Escola Politécnica da Universidade de São Paulo

Adriano Antonio da Silva
Marcos Paulo Pacifico
Pedro Generoso Vique Dantas

Desenvolvimento de um aplicativo para mobilidade interna na Cidade Universitária Armando de Salles Oliveira: BusCar

São Paulo
2024

Adriano Antonio da Silva
Marcos Paulo Pacifico
Pedro Generoso Vique Dantas

Desenvolvimento de um aplicativo para mobilidade interna na Cidade Universitária Armando de Salles Oliveira: BusCar

Trabalho de conclusão de curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do bacharelado em Engenharia

São Paulo
2024

Adriano Antonio da Silva
Marcos Paulo Pacifico
Pedro Generoso Vique Dantas

Desenvolvimento de um aplicativo para mobilidade interna na Cidade Universitária Armando de Salles Oliveira: BusCar

Trabalho de conclusão de curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do bacharelado em Engenharia

Área de concentração:
Engenharia Elétrica com Ênfase em Computação

Orientador:
Jorge Luis Risco Becerra

São Paulo
2024

RESUMO

O presente trabalho teve como motivação inicial a ineficiência dos ônibus circulares na Cidade Universitária, que não possuem a frota necessária nem o número de percursos necessários para realizar de forma satisfatória a mobilidade interna dos estudantes. Utilizando o método conhecido como *Design Thinking,* buscou-se aprofundar no contexto e no problema apresentado, para então gerar ideias que poderiam ser prototipadas e implementadas para solucionar este problema apresentado. Com isso, obteve-se o objetivo deste trabalho, que é o desenvolvimento de um aplicativo *mobile* capaz de aproveitar os recursos existentes dentro do campus, a fim de otimizar a mobilidade interna dos seus estudantes e funcionários. O aplicativo intitulado como "BusCar" tem a premissa de unificar os meios de transporte existentes, como carros particulares, ônibus circulares, transporte coletivos (ônibus convencionais que circulam na Cidade Universitária, mas sem vínculo com a Prefeitura da Universidade de São Paulo - PUSP) e bicicletas, para que os estudantes e funcionários possam obter o trajeto ótimo até seus destinos. Sabendo da limitação de tempo presente em um TCC, optou-se por uma primeira versão da plataforma que utiliza dos carros particulares já existentes no campus para realizar caronas solidárias aos alunos que utilizam ônibus circulares ou se locomovem a pé. Esta primeira versão e seu protótipo derivado foram desenvolvidos utilizando-se de técnicas e conhecimentos aprendidos em sala, juntamente com outros estudos feitos sobre Engenharia e Desenvolvimento de Software. Com essa solução inicial já é possível amenizar o problema da mobilidade interna e dar os primeiros passos para este aplicativo ideal, que é capaz de trazer para a Cidade Universitária Armando de Salles Oliveira os princípios de uma "*Smart City".*

Palavras-chave: Aplicativo de caronas. Engenharia de software. Desenvolvimento mobile. Cidades Inteligentes.

ABSTRACT

The present work was initially motivated by the inefficiency of circular buses in the University City, which do not have the necessary fleet nor the number of routes necessary to satisfactorily carry out the internal mobility of students. Using the method known as Design Thinking, it was possible to dive deeper into the context and the problem presented, to then generate ideas that could be prototyped and implemented to solve or alleviate this problem presented. With this in mind, the objective of this work was achieved, which is the development of a mobile application capable of taking advantage of existing resources within the campus, in order to optimize the internal mobility of its students and employees. The application titled as “BusCar”, has the premise of unifying existing means of transport, such as private cars, circular buses, public transport (conventional buses that circulate in the University City, but without links to the City Hall of the University of São Paulo) and bicycles, so students and staff can get the optimal route to their destinations. Knowing the time constraints present in this project, it was decided to start with a first version that uses private cars already on campus to provide carpools for students who use shuttle buses or travel on foot. This first version and the following prototype were developed using techniques and knowledge learned in class, along with other studies carried out on Software Engineering and Development. Therefore, it is possible to alleviate the problem of internal mobility, and take the first steps towards this ideal application, which is capable of bringing the principles of a “Smart City” to Cidade Universitária Armando de Salles Oliveira.

Keywords: Carpooling app. Software Engineering. Mobile development. Smart Cities.

