo do mesmo e então poder iniciar o carregamento.

4.1.2 Carregamento do veículo

Encontrada a estação de carga, o aplicativo deve permitir que o usuário inicie um carregamento de seu veículo utilizando a câmera de seu dispositivo para efetuar a leitura de um código QR. Caso a leitura do código venha a falhar por qualquer motivo, deve-se oferecer a possibilidade de digitação manual dele. Em último caso, ainda resta a opção de iniciar a carga pela tela de detalhes da estação de carga, conforme listado anteriormente.

A tela de um ponto de carga deve incluir os seguintes detalhes dele:

- Tipo de corrente
- Tipo de conector
- Potência de carga em kW
- Disponibilidade do carregador
- Ocupado: quando está sendo utilizado em um carregamento ativo
- Disponível: quando está livre e apto para iniciar um carregamento
- Indisponível: quando está desativado por qualquer motivo técnico
- Valor do kWh a ser aplicado no carregamento
- Código identificador do carregador
- Nome da estação de carga em que está localizado

Bem como exibir controles para configurar o início de um carregamento:

- Seletor deslizável do valor a ser retido no cartão, com opção de digitar manualmente
- Seletor do cartão de crédito que se deseja utilizar na transação
- Botão para iniciar o carregamento

A tela de um carregamento em curso ou concluído deve apresentar:

- Status do carregamento: carregando, com falha ou concluído
- Motivo da parada (quando aplicável)
- Consumo atual em kWh com precisão de 3 casas decimais
- Consumo monetário atual em Reais com precisão de 2 casas decimais
- Data e hora de início da carga
- Data e hora de fim da carga (quando aplicável)
- Tempo decorrido em horas e minutos
- Limite retido no cartão de crédito
- Tarifa em vigor de cada kWh
- Estação do ponto de carga
- Código identificador do ponto de carga
- Tipo de conector em uso
- Tipo de corrente em uso
- Potência de carga em uso
- Cartão de crédito utilizado
- Botão para parar a carga manualmente

A tela também deve ser atualizada em tempo real e sem necessitar intervenção do usuário, de forma a mostrar o progresso do carregamento gradualmente e receber informações de eventuais paradas por parte do sistema.

- Os possíveis motivos de parada de um carregamento predefinidos devem ser:
- A bateria atingiu 100% de carga
- O consumo atingiu o valor de limite retido no cartão de crédito
- O conector foi removido manualmente do veículo
- Parada manual via botão no aplicativo
- Falha de conexão ao carregador

4.1.3 Pagamento

O CPMS que foi fornecido para a SAVE possui integração, atualmente, somente com a Stripe, uma das maiores plataformas de intermediação de pagamento disponíveis atualmente (CONRAD, 2022). Portanto, a SAVE já havia providenciado o cadastro da empresa com a Stripe e possuía todos os dados de integração necessários.

Para que o fluxo de pagamento desejado fosse implementado, ou seja, com retenção em limite de cartão, foi solicitado como requisito uma tela de gerenciamento dos cartões de crédito do usuário. Nela, deve ser possível listar os cartões cadastrados, apagá-los e cadastrar novos.

A pedido da SAVE, o detalhamento dos requisitos dessa página será omitido neste trabalho, e destaca-se que o conteúdo segue as diretivas da Lei Geral de Proteção de Dados Pessoais (LGPD), Lei n° 13.709/2018.

O processo de cadastro do cartão deve seguir o fluxo recomendado pela Stripe para que haja segurança na transação e armazenamento dos dados do cartão. Isso envolve o envio dos dados crus do cartão diretamente para a Stripe, sem passar pelos servidores da SAVE, incluindo o próprio CPMS. Os únicos dados que a SAVE pode consultar dos cartões dos clientes são os mesmos listados acima, para a listagem de cartões já cadastrados, fora o código de identificação único do cartão gerado pela Stripe para identificá-lo em sua base de dados.

4.1.4 Histórico de carregamentos

Para o usuário ter controle do seu histórico de carregamentos, deve haver uma tela com uma listagem resumida dos seus carregamentos com a SAVE. Para cada carregamento, deve-se exibir os seguintes dados de maneira resumida, podendo-se utilizar ícones em vez de rótulos para economizar espaço:

- Nome da estação
- Data e hora do início
- Valor retido
- Tarifa do kWh
- Consumo em kWh
- Consumo em reais

- Status

Ao clicar em um carregamento, abre-se a página dele, já descrita na seção de carregamento do veículo acima. Por fim, deve-se oferecer, na interface do aplicativo, um atalho que redirecione para o carregamento em curso. Não havendo um, deve ser redirecionado para o histórico.

4.1.5 Cadastro

O aplicativo deve ter áreas de criação, recuperação, autenticação e modificação de cadastro do usuário.

Para criação, deve haver a possibilidade de efetuar um novo cadastro em uma tela alcançável sem a necessidade de autenticação. Para o cadastro, deve ser exigido:

- Nome completo
- E-mail
- Número de telefone celular
- Senha

O usuário deve poder visualizar os termos de serviço e política de privacidade, repetir a senha para confirmação, e deve verificar o e-mail informando um código de 6 dígitos recebido nele.

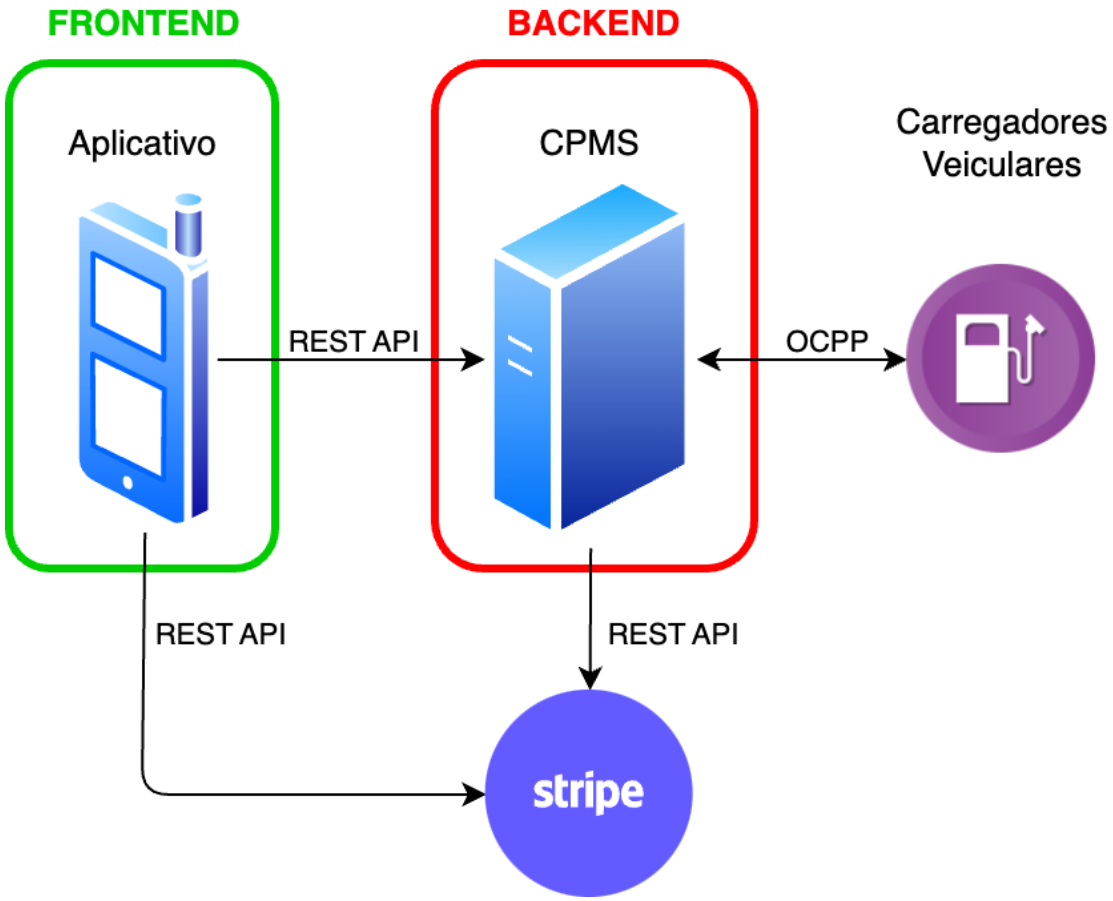
Para a autenticação, deverá ser exibida, como tela inicial geral do aplicativo, um formulário solicitando o e-mail e senha.