SUMÁRIO

1. Introdução

Para o início do projeto, a principal tarefa foi a compreensão do contexto para que se chegasse à solução proposta, capaz de amenizar o problema da mobilidade interna no campus, que por muitas vezes é demorada e gera estresse aos membros da comunidade USP. Para isto, utilizou-se de alguns métodos conhecidos no mercado, como Design Thinking e Design Science e algumas técnicas aprendidas ao longo do tempo de graduação, como Engenharia de Requisitos, para entender o problema, sua relevância e se a solução proposta o resolveria de forma satisfatória. As etapas constituintes do entendimento do contexto e da elaboração de uma solução estão presentes nos capítulos a seguir.

1.1 Motivação

O problema identificado pelo grupo, como já exposto de forma breve anteriormente, se apresentou na vivência dos alunos dentro da Cidade Universitária Armando Salles de Oliveira (CUASO) e nos problemas de mobilidade que estudantes se deparam no dia-a-dia, e assim como se comprova na publicação da Adusp (publicação online)[1].

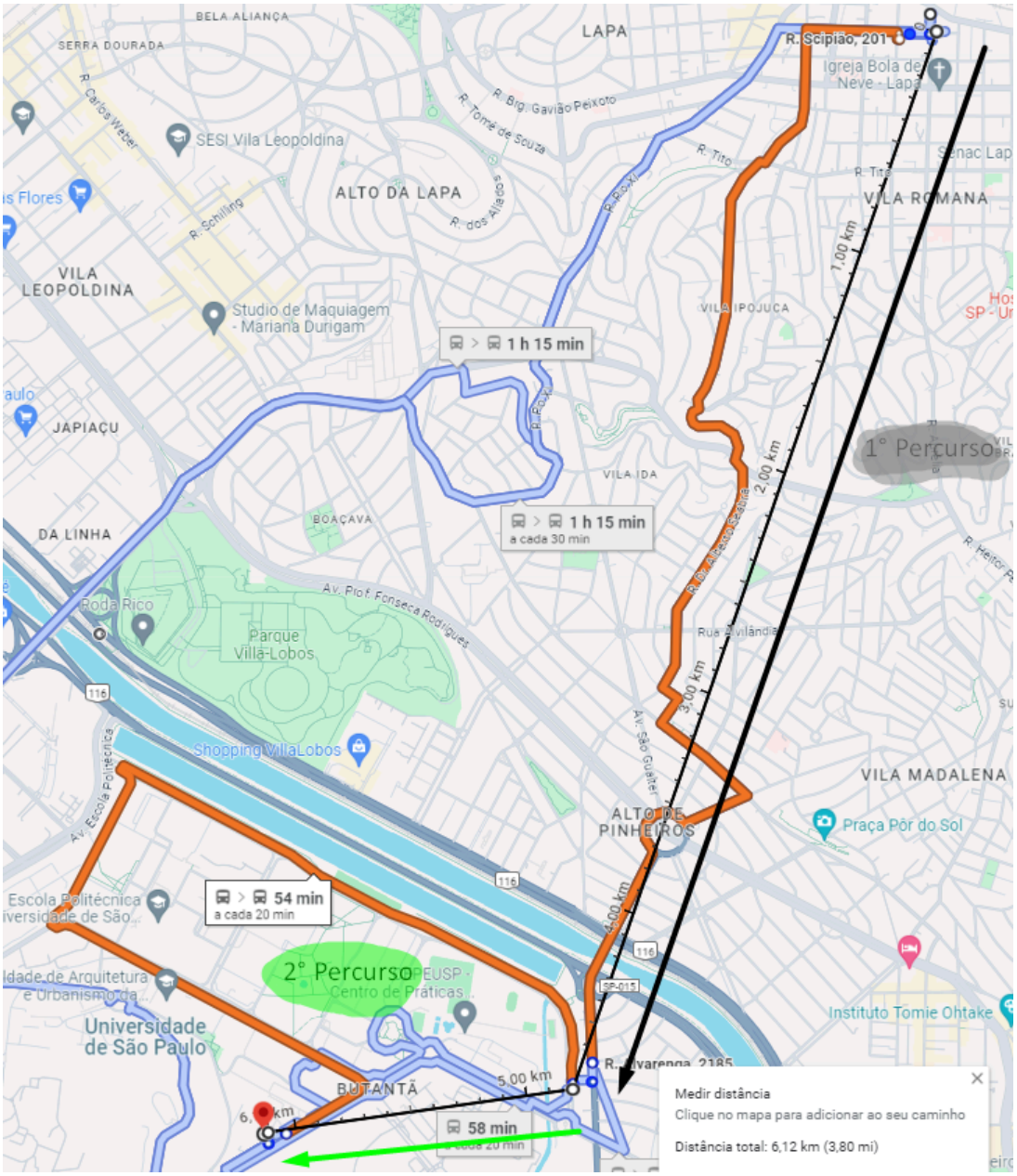
Um exemplo mais visual do problema a ser tratado é encontrado utilizando a ferramenta do *Google Maps,* amplamente utilizada e reconhecida, que estipula com bastante precisão o tempo do percurso realizado entre dois pontos. Abaixo seguem capturas da tela que ajudam a entender a proporção desigual entre o tempo de trajeto até os limites da USP e o tempo gasto se locomovendo dentro do campus.

A figura 1 mostra o exemplo de um estudante que mora nos arredores do bairro da Lapa, na zona Oeste da capital de São Paulo, e tem uma aula começando às 7:20h. As setas mostram a distância vetorial entre estes dois pontos e os percursos em laranja são os recomendados pelo *Google Maps* utilizando os ônibus públicos. A seta preta (1° Percurso) evidencia a grande distância entre a Lapa e a entrada da USP, enquanto a seta verde clara (2° Percurso) evidencia a pequena distância entre esta entrada e o Instituto de Química da USP (IQ). Nesta imagem é

[1] Prefeitura do Câmpus da Capital recua de mudanças nos circulares e abre “consulta pública”; proposta inicial, que seria implantada no reinício das aulas, foi questionada pela comunidade. Adusp, 2024. Disponível em: https://adusp.org.br/mobilidade-urbana/circulares-consulta/. Acesso em: 05 mai. 2024

possível identificar que o 2° percurso não é ótimo, visto que o circular 8022-10[2] dá uma grande volta para realizar este pequeno deslocamento (e é o único disponível para este trajeto).

Figura 1 - Caminho sugerido da Lapa até IQ

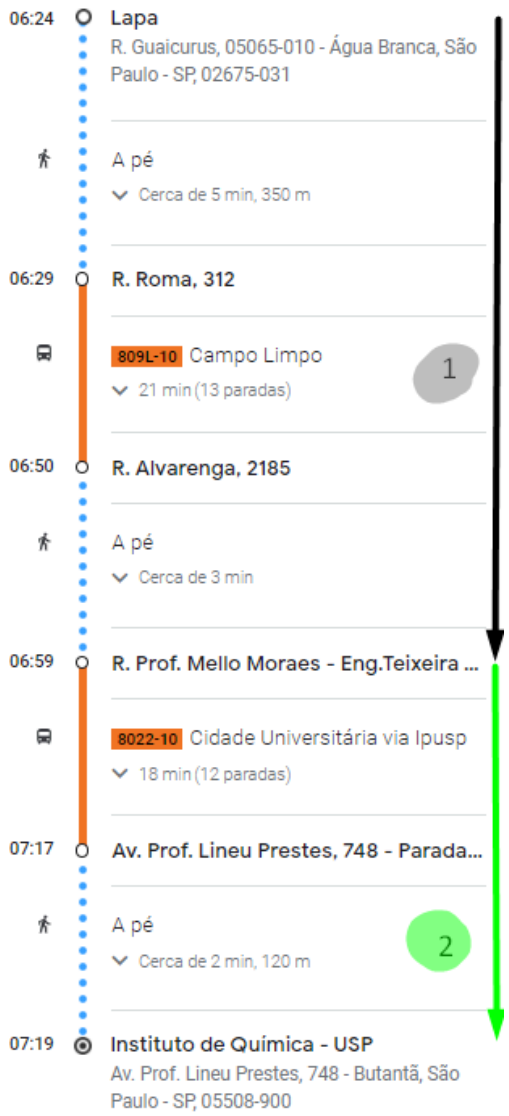


Fonte: elaborada pelos autores

[2] As informações do cálculo foram escritas quando os ônibus circulares seguiam este trajeto, porém no meio de 2024 houve alteração das rotas. Ainda assim, o problema persistiu, como foi comunicado por diversos estudantes.

A figura 2 traz os horários estimados para fazer este percurso, pensando neste aluno que precisa chegar no IQ às 7:20h. Nela, observa-se que o trajeto todo leva em torno de 55 minutos para ser realizado, e que destes, cerca de 20 minutos são gastos desde que o aluno chega no ponto do 8022-10 até o momento que chega no Instituto das Químicas.