Para recuperação de senha, deverá ser disponibilizada uma tela de recuperação de senha. O usuário deve informar o e-mail do cadastro para receber nele um código de 6 dígitos que pode, então, ser usado para a alteração de senha.

Para a alteração de dados do cadastro e de senha sem necessidade de código de verificação, deve ser disponibilizada uma tela, disponível somente após a autenticação, onde seja possível alterar nome, número do celular e senha.

A senha deve ser composta de, pelo menos, 8 caracteres. Deve ter, ao menos: um número; uma letra minúscula; uma letra maiúscula e um caractere especial.

Figura 4.1: Arquitetura preliminar da solução



Fonte: O Autor

4.2 Arquitetura preliminar da solução

Com todos os requisitos definidos, a arquitetura preliminar da solução, ilustrada na Figura 4.1 envolve o uso dos recursos do CPMS para atuar como o *backend* do serviço, disponibilizando uma API REST para o aplicativo móvel — o *frontend* — coletar e enviar as informações necessárias para seu funcionamento.

O CPMS, por usa vez, fica responsável pela implementação das regras de negócio, gerenciamento dos carregadores via OCPP e integração segura com o meio de pagamento via API REST com a chave privada de acesso da Stripe.

Por fim, com objetivo de implementar as regras de segurança para integração com a Stripe, o *frontend* também deverá ter uma integração básica via API REST diretamente com a Stripe utilizando a chave pública de acesso, para evitar o trânsito de dados sensíveis por dentro do CPMS.

4.3 Ajuste de escopo e requisitos

Após a consolidação dos requisitos acima, o trabalho de desenvolvimento da interface foi iniciado na plataforma AppGyver. Entretanto, após algumas semanas em contato com os fornecedores do CPMS, a SAVE informou que não haveria uma API REST disponível para efetuar a integração do aplicativo com o CPMS. O fornecedor esclareceu que a API atualmente disponível ainda não estava em fase de disponibilização para utilização por parte de terceiros, e que somente era usada para a integração com o próprio aplicativo móvel também fornecido por eles.

Esse aplicativo já era conhecido pela SAVE, e foi desconsiderado pois é um aplicativo genérico para uso de qualquer cliente com o CPMS fornecido, e não conta com nenhuma possibilidade de tradução para português ou personalização de marca.

O projeto inicial deste trabalho, descrito até aqui, contava com uma API pronta para comunicação com o CPMS, com todas as suas regras de negócio prontas. Com isso em vista, e boa parte da interface do aplicativo pronta, foram feitas novas reuniões para decidir o caminho a ser tomado. Em primeiro momento, foi feita uma pesquisa sobre as possibilidades de aumentar o escopo do trabalho, a fim de desenvolver o caminho completo de comunicação do aplicativo com os carregadores. Concluiu-se que seria necessário o desenvolvimento de um *backend* para atender as requisições do aplicativo segundo as regras de negócio e requisitos da SAVE, o que inclui integração com a API de

pagamentos da Stripe.

Para isso, seria necessário o desenvolvimento de um servidor de *backend* minimamente funcional para servir todos os recursos necessários para o funcionamento do aplicativo. Além disso, a comunicação via OCPP com os carregadores deveria ser implementada no mesmo servidor.

4.3.1 Desenvolvimento do *backend*

Conforme já mencionado, existem diversos fornecedores de CPMS no mercado, mas os valores de licenciamento ainda são altíssimos. Soluções *open-source* foram pesquisadas, mas ainda são escassas. Foi encontrada uma possível ferramenta para ajudar na integração com os carregadores: o SteVe (STEVE, 2023).

O SteVe é, basicamente, uma *dashboard* que oferece interfaces amigáveis para enviar e receber comandos dos carregadores. É possível cadastrar os carregadores, iniciar e parar a carga, entre outras funções previstas no protocolo OCPP. No entanto, esse servidor, infelizmente, não oferece uma API REST suficientemente completa que possibilite usá-lo como um intermediário de comunicação com o nosso servidor de *backend* do aplicativo. Uma versão do SteVe com uma API REST mais completa foi disponibilizada após a finalização do aplicativo e início de escrita deste texto, por isso acabou não sendo avaliada.

Considerando a complexidade da tarefa de implementar essa comunicação com carregadores reais em tempo hábil, foi acordado que isso ficaria fora do escopo deste trabalho. Em vez disso, foi adicionado como requisito uma forma de simular o avanço e paradas de uma carga diretamente via aplicativo:

- Uma função para alterar manualmente o consumo atual de uma carga, para simular o consumo aumentando e as estatísticas do carregamento sendo atualizadas
- Uma forma de simular a parada do carregamento usando os motivos de parada previstos

Além disso, adicionou-se ao escopo o desenvolvimento de um servidor *backend* para disponibilizar uma API REST JSON para homologação do aplicativo, de forma que pudesse ser testado em sua forma completa.

Tendo em vista que a escolha da plataforma *no-code* já havia sido feita e que boa parte da interface já estava pronta, foi decidido que, ao invés de refazer o aplicativo

em outra plataforma que também tivesse recursos de *backend*, ele seria desenvolvido de maneira tradicional, utilizando Laravel para PHP — um framework livre e open-source criado por para o desenvolvimento de sistemas web (OTWELL, 2023) — cujo autor deste texto possui anos de experiência profissional. O servidor para a hospedagem do *backend* foi disponibilizado pela SAVE.

4.3.2 Integração com a Stripe

Revistos os requisitos, iniciou-se uma pesquisa sobre a integração com a API de pagamentos da Stripe. Infelizmente, ela não fornece uma API REST com corpo da requisição em formato JSON, e, sim, em formato *x-www-form-urlencoded*. Esse formato trabalha com a serialização dos dados em pares de chave-valor, separados por "&", e com os valores codificados de acordo com as regras padrões de codificação URL.

Isso acabou sendo um problema potencialmente grande, ao passo que o AppGyver somente oferece métodos de integração com APIs REST JSON. Dessa forma, inviabilizou-se que fossem feitas chamadas à API da Stripe diretamente do aplicativo.

A solução para esse problema se deu por meio de um componente *webview* disponível para uso no AppGyver. Nesse componente, é possível definir uma URL para ser renderizada dentro da área do *webview*. Uma página web foi desenvolvida manualmente em HTML, CSS e JavaScript para exibir o formulário de cadastro dos cartões de crédito, para ser renderizada dentro do *webview* no aplicativo.

Dessa forma, a primeira tentativa foi utilizar o pacote JavaScript pronto que a Stripe fornece, com todas as medidas de segurança já bem implementadas. Porém, o módulo que implementa a *webview* dentro do AppGyver acabou não sendo compatível com o módulo da Stripe, e o formulário não funcionava dentro do aplicativo.

Por fim, a solução encontrada foi implementar uma solução em JavaScript nativo — sem a utilização de pacotes externos — que utiliza a função fetch nativa do JavaScript para efetuar requisições na API da Stripe.

Basicamente, para evitar que dados sensíveis dos cartões do usuário sequer passem por dentro dos servidores da SAVE, essa página, carregada na *webview* dentro do celular do usuário, faz uma requisição de criação de cartão diretamente à Stripe utilizando a chave pública de acesso da SAVE, que não oferece riscos mesmo estando embarcada e, potencialmente, visível no celular do usuário. A resposta da requisição é um token que representa o cadastro do cartão na Stripe, sendo temporário e de uso único.

Imediatamente após receber o token, a página faz uma nova solicitação de API, dessa vez a da SAVE, enviando o token para que, então, utilizando a chave privada e secreta, armazenada com segurança dentro do servidor, seja feita uma requisição que vincula o cartão identificado pelo token ao cadastro do usuário na Stripe (que foi criado junto com o cadastro na SAVE).

Outro desafio foi que a *webview* trabalha de forma isolada ao aplicativo. Não é possível receber dados da página de volta ao aplicativo, mas é possível enviar dados como parte da URL. Portanto, para conseguir informar o token de forma segura na requisição à API da SAVE — visto que a página dentro da *webview* não tem dados para autenticação de usuário —, antes de abrir a *webview*, o aplicativo faz uma requisição de geração de URL assinada temporária, atrelada ao usuário. Essa URL é codificada e passada como parâmetro na URL da página da *webview*, que a decodifica e usa para enviar o token.

Após um retorno positivo, é exibido uma mensagem de sucesso, e fica a cargo do usuário tocar no botão "voltar" no aplicativo para voltar à tela anterior, visto que não é possível receber um evento da *webview* para fazer um redirecionamento automático.

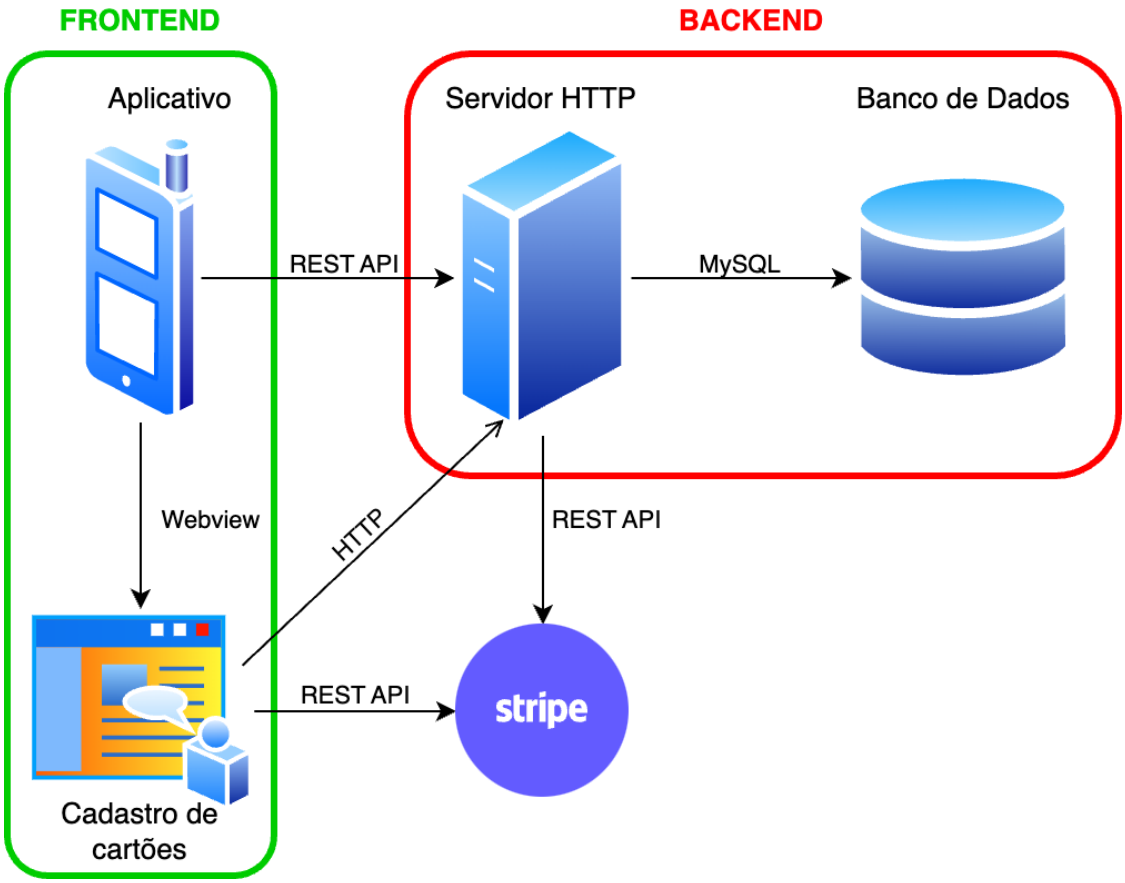
4.4 Arquitetura final da solução

Após todos os ajustes em relação ao *backend* e a introdução da *webview* ao *frontend*, a arquitetura final do projeto ficou conforme a Figura 4.2.

O *backend*, agora sem recursos de comunicação com os carregadores, apenas simula os carregamentos para possibilitar o teste completo do aplicativo. Para ser possível o armazenamento das informações, o banco de dados, antes implícito no CPMS, agora aparece na arquitetura pois é necessário para a implementação. A API REST da Stripe é utilizada para sincronizar os cadastros dos usuários, utilizando a chave privada.

O aplicativo utiliza o componente de *webview* para renderizar a página de criação de cartões, que por sua vez utiliza a API REST da Stripe com a chave pública para enviar os dados sensíveis de cartão.

Figura 4.2: Arquitetura final da solução



Fonte: O Autor

5 APRENDIZADO E IMPLEMENTAÇÃO

Ao todo, 17 páginas foram criadas no Composer Pro do AppGyver para que fosse possível a implementação de todos os requisitos listados anteriormente. O processo de criação do aplicativo se estendeu por meses em função de todos os problemas e desafios que apareceram ao longo do estudo de caso. Os tempos de desenvolvimento dos recurso de API necessários e das páginas do aplicativo acabaram sendo mesclados em função dos mesmos terem sido desenvolvidos quase que simultaneamente, e adaptados conforme necessário.

No entanto, os tempos de aprendizado de ferramentas e recursos específicos foram medidos e podem ser apresentados de forma independente dos tempos de real implementação. Então, para medir os tempos de implementação, as principais páginas do aplicativo foram refeitas do zero para que fosse possível entender quanto tempo elas demandam de um usuário experiente na plataforma e que possui todos os recursos de API prontos para integração.

5.1 Tempo de aprendizado

A arquitetura utilizada no Composer Pro pode não ser tão trivial como parece ao olhar para sua interface. Alguns conceitos podem se tornar confusos até que se comece a criar aplicações de testes e a brincar com os componentes e lógicas. Infelizmente, a documentação própria da SAP foi insatisfatória em alguns pontos, por ser simples demais e carecer de detalhes. Ela somente apresenta como utilizar os conceitos, mas não oferece uma explicação ampla do que são esses conceitos e porque são usados de tal forma. Felizmente, o fórum da comunidade possui diversos tópicos abertos e resolvidos de pessoas que tiveram dificuldades. Canais no YouTube também foram bastante úteis para o entendimento dos conceitos e de como utilizá-los.

Uma análise inicial da plataforma, incluindo download do aplicativo de testes e primeira simulação nele usando o projeto padrão com somente uma tela e texto levou em torno de 20 minutos.

5.1.1 Componentes personalizados

Conforme detalhado no Capítulo 3, é possível criar componentes personalizados e reaproveitá-los no resto no aplicativo. Basta adicionar um componente *container* à pagina e, dentro dele, ir adicionando outros componentes até montar o que precisa. Feito isso, existe um botão em suas propriedades para convertê-lo em um componente personalizado.

Existem dois escopos de edição de um componente personalizado: local e geral. O local altera somente a própria instância em que se está trabalhando, se tornando uma extensão do componente geral. O geral altera a base do componente e tem efeito em todo o aplicativo. Essa disposição não ficou totalmente clara a partir da documentação e interface, e foi fonte de gasto de tempo com estudos alternativos.

Aproximadamente, 5 horas foram utilizadas para o entendimento completo, com testes, de componentes personalizados, incluindo criação de propriedades e lógica interna.

5.1.2 Estilos e leiaute

A organização de estilos no Composer Pro é trivial para pessoas que tem experiência em criação de leiautes para a web. No entanto, pode ser um pouco confusa, inicialmente, para leigos. A documentação, porém, acaba sendo suficiente para, junto com testes, entender e aplicar os conceitos.

O leiaute das páginas também segue os mesmos padrões e nomenclaturas de leiaute. Ao contrário dos estilos, aqui a documentação acabou pecando em não explicar com detalhes o que cada nome significa dentro do leiaute. Um usuário leigo pode ficar confuso ao tentar entender a diferença entre as diferentes formas de alinhamento. No entanto, buscas rápidas pelos termos tendem a esclarecer essas dúvidas, visto que, por serem jargões amplamente utilizados, a quantidade de resultados com explicações é abundante.

Aproximadamente, 2 horas foram utilizadas para se obter um entendimento completo.