Figura 2 - Tempo gasto no caminho sugerido



Fonte: elaborada pelos autores

Fazendo uma conta simples, podemos chegar na velocidade média para realizar estes dois percursos, e assim, averiguar o quão lento é o transporte interno. O percurso 1 tem uma distância vetorial de aproximadamente 4,8 km e o percurso 2 tem cerca de 1,3 km de distância vetorial, assim como mostra a escala na figura 1. A

velocidade média para percorrer esta distância se dá dividindo a distância pelo tempo gasto. Sendo assim, o percurso 1 é realizado em uma velocidade média de 8,2 km/h, enquanto o percurso 2 é realizado em uma velocidade média de 3,9 km/h. Portanto, fica evidente que neste cenário o aluno demora cerca do dobro de tempo para circular internamente pelo campus.[3]

O público-alvo, estudantes e funcionários sem transporte privado motorizado, se vê obrigado a escolher dentre poucos métodos disponíveis para se locomover, dentre eles: caminhada; transportes não motorizados (tal qual bicicletas e afins); e o transporte público, representado principalmente pelos ônibus circulares que oferecem transporte gratuito para membros da Universidade de São Paulo (USP).

Como grande parte dos estudantes acaba se voltando para o uso dos circulares, o foco no uso do transporte coletivo público acaba gerando diversos problemas pros usuários majoritários desse transporte. Dentre as principais consequências do problema de mobilidade identificado, estão:

- A superlotação de circulares e criações de filas em horários de pico comuns a estudantes, como almoço, chegada e saída das aulas, que geram desconforto para os usuários e atrasos na circulação;
- Atrasos nos circulares devido a limitação de frota e períodos de rotatividade;
- Redução do rendimento acadêmico para os afetados, como abordado no artigo sobre permanência de estudantes de baixa renda[4].

1.2 Objetivo

Identificado o problema existente e suas consequências para a comunidade USP, buscou-se encontrar uma solução viável e eficaz, seguindo os conceitos da metodologia "*Design Science"*, para extingui-los ou pelo menos amenizá-los. Para chegar nesta solução, o grupo se valeu também do método *"Design Thinking",* dividido em quatro etapas (Imersão, Ideação, Prototipação, Desenvolvimento).

Desta maneira, chegou-se na solução do BusCar, uma plataforma *mobile* integradora das modais de transporte disponíveis no campus, que auxilia o usuário

[3] Vale ressaltar que este cenário citado é utilizando estimativas realizadas pelo *Google Maps*, e assim sendo, não engloba os casos extremos onde o 8022-10 pode demorar até 30 minutos para passar no ponto. O cenário utilizado para a conta considerou que o aluno esperou por 6 minutos o ônibus circular, como mostrado na figura 2.

[4] ALMEIDA, Wilson Mesquita de. Revisitando "USP para Todos?" : desafios permanentes na inclusão dos estudantes de baixa renda no ensino superior público brasileiro. . Revista de Ciências Sociais, [S. l.], v. 51, n. 3, p. 21–62, 2020. DOI: 10.36517/rcs.51.3.d02. Disponível em: http://www.periodicos.ufc.br/revcienso/article/view/54817. Acesso em: 6 maio. 2024.

passageiro a chegar ao seu destino com mais eficiência, entendendo o estado atual de todos estes meios de transporte e indicando ao passageiro qual a melhor rota.

Entretanto, a complexidade de um aplicativo deste porte é grande, o que não o torna viável de ser desenvolvido durante o período da disciplina de Projeto de Conclusão de Curso, visto que são muitos meios de transporte e muitas variáveis a serem levadas em conta. Assim sendo, o objetivo deste trabalho será o desenvolvimento de uma primeira versão deste aplicativo, que organiza, em tempo real, caronas de membros da comunidade que se deslocam com seus carros particulares, para outros membros da comunidade USP, que utilizam os demais meios de transporte.

Para essa primeira versão há duas etapas essenciais: a primeira etapa é o desenvolvimento detalhado do produto, assim como uma especificação técnica do funcionamento em alto nível; a segunda etapa é a implementação de um protótipo que comprova a viabilidade da solução. Essa implementação traz consigo algumas simplificações, porém ainda com as funcionalidades principais de "*match*" de passageiro e motorista e a notificação em tempo real do andamento da carona para os usuários.

1.3 Justificativa

A solução proposta para a primeira versão da plataforma, é a capacidade de organizar caronas dentre os membros da CUASO, como forma de aproveitar a frota de veículos já existente (comumente usados para transporte individual), incentivar o transporte rápido, confortável e diminuir o estresse sobre os demais modais de transporte. Como demais vantagens, seria possível coletar dados sobre o transporte na USP, que podem ser de grande utilidade acadêmica, financeira e social.