5.1.3 Variáveis

As variáveis de aplicativos e de página, bem como os parâmetros são de trivial entendimento e, mesmo para leigos, o texto explicativo sobre cada uma que aparece na

própria interface do Composer Pro é suficiente para um rápido entendimento do escopo de cada.

As variáveis de dados, contudo, são outra história. Elas requerem a criação de uma fonte de dados a qual devem ser vinculadas. A documentação é um tanto vaga, mas tutoriais no YouTube foram bastante úteis para o entendimento. O tempo aproximado gasto com o entendimento da estrutura geral, com criação de testes de integração com API externas usando os diferentes métodos para consultar um objeto ou uma lista de objetos, criar, modificar e apagar levou em torno de 8 horas.

5.1.4 Lógica

A interface de criação de lógica foi um dos pontos que surpreenderam na facilidade de entender e criar fluxos de trabalho. Todo fluxo de trabalho começa a partir de algum evento, como tocar um componente e abrir uma página. Os componentes de lógica disponíveis são, em geral, autoexplicativos, como “Abrir página”, “Tirar foto” e “Escanear código QR”.

Basta arrastar o componente e ligar ele no evento que se deseja utilizar como disparo da ação. O resultado dele pode ser ligado com outros componentes que armazenam os dados em uma variável, por exemplo, e o componente visual que estiver vinculado nessa variável é atualizado dinamicamente com o novo valor. Para controlar o fluxo, existe um componente condicional, para o qual se deve utilizar uma variável ou fórmula que resultem em um valor verdade.

O maior tempo aqui foi gasto com testes de fluxo e de formulas, a fim de entender até que nível de complexidade se podia chegar, visto que entender como se utilizar a interface é trivial. Aproximadamente, 4 horas foram utilizadas.

5.1.5 Compilação

Embora disponível, o processo de compilação de um aplicativo na plataforma não é nada trivial. Conhecimentos específicos de desenvolvimento das lojas de aplicativos são necessários. É necessário providenciar um certificado de distribuição no caso do iOS, que deve ser obtido com a Apple. No caso do Android, é necessário gerar uma *keystore*, que basicamente permite armazenar chaves privadas em um contêiner para dificultar a

extração do dispositivo. A documentação falha em fornecer detalhes sobre isso, e não encontramos tutoriais específicos para AppGyver, o que certamente seria um grande problema para usuários leigos.

Com conhecimentos sobre como gerar a *keystore*, foi possível compilar o aplicativo para Android e instalá-lo em um celular para testes. O tempo gasto para isso foi em torno de 2 horas.

Tabela 5.1: Resumo dos tempos de aprendizado

Objeto de estudo	*Tempo investido*
Componentes personalizados	5 horas
Estilos e leiaute	2 horas
Variáveis	8 horas
Lógica	4 horas
Compilação	2 horas
TOTAL	16 horas e 20 minutos

Fonte: O Autor

5.2 Tempos e desafios de implementação

Algumas páginas foram notoriamente simples de implementar, enquanto outras exigiram várias horas para que se pudesse implementar todos os fluxos de trabalho necessários. O fluxo esperado de ações da página — comumente chamado de "caminho feliz" — é fácil de ser mentalizado, bastando somente tempo para traduzi-lo em lógica. Porém, para se obter uma experiência agradável de uso do aplicativo, é preciso fazer testes e forçar erros a fim de encontrar os "caminhos tristes". Esse tratamento de erros, em muitos casos, tomou tanto tempo quanto a criação da página em si.

A Tabela 5.2 mostra o resumo do tempo investido para a implementação de cada página. Comentaremos sobre os pontos de maiores desafios e investimento de tempo.

5.2.1 Validação

A página Cadastro merece atenção por ter sido necessário criar um fluxo de trabalho complexo e com muitas ramificações, a fim de tratar os possíveis erros retornados pela API e, também, para efetuar a validação de dados antes mesmo de efetuar a requisição para a API. Mesmo assim, o tratamento não ficou completo. Alguns exemplos de validação no aplicativo são: verificação se o número de telefone só contem número; se

Tabela 5.2: Resumo dos tempos de implementação

#	*Página*	*Tempo investido*
1	Adicionar cartão	20 minutos
2	Alterar cadastro	30 minutos
3	Cadastro	2 horas
4	Confirmação da recuperação de senha	30 minutos
5	Estação de Carga	1 hora
6	Identificar Carregador	20 minutos
7	Mapa	2 horas
8	Meu cadastro	1 hora
9	Meus Cartões	1 hora
10	Obtendo localização	20 minutos
11	Ponto de Carga	1 hora
12	Recarga	4 horas
13	Recargas	1 hora
14	Solicitação de recuperação de senha	30 minutos
15	Tela inicial - Login	1 hora
16	Trocar Senha	1 hora
17	Verificar e-mail	20 minutos
	TOTAL	17 horas e 40 minutos

Fonte: O Autor

o nome contém 5 ou mais caracteres; se o e-mail é válido e se as duas senhas digitadas são iguais. Caso algo falhe, uma mensagem é exibida na tela pedindo para revisar. Com dados preliminarmente válidos, é feita a requisição à API, que também pode retornar outros erros de validação, como complexidade da senha e e-mail em uso. Uma mensagem de erro genérica foi implementada em casos de erros da API, pedindo para que o usuário reveja os dados e tente novamente.

Ficou claro, ao desenvolver essa tela, que uma implementação mais robusta de tratamento de erros é de difícil escala. Para uma versão MVP, ficou decidido que poderia ser feito algo mais simples, por isso a decisão de exibir uma mensagem genérica para qualquer erro de API.

5.2.2 Autenticação

Conforme explicado na Seção 3.4, após uma autenticação bem sucedida na API, é preciso trocar o contexto da aplicação para o contexto autenticado. Essa ação é trivial, porém é necessário armazenar os dados de sessão de alguma forma que não sejam perdidos após o usuário fechar o aplicativo, nos casos em que ele escolher a opção para manter-se conectado. A forma encontrada de se fazer isso foi armazenando o *token* de

acesso retornado pela API no armazenamento local do dispositivo. Caso o usuário não escolha manter-se conectado, também é salva a data e hora de expiração.

O próximo desafio foi implementar o tratamento de erro de *token* expirado. Em todas as páginas que fazem requisições autenticadas para a API, é preciso verificar se houve erro de autenticação e, caso positivo, redirecionar o usuário de volta para o contexto não autenticado. Felizmente, é possível criar um componente de lógica personalizado a partir de um pedaço de fluxo de trabalho. Isso economiza algum tempo por não ter que refazer o mesmo fluxo em todas as páginas necessárias, mas ainda assim requer que o desenvolvedor lembre de coletar o retorno da API e passar por esse fluxo antes de continuar.

5.2.3 Localização e mapa

O componente de mapa disponível para utilização na plataforma não está disponível de forma padrão, por não estar em uma versão completa estável ainda. Ele deve ser instalado manualmente pela *marketplace*. O maior problema da versão disponível na época de criação da página era que o componente somente abre o mapa nas coordenadas solicitadas, e não as atualiza em tempo real. Portanto, antes de renderizar o mapa, as coordenadas devem estar prontas, e não é possível fazer um acompanhamento dinâmico da localização atual do dispositivo.

Devido ao fluxo de eventos utilizado na compilação das páginas, foi necessário separar a coleta da localização do usuário do mapa em páginas distintas. Em uma única página, o mapa acaba sendo renderizado antes do retorno do sensor de localização, e acaba usando coordenadas zeradas.

Portanto, foi necessário o uso de um botão para coleta de localização atual que, quando utilizado, abre a página de localização, que fica aguardando até que o sensor responda e, com a as coordenadas prontas, encaminha via parâmetro para a página do mapa em si. Essa página também é exibida após o *login*.

Fora esses problemas, o componente de mapa ainda não estava com a configuração de estilos em total funcionamento. Foi possível alterar ícones e cores, porém o tamanho dos ícones, não.

Em geral, a experiência de usuário de utilização do mapa ficou bastante insatisfatória. Atendeu à necessidade do MVP, mas teria que receber muitas melhorias para uma versão futura, levando em conta a necessidade de desenvolvimento da Rota Segura.