Para validar a solução, foram pesquisados trabalhos semelhantes já desenvolvidos, a fim de validar a atualidade desta monografia. Sendo assim, foi encontrado um outro TCC[5], com algumas especificidades de contexto, que mostra a viabilidade e vantagens de se implementar um sistema deste tipo. O trabalho citado mostra que a maioria dos alunos da Universidade Federal do Ceará tem interesse em ter uma forma alternativa de transporte (87,1%), e que a maioria dos alunos com

[5] SOUZA, Alex Sandro Alves de. Estudo de viabilidade para implantação de um aplicativo de carona compartilhada na Universidade Federal do Ceará do Campus Quixadá. 2023. 56 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Federal do Ceará, Campus de Quixadá, Quixadá, 2023.

carro próprio estaria disposta a oferecer caronas (74,2%).Com isso, identificamos que a solução é bem vista por estudantes que passam por problemas similares aos citados acima, contanto que seja adaptada ao contexto do Campus Butantã e suas especificidades.[6]

Um outro Trabalho de Conclusão de Curso[7] consultado, agora no campo da Engenharia Civil, estuda a influência de um aplicativo de carona na mobilidade urbana. Neste trabalho o autor conclui, por meio de análise de perfil e pesquisas, que a carona solidária (no caso o Blablacar) é algo bem visto do ponto de vista ambiental e social. Ele mostra que aplicativos de carona podem reduzir o impacto no deslocamento urbano ocasionado pelo transporte privado.

Para além do meio acadêmico, também valeu-se de estudos de alternativas semelhantes à proposta pelo grupo, que já foram implementadas e difundidas no mundo por empresas consolidadas como a Uber e 99. A solução mais próxima encontrada como referencial foi a funcionalidade Uber Pool, depreciada do aplicativo Uber, que tinha como função atribuir diferentes passageiros para um mesmo motorista de forma a otimizar a rota de ambos e permitir um preço acessível aos clientes.

Dado a semelhança com a proposta do grupo, foi importante entender os problemas que podem ter levado a eventual extinção do Uber Pool e como isso poderia guiar o projeto e avaliar sua viabilidade. O que foi possível elencar como hipóteses de falha são:

- Locais de embarque pré-definidos que poderiam levar passageiros a se perder, ou a lugares perigosos;
- Não oferecia vantagem financeira suficiente para motoristas e passageiros.
- A preferência dos clientes por viagens sem contato com desconhecidos e os possíveis riscos implicados;
- Dificuldade de encontrar rotas e passageiros usuários do serviço;
- Desvios na rota dos passageiros ocasionaram atrasos que estes não estavam dispostos a ter, principalmente em megalópoles como São Paulo, em que pequenos desvios podem significar grandes atrasos.

---

[6] O BusCar tem como premissa o desafogamento dos circulares, que são utilizados majoritariamente para locomoção dentro do campus e no máximo à estação Butantã. Por isso, ele acaba se diferindo do sistema citado por ajudar alunos que estão, por exemplo, nos pontos de ônibus, enquanto que o estudo feito no TCC citado ajuda nas caronas Campus-Rodoviária.

[7] LACERDA, Guilherme Mangueira. A influência do uso de aplicativo de carona na mobilidade urbana: um estudo de caso do Blablacar. 2023. Trabalho de Conclusão de Curso.

Com essas hipóteses, foi possível avaliar se a solução trazida neste trabalho levaria aos mesmos problemas. A conclusão do grupo foi que os problemas elencados não afetariam a solução proposta, justamente pelo contexto de aplicativo de caronas localizada que é voltado para uma comunidade fechada como a da universidade, assim como seguem:

- Os locais de embarque seriam locais conhecidos, com bastante circulação, de forma a garantir a segurança dos usuários;
- O passageiro não teria que pagar pela carona, e o motorista, além do benefício social que estaria providenciando, seria bonificado de forma não-monetária pela prefeitura do Campus para oferecer essas caronas;
- Todos os usuários do sistema pertenceriam à comunidade USP, garantindo mais segurança e oferecendo maiores possibilidades de socialização;
- Dada à coincidência de horários de locomoção na cidade universitária, como já mencionado, há grande potencial de uso e disponibilidade de viagens;
- Dado o escopo espacial e de funcionalidade que o projeto pretende desenvolver, o uso do serviço tem como objetivo aumentar a velocidade de chegada ao destino em relação às alternativas de transporte já existentes;

Portanto, é possível verificar a atualidade deste trabalho, visto que os trabalhos consultados são recentes e também carregam consigo a conclusão de que um aplicativo de caronas ameniza problemas de mobilidade urbana. Dado o contexto restrito do ambiente fechado e controlado da USP, os problemas enfrentados por aplicativos similares podem ser suprimidos.