5.2.4 Páginas dinâmicas

A página que mais demandou tempo foi a página de Recarga. Essa página conta com elementos dinâmicos que se tornam visíveis ou invisíveis conforme o status do carregamento. As estatísticas atualizam em tempo real, como consumo em kWh, consumo em reais e tempo decorrido. Eventuais erros na recarga também são tratados e componentes trocam de cor para indicar isso.

Ainda, para simular o carregamento, é possível alterar os dados diretamente via banco de dados, e o aplicativo a as refletirá, ou pode-ser usar atalhos ocultos na tela para enviar comandos via API que simulam o carregamento.

5.3 Reconsideração de outras plataformas

Aproveitando o estudo preliminar das plataformas, pode-se analisar se haveria uma opção melhor após ter o conhecimento das mudanças que ocorrem ao longo do estudo. Basicamente, os dois detalhes que teriam feito uma grande diferença, tendo em vista a versão final dos requisitos do aplicativo, seriam: a disponibilidade de um *backend* hospedado pela própria plataforma e integrado ao aplicativo, de forma a permitir implementar-se as regras de negócio nele sem necessidade de escrever código; e uma integração com REST APIs de qualquer formato de dados ou, melhor ainda, uma integração nativa com a Stripe.

5.3.1 FlutterFlow

A plataforma FlutterFlow oferece integração nativa com a Stripe, e o processo de integração é simples e fácil, como mostrado em (FLUTTERFLOW, 2023). No entanto, a forma de se trabalhar com um *backend* na FlutterFlow é, na verdade, uma integração nativa com o Google Firebase, que possibilita a criação de *backends no-code* para integração com aplicativos (GOOGLE, 2023). Portanto, isso acabaria gerando dois custos separados, bem como necessidade de aprender a utilização de diferentes plataformas e interfaces.

5.3.2 Mendix

A Mendix permite fazer a integração com a Stripe, por possuir suporte ao formato de resposta utilizado na API da plataforma de pagamentos. O processo de integração, porém, é bastante técnico e pode acabar sendo complexo demais para o cidadão-desenvolvedor. Também é possível gerar aplicativos para iOS e Android, e existe uma boa documentação detalhando o processo embora seja necessário instalar um programa de computador proprietário da Mendix para viabilizar o processo.

5.3.3 Adalo

A Adalo oferece integração nativa e fácil com a Stripe, também com boa documentação para tirar dúvidas. Ela trabalha com banco de dados integrado, tornando o desenvolvimento do *backend* e *frontend* fácil e transparente, como na Bubble.io. Finalmente, a Adalo oferece meios de publicar os aplicativos diretamente nas lojas de aplicativos da Apple, e para a Google é fornecida uma documentação completa com todos os passos necessários. A documentação mostra o passo a passo desde a criação de contas de desenvolvedor nas lojas, como geração das credenciais e arquivos necessários e até formas de publicar em modo teste nas plataformas.

6 CONSIDERAÇÕES FINAIS

O objetivo proposto deste trabalho foi de entender se as plataformas *no-code* disponíveis atualmente teriam a capacidade de ajudar o cidadão-desenvolvedor a criar um aplicativo em formato MVP para seu negócio. Por meio do estudo de caso da SAVE, foi feita uma filtragem e análise inicial de diversas plataformas que atendiam, pelo menos preliminarmente, aos objetivos do MVP da empresa. A plataforma AppGyver foi, finalmente, escolhida, e iniciou-se o trabalho de implementar o aplicativo móvel com ela. Diversos desafios surgiram ao longo do caminho, exigindo mudanças de escopo do trabalho e, até mesmo, o desenvolvimento tradicional de um *backend* de homologação para servir o aplicativo. Foi possível atender o escopo completo, mas tendo sido necessário desenvolver a página de cadastro de cartões com código. Finalmente, em conjunto com a breve análise de outras plataformas na Seção 5.3, é possível concluir que o cidadão-desenvolvedor poderia criar um aplicativo simples e limitado por conta própria, embora ainda tenha poucas opções de plataformas que poderiam propiciar isso a ele.

Caso as premissas iniciais do estudo de caso tivessem sido mantidas e considerando os tempos analisados no capítulo anterior, teria sido possível implementar o aplicativo em formato MVP em menos de uma semana, considerando uma jornada de trabalho de 8 horas por dia. Isso é um resultado excelente, levando em conta que desenvolver algo semelhante utilizando-se dos SDKs nativos para cada plataforma poderia durar meses, visto que o trabalho teria que ser feito em dobro para atender ambos os sistemas operacionais para os quais o AppGyver possibilita a compilação.

Entretanto, após esse estudo detalhado da plataforma, conclui-se que ela não seria, talvez, a melhor opção para os cidadãos-desenvolvedores, devido à complexidade de algumas tarefas e à falta de alguns recursos. A dificuldade de compilar os aplicativos e publicá-los nas lojas provavelmente seria impeditiva por si só. Para o caso da SAVE, a falta de integração com APIs REST de outros formatos que não JSON, também, seria proibitiva, visto que impede a integração com a Stripe, por exemplo, sem usar código.

Casos de uso um pouco mais avançados, como a exibição de um mapa com atualização em tempo real simplesmente não são possíveis na versão atual. Isso não impediu a construção da versão MVP para a SAVE, mas não seria aceitável para uma versão final do produto.

Todavia, para casos de uso menos complexos quanto foi o da SAVE, a plataforma é uma ótima opção para aceleração de MVPs, e até mesmo poderia ser utilizada para

versões finais de aplicações mais simples, assumindo-se que se tenha o *backend* necessário pronto. A sensação que ficou, porém, é que a plataforma não é voltada para o cidadão-desenvolvedor comum, e sim para pessoas que já possuem alguma experiência com desenvolvimento e querem agilizar o tempo de entrega dos aplicativos.

Após todo esse percurso, a conclusão geral é que, mesmo escolhendo uma plataforma que atendesse ao escopo final deste trabalho — como a Adalo, por exemplo — os *stakeholders* da SAVE provavelmente teriam dificuldades para implementar o projeto completamente, englobando a interface com os carregadores.

Nenhuma plataforma *no-code* ou *low-code* atual oferece suporte ao OCPP. E, mesmo que houvesse um CPMS com API REST disponível para a integração, é possível que essa integração não seja tão trivial quanto se imaginava, exigindo alguns conhecimentos específicos de desenvolvimento de software que o cidadão-desenvolvedor pode não possuir.

A recomendação que ficaria para a SAVE seria contratar um CPMS que ofereça aplicativos prontos *whitelabel* para poder usar como seu MVP e alavancar o negócio. O CPMS também deve ter integrações prontas e bem documentadas via API REST, pois tendo um bom retorno do público, um caminho a seguir poderia ser contratar um desenvolvedor de software iniciante para criar um novo aplicativo personalizado em umas das plataformas mencionadas neste capítulo e que tenha integração com o CPMS. Dessa forma, o escopo inteiro do projeto estaria englobado de forma mais personalizada.

REFERÊNCIAS

ALTAF, Y. **Four Main Types Of Minimum Viable Products And How They Can Help Software Startups**. 2022. Disponível em <https://www.forbes.com/sites/forbestechcouncil/2022/07/08/four-main-types-of-minimum-viable-products-and-how-they-can-help-software-startups>. Acesso em: 5 set. 2023.

BOBOYOROVA, C. **An Introduction To No-Code/Low-Code Platforms: Seven Things You Should Know**. 2023. Disponível em <https://www.forbes.com/sites/forbestechcouncil/2023/03/07/an-introduction-to-no-codelow-code-platforms-seven-things-you-should-know>. Acesso em: 5 set. 2023.

CIAT. **What are HTML, CSS, Javascript?** 2022. Disponível em <https://www.ciat.edu/blog/html-css-javascript>. Acesso em: 5 set. 2023.

CODAR, C. S. **Guia de Ferramentas No-Code**. 2023. Disponível em <https://guianocode.bubbleapps.io/version-test>. Acesso em: 5 set. 2023.

CODECADEMY. **What is REST?** 2023. Disponível em <https://www.codecademy.com/article/what-is-rest>. Acesso em: 5 set. 2023.

COLEMAN, G.; O'CONNOR, R. An investigation into software development process formation in software start-ups. **J. Enterprise Inf. Management**, v. 21, p. 633–648, Out. 2008.