1.4 Organização do Trabalho

Este trabalho está dividido da seguinte forma:

- No capítulo 2, são apresentados os aspectos conceituais para o desenvolvimento do trabalho;
- No capítulo 3, é apresentado o método de trabalho, isto é, as fases e processos que compuseram o desenvolvimento do trabalho;
- No capítulo 4, é apresentada a parte principal desta monografia, onde consta o desenvolvimento do trabalho. O desenvolvimento tem subdivisões que ajudam a separar este grande processo, como: especificação de requisitos, transformação dos requisitos em features, especificação das

features/arquitetura, implementação do protótipo e as tecnologias utilizadas, e por fim a avaliação e testes;

- No capítulo 6, são apresentadas as considerações finais deste trabalho, como: conclusões acerca do projeto, contribuições e as perspectivas de continuidade.

2. Aspectos Conceituais

2.1. Transporte Público Coletivo

Segundo o site ITDP (Instituto de Políticas de Transporte e Desenvolvimento, 2022) o transporte público e coletivo visa atender as demandas de locomoção dos passageiros para fins diversos, incluindo trabalho, lazer, educação e as oportunidades oferecidas pela cidade. Não somente isso, ele proporciona redução do uso de veículos particulares, impactando na quantidade de emissão de gases poluentes e trazendo benefícios para o meio ambiente e saúde local.

Entretanto, visto que a frota de ônibus disponível na universidade não é suficiente e que ainda há muitos carros com vagas livres circulando, uma alternativa e otimização do uso desses recursos seria um sistema de caronas compartilhadas.

2.2. Design Science

Segundo o artigo “Design Science Research: método de pesquisa para a engenharia de produção” (LACERDA et al., 2013), para que uma pesquisa seja reconhecida ela deve apresentar que foi desenvolvida com rigor e passível de debate e verificação. Com base no que foi pesquisado e seguindo orientações a equipe decidiu por desenvolver esse trabalho de pesquisa utilizando a metodologia de Design Science.

De acordo com o artigo técnico “Ciência do Artificial e Design Science Research” (REIS, 2019), Design Science Research (DSR) é uma abordagem metodológica que visa a criação e avaliação de artefatos destinados a resolver problemas práticos, além de contribuir para o conhecimento científico. Introduzida por Herbert Simon, a DSR enfoca a construção de objetos artificiais (artefatos) cuja criação é baseada em métodos científicos. A metodologia é iterativa, integrativa e multidisciplinar, permitindo a análise contínua e a adaptação dos artefatos para atingir objetivos específicos. A aplicação da DSR envolve várias etapas: identificação do problema, definição dos objetivos da solução, design e

desenvolvimento do artefato, demonstração, avaliação, comunicação dos resultados e iteração. Este processo garante que as soluções sejam eficazes e cientificamente válidas.

Diretrizes para Aplicação da DSR - Design Science Research (Hevner et al., 2004)**:**

- **Relevância do Problema**: Garanta que o problema abordado seja relevante tanto para a prática quanto para a teoria.
- **Contribuição à Pesquisa**: A solução deve contribuir para o corpo de conhecimento existente.
- **Rigor Metodológico**: Utilize métodos científicos rigorosos na construção e avaliação do artefato.
- **Design como Artefato**: O artefato deve ser claramente definido e documentado.
- **Avaliação**: Avalie a eficácia e eficiência do artefato de forma rigorosa.
- **Pesquisa Iterativa**: O processo deve ser iterativo, permitindo melhorias contínuas.
- **Comunicação**: Os resultados devem ser comunicados de forma clara e compreensível para ambos os públicos, acadêmico e prático.

Para aplicar a DSR, começa-se pela definição do contexto, compreendendo o ambiente de aplicação e identificando problemas específicos. Em seguida, é necessário coletar dados para entender as causas do problema, desenvolver modelos representativos e protótipos para testar soluções em um ambiente controlado. A análise e o feedback dos testes permitem ajustes no artefato, e após validação, a solução é implementada em escala maior com monitoramento contínuo de desempenho. Seguindo essas etapas e diretrizes, a metodologia DSR pode ser aplicada de forma eficaz para desenvolver soluções práticas e robustas que também contribuem para o avanço do conhecimento científico.

Design thinking

Segundo ARCHILLI (2024), “design thinking” é definido como uma abordagem que usa métodos de designers para resolver problemas, além de se evitar uma

postura excessivamente planejadora e focar na execução das atividades. Alguns métodos são pesquisa, brainstorms, seleção de ideias e prototipagem. Tendo isso em mente a equipe decidiu por escolher esse método como guia de execução e composição do protótipo do aplicativo.

O blog “3pm3” (SOMOS 3 INTERNET S.A, 2024) também considera a abordagem voltada para resolução de problemas complexos visando solucionar problemas relacionados a um determinado público alvo. A metodologia como pode ser encontrada em diversas fontes é baseada em empatia, colaboração e experimentação. Juntas fundamentam as ideias que o processo de criação deve entender muito bem o contexto de aplicação, usando da colaboração para ampliar a inovação e validar as hipóteses por experimentação.

As quatro etapas do design thinking são: Imersão, Ideação, Prototipação e Desenvolvimento. A etapa de imersão é fundamental, pois é o momento em que o desenvolvedor busca compreender profundamente o contexto e o problema que deseja resolver. Nessa fase, é importante identificar as forças, fraquezas, ameaças e oportunidades da solução. Além disso, é importante focar em descobertas, investigando e descobrindo informações relevantes sobre o mercado, stakeholders e alternativas às soluções. Coletar feedbacks dos usuários também é essencial para entender suas necessidades e expectativas. O objetivo é reunir o máximo de informações possíveis para ter uma visão clara e abrangente que permita a criação de soluções alinhadas com os objetivos do produto em questão.

Na fase de ideação, o foco é gerar ideias criativas e inovadoras. Aqui a equipe deve realizar sessões de brainstorming, onde todos podem contribuir. Isso estimula a criatividade e a colaboração entre os participantes. Usar as informações coletadas na fase de imersão para direcionar as ideias para as áreas que precisam de melhoria é essencial. A ideia é criar um ambiente onde as pessoas se sintam confortáveis para compartilhar suas opiniões, resultando em um fluxo rico de ideias que podem ser desenvolvidas.

A prototipação é a fase de testes, onde você transforma as melhores ideias em versões preliminares para validação. Aqui a equipe deve reunir as ideias com maior potencial e chance de sucesso, desenvolvendo versões de teste (protótipos) do produto ou funcionalidade. Esses protótipos não são versões finais, mas sim ferramentas para validar e ajustar as ideias. Durante essa fase, é fundamental testar os protótipos com os usuários e demais stakeholders, coletando feedbacks para

aprimorar a ideia inicial. O objetivo é aprender com os feedbacks e ajustar o protótipo até que ele se aproxime do resultado esperado.

Após validar a solução durante a prototipação, é hora de desenvolver a versão final do produto. Nesta fase, a equipe deve implementar todas as melhorias identificadas durante a fase de prototipação, garantindo que o produto final atenda às expectativas dos usuários. Concluir a versão oficial do produto é crucial, e preparar uma estratégia de go-to-market para introduzir o produto no mercado é a última etapa. Esta fase é sobre concretizar o trabalho realizado nas fases anteriores e garantir que a solução desenvolvida seja implementada de forma eficaz e bem-sucedida.

2.3. BPMN

O padrão de Modelagem e Notação de Processos de Negócios (BPMN) foi escolhido para modelar o projeto, padrão desenvolvido pelo Object Management Group (OMG, 2010). O principal objetivo do BPMN segundo a fonte criadora OMG é fornecer uma notação que seja facilmente compreendida por todos os usuários de negócios, desde o grupo que criou os rascunhos iniciais dos processos, até a etapa de desenvolvimento técnico responsável por implementar a tecnologia que executará esses processos, e, finalmente, para a etapa de negócios que gerenciará e monitorará esses processos. Assim, o BPMN cria uma ponte padronizada para preencher a lacuna entre o design do processo de negócios e a implementação do processo.

Fazer um diagrama BPMN ajuda a obter mais clareza sobre os processos de negócios, consequentemente, melhorando a comunicação, a colaboração e entendimento entre a equipe. Um diagrama BPMN é um tipo de fluxograma que usa ícones padronizados para representar os diferentes elementos e fluxos de um processo de negócios.