CONRAD, A. **The Collison Brothers Built Stripe Into A $95 Billion Unicorn With Eye-Popping Financials. Inside Their Plan To Stay On Top**. 2022. Disponível em <https://www.forbes.com/sites/alexkonrad/2022/05/26/stripe-exclusive-interview-collison-brothers-95-billion-plan-to-stay-on-top>. Acesso em: 5 set. 2023.

CROCKFORD, D.; MORNINGSTAR, C. Standard ecma-404 the json data interchange syntax. Dez. 2017.

CURRENT. **What Is A Charge Point Management System And How Does It Work?** 2022. Disponível em <https://www.current.eco/ev-glossary/charge-point-management-system-cpms>. Acesso em: 5 set. 2023.

FLUTTERFLOW. **Stripe - FlutterFlow Docs**. 2023. Disponível em <https://docs.flutterflow.io/settings-and-integrations/in-app-purchases-and-subscriptions/stripe>. Acesso em: 5 set. 2023.

G2. **No-Code Development Platforms**. 2023. Disponível em <https://www.g2.com/categories/no-code-development-platforms>. Acesso em: 5 set. 2023.

GARTNER. **Enterprise Low-Code Application Platforms**. 2023. Disponível em <https://www.gartner.com/reviews/market/enterprise-low-code-application-platform>. Acesso em: 5 set. 2023.

GOOGLE. **Firebase - Google's Mobile and Web App Development Platform**. 2023. Disponível em <https://firebase.google.com>. Acesso em: 5 set. 2023.

GRIFFITH, C. **What is Hybrid Mobile App Development?** 2023. Disponível em <https://ionic.io/resources/articles/what-is-hybrid-app-development>. Acesso em: 5 set. 2023.

HUGHES, C. **4 security concerns for low-code and no-code development**. 2022. Disponível em <https://www.csoonline.com/article/572021/4-security-concerns-for-low-code-and-no-code-development-2.html>. Acesso em: 5 set. 2023.

IETF. **RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax**. 2005. Disponível em <https://datatracker.ietf.org/doc/html/rfc3986>. Acesso em: 5 set. 2023.

IETF. **RFC 9110 - HTTP Semantics**. 2022. Disponível em <https://datatracker.ietf.org/doc/html/rfc9110>. Acesso em: 5 set. 2023.

OTWELL, T. **Laravel - The PHP Framework for Web Artisans**. 2023. Disponível em <https://laravel.com/>. Acesso em: 5 set. 2023.

PATERNOSTER, N. et al. Software development in startup companies: A systematic mapping study. **Information and Software Technology**, v. 56, Out. 2014.

REDHAT. **O que é API REST?** 2023. Disponível em <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>. Acesso em: 5 set. 2023.

RIES, E. **The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses**. New York: Crown Business, 2011.

SAP. **SAP AppGyver - Composer Pro**. 2023. Disponível em <https://platform.appgyver.com/auth/community>. Acesso em: 5 set. 2023.

SAVE. **Nossa História**. 2022. Disponível em <https://save-ev.com/2023/01/30/post-3/>. Acesso em: 5 set. 2023.

STAMFORD, C. **Gartner Forecasts Worldwide Low-Code Development Technologies Market to Grow 20% in 2023**. 2022. Disponível em <https://www.gartner.com/en/newsroom/press-releases/2022-12-13-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-20-percent-in-2023>. Acesso em: 5 set. 2023.

STEVE. **SteVe - OCPP server implementation in Java**. 2023. Disponível em <https://github.com/steve-community/steve>. Acesso em: 5 set. 2023.

TIBCO. **What is API Throttling?** 2023. Disponível em <https://www.tibco.com/reference-center/what-is-api-throttling>. Acesso em: 5 set. 2023.

WEVO. **What is OCPP (Open Charge Point Protocol) and How Is It Used?** 2022. Disponível em <https://wevo.energy/glossary/what-is-ocpp>. Acesso em: 5 set. 2023.

WICKRAMARACHCHI, V. **The BFF Pattern (Backend for Frontend): An Introduction**. 2021. Disponível em <https://blog.bitsrc.io/bff-pattern-backend-for-frontend-an-introduction-e4fa965128bf>. Acesso em: 5 set. 2023.

ANEXO A — REPOSITÓRIO DO CÓDIGO FONTE DO *BACKEND*

https://github.com/alexandre-da-costa/save-ev-api

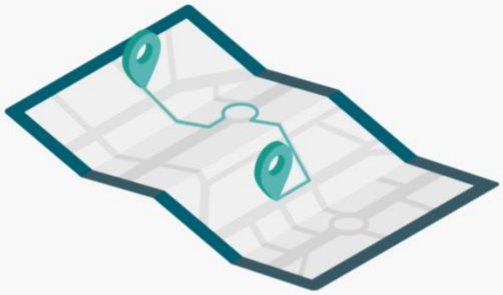
ANEXO B — IMAGENS DAS PÁGINAS DO APLICATIVO


20:19
SAVE
Carga Rápida e Segura
E-mail
Senha
Manter conectado
ENTRAR
Ainda não sou cadastrado
Esqueceu sua senha

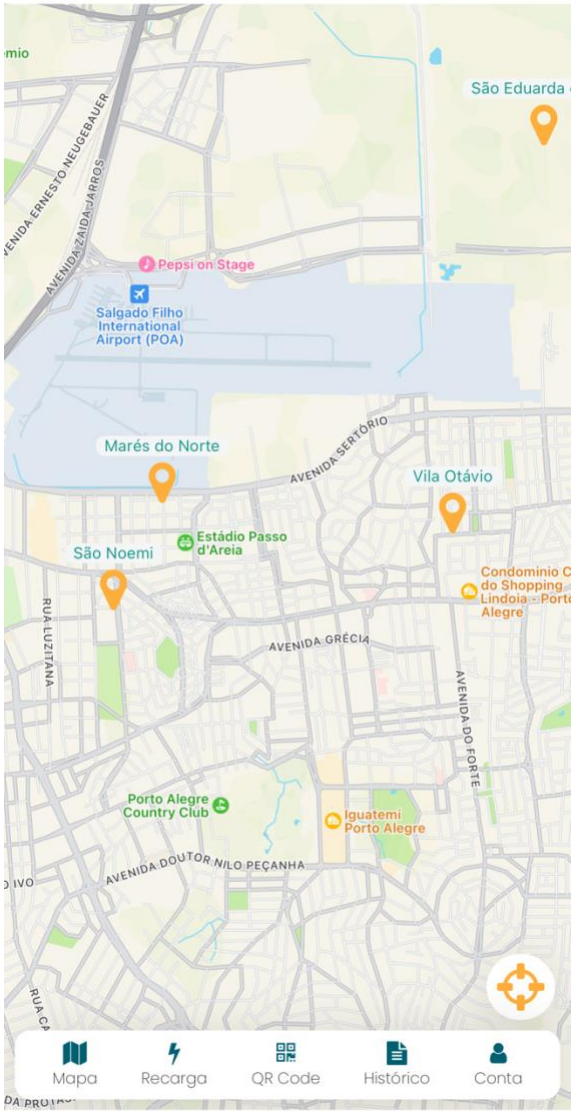

20:19
SAVE
Carga Rápida e Segura
Nome completo
Celular com DDD
E-mail
Senha
Confirmação da senha
Termos de Serviço e Política de Privacidade
Ao se cadastrar, você confirma que aceita nossos Termos de Serviço e Política de privacidade.
CADASTRAR
Já sou cadastrado

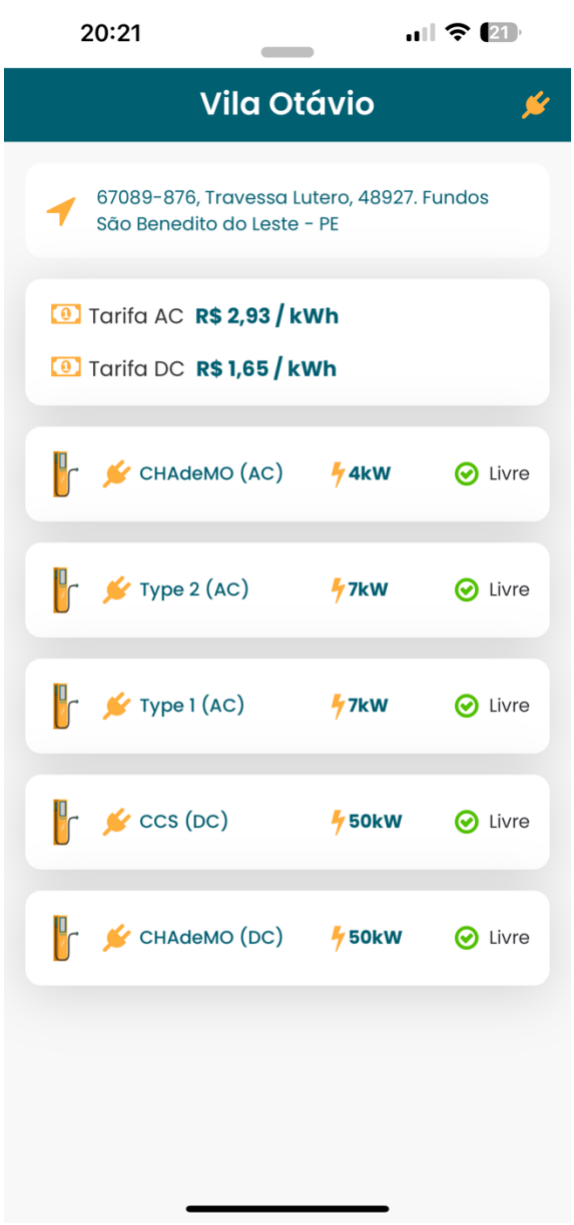
20:21

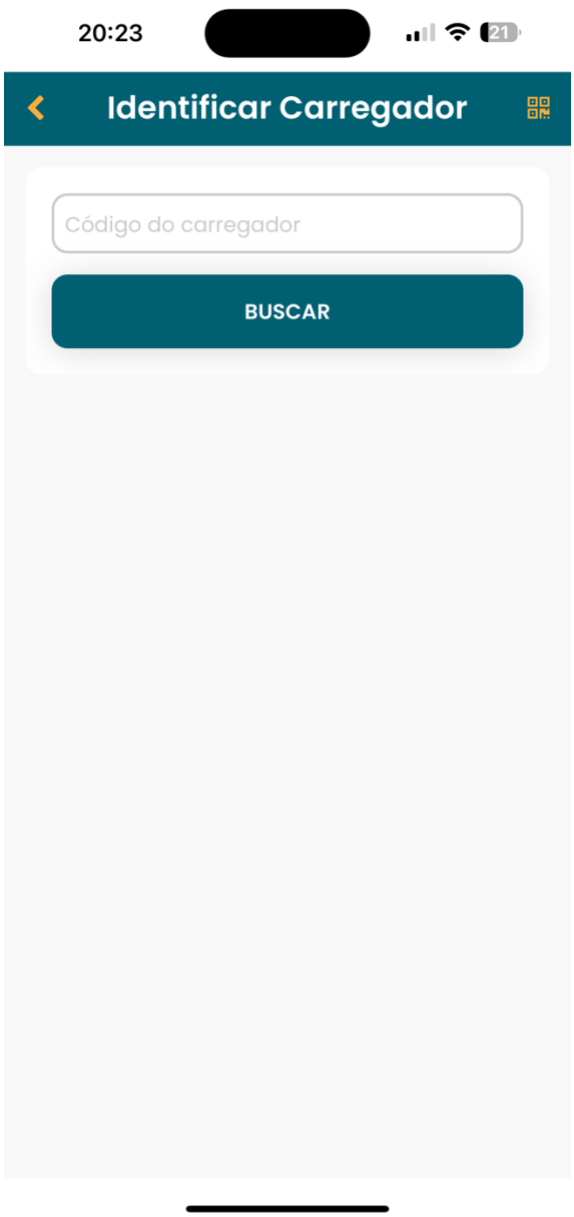
Obtendo sua localização...



20:21

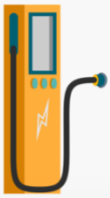



20:21
Vila Otávio
67089-876, Travessa Lutero, 48927. Fundos
São Benedito do Leste - PE
Tarifa AC R$ 2,93 / kWh
Tarifa DC R$ 1,65 / kWh
CHAdeMO (AC) 4kW Livre
Type 2 (AC) 7kW Livre
Type 1 (AC) 7kW Livre
CCS (DC) 50kW Livre
CHAdeMO (DC) 50kW Livre


20:23
Identificar Carregador
Código do carregador
BUSCAR

20:24

Ponto de Carga



Vila Otávio

ID **UPV8302607**

Conector **CHAdeMO**

Potência **4kW**

Tarifa **R$ 2,93/kWh**

Valor máximo: R$ 50,00

FULANO DE TAL
xxxx xxxx xxxx 4242
Valido até 12/2027

INICIAR RECARGA

20:24

Recarga

Carregando...

12,345kWh **R$ 20,37**

Início **28/08/2023 20:24**

Tempo decorrido **3h0m**

Limite retido **R$ 50,00**

Tarifa **R$ 1,65/kWh**

Vila Otávio

Carregador **UPV8302607**

CHAdeMO AC 4kW

Fulano de tal
xxxx xxxx xxxx 4242
Valido até 12/2027

PARAR RECARGA

20:22

Recarga

Concluído
Valor solicitado atingido.

43,234kWh	R$ 120,62
Início	16/04/2023 21:21
Fim	16/04/2023 21:47
Tempo decorrido	25m

Limite retido	R$ 140,00
Tarifa	R$ 2,79/kWh
São Jaqueline	
Carregador	111
CHAdeMO AC 4kW	

Fulano de Tal
xxxx xxxx xxxx 0002
Valido até 12/2036

20:22

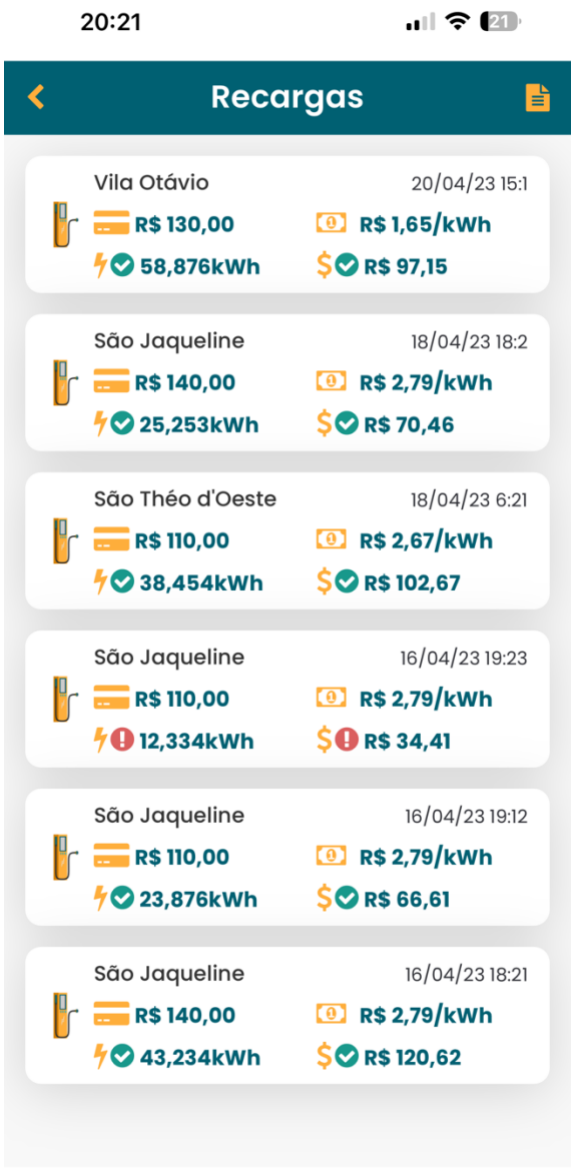
Recarga

Falha
O carregador foi desconectado.

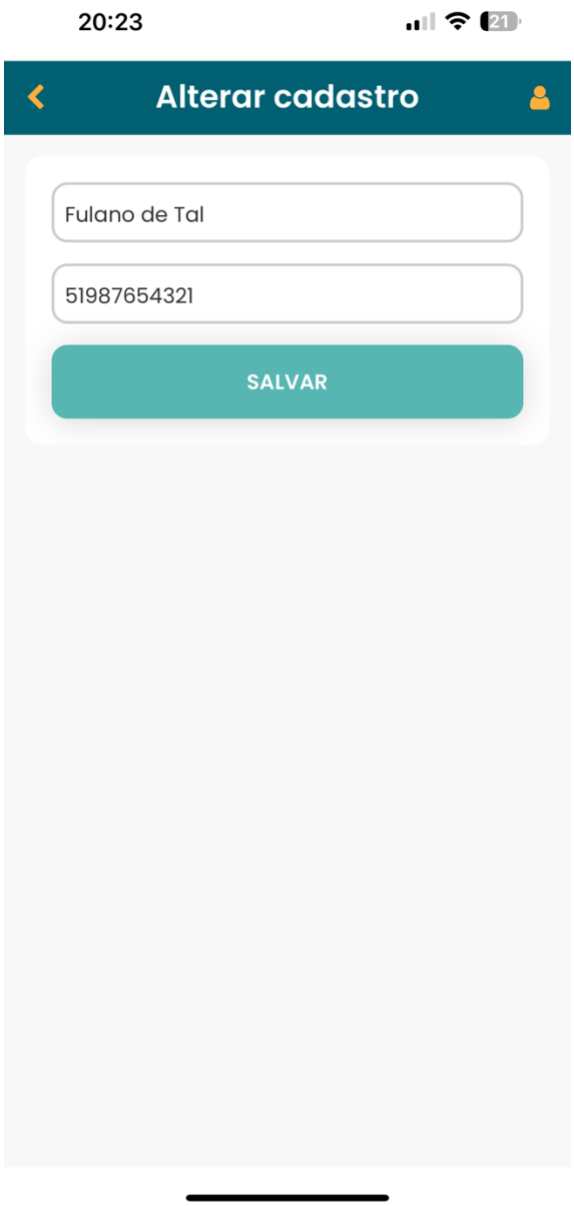
12,334kWh	R$ 34,41
Início	16/04/2023 22:23
Fim	17/04/2023 20:38
Tempo decorrido	22h15m

Limite retido	R$ 110,00
Tarifa	R$ 2,79/kWh
São Jaqueline	
Carregador	111
CHAdeMO AC 4kW	

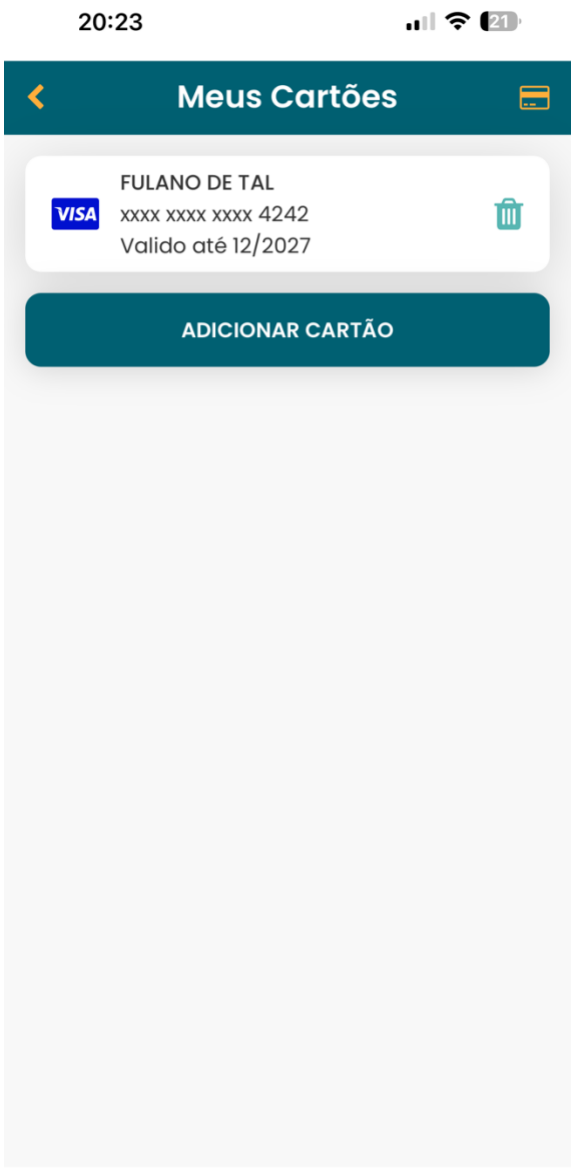
Fulano de Teste

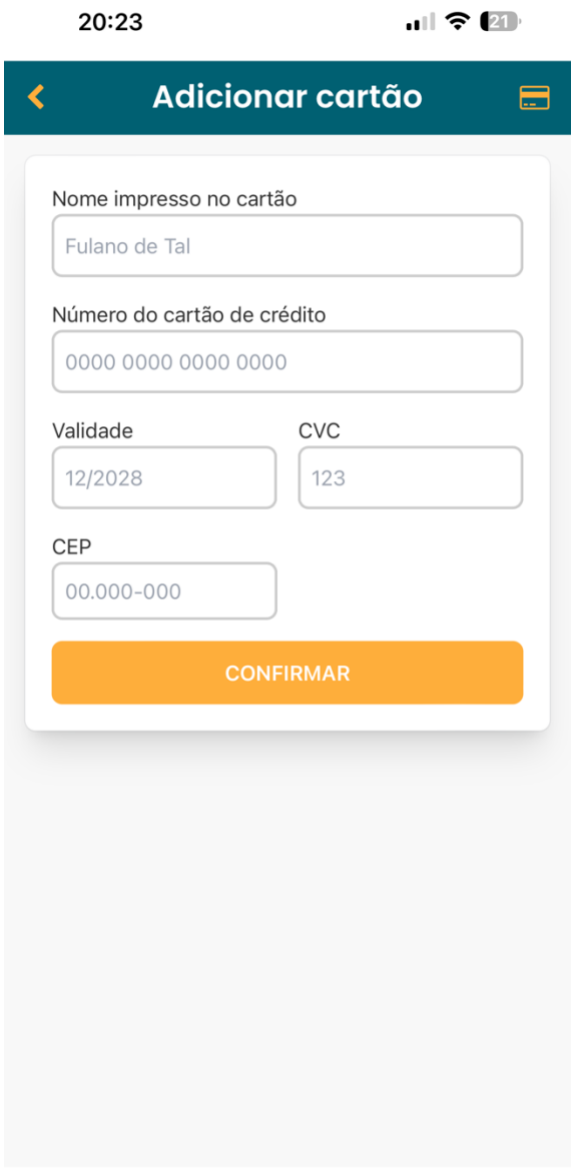

20:21
Recargas
Vila Otávio
20/04/23 15:1
R$ 130,00
R$ 1,65/kWh
58,876kWh
R$ 97,15
São Jaqueline
18/04/23 18:2
R$ 140,00
R$ 2,79/kWh
25,253kWh
R$ 70,46
São Théo d'Oeste
18/04/23 6:21
R$ 110,00
R$ 2,67/kWh
38,454kWh
R$ 102,67
São Jaqueline
16/04/23 19:23
R$ 110,00
R$ 2,79/kWh
12,334kWh
R$ 34,41
São Jaqueline
16/04/23 19:12
R$ 110,00
R$ 2,79/kWh
23,876kWh
R$ 66,61
São Jaqueline
16/04/23 18:21
R$ 140,00
R$ 2,79/kWh
43,234kWh
R$ 120,62


20:23
Meu cadastro
Fulano de Tal
fulano@example.com
51987654321
ALTERAR DADOS
TROCAR SENHA
MEUS CARTÕES
SAIR


20:23
Alterar cadastro
Fulano de Tal
51987654321
SALVAR


20:23
Trocar senha
Senha atual
Nova senha
Confirmação nova senha
SALVAR


20:23
Meus Cartões
FULANO DE TAL
xxxx xxxx xxxx 4242
Valido até 12/2027
ADICIONAR CARTÃO


20:23
Adicionar cartão
Nome impresso no cartão
Fulano de Tal
Número do cartão de crédito
0000 0000 0000 0000
Validade
12/2028
CVC
123
CEP
00.000-000
CONFIRMAR

Carga Rápida e Segura

Digite seu e-mail para receber um código de redefinição de senha:

E-mail de cadastro

ENVIAR