2.4. Arquitetura em Camadas

Segundo o livro “Engenharia de Software Moderna” (VALENTE, 2020), uma definição para arquitetura de software considera que arquitetura preocupa-se com projeto em mais alto nível. O foco deixa de ser a organização e interfaces de classes individuais e passa a ser em unidades de tamanho maior, como pacotes, componentes, módulos, camadas ou serviços. Esses componentes arquiteturais não

são apenas maiores em escala, mas também devem ser relevantes para alcançar os objetivos do sistema. Por exemplo, em um sistema de informações, o módulo de persistência é essencial para automatizar e manter informações de processos de negócio.

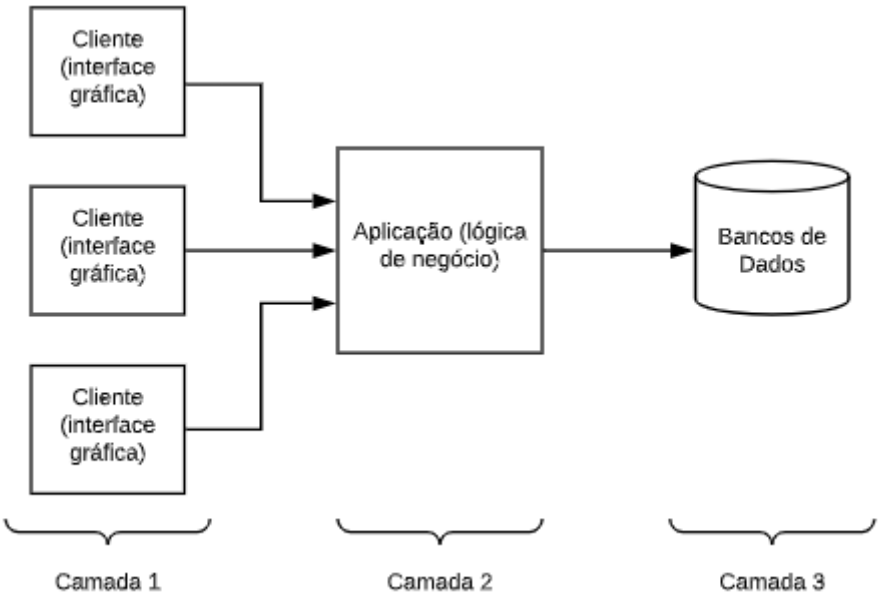
A arquitetura em camadas é um dos padrões mais utilizados desde os primórdios do desenvolvimento de sistemas de software complexos nas décadas de 60 e 70. Nesse padrão, as classes são agrupadas em módulos maiores conhecidos como camadas, organizadas hierarquicamente, de forma semelhante a um bolo. Cada camada utiliza serviços da camada imediatamente inferior, permitindo um controle estrito das dependências entre elas. Isso simplifica o desenvolvimento, facilita a manutenção e promove a evolução do sistema. Além disso, favorece a substituição de camadas individuais, como mudar de TCP para UDP, e promove o reuso de camadas por múltiplas camadas superiores, como a camada de transporte sendo utilizada por vários protocolos de aplicação como HTTP, SMTP, DHCP, entre outros.

As três camadas dessa arquitetura são compostas por:

- Interface com o Usuário, também chamada de camada de apresentação. Ela trata tanto da exibição de informação, como da coleta e processamento de entradas e eventos de interfaces, tais como cliques em botões, marcação de texto, etc. A camada de interface pode ser uma aplicação desktop, em Windows ou outro sistema operacional com interface gráfica, como também Web.
- Lógica de Negócio, também conhecida como camada de aplicação, implementa as regras de negócio do sistema. A camada de aplicação seria responsável por receber os dados inseridos pelo usuário. Ela validaria esses dados de acordo com as regras de negócio estabelecidas, como, por exemplo, garantir que eles estejam dentro de um intervalo válido (ex: maior ou igual a zero e menor ou igual ao valor de avaliação). Após a validação, a camada de aplicação poderia acionar a camada de lógica para verificar a conformidade com essas regras
- Banco de Dados, que armazena os dados manipulados pelo sistema. Por exemplo, após o lançamento e validação dos dados, eles são salvos em um banco de dados.

Normalmente, uma arquitetura em três camadas é uma arquitetura distribuída. Isto é, a camada de interface executa na máquina dos clientes. A camada de negócio executa em um servidor, muitas vezes chamado de servidor de aplicação. E, por fim, temos o banco de dados que pode ser local ou em nuvem. A figura a seguir mostra um exemplo, que assume que a interface oferecida aos clientes é uma interface gráfica.

Figura 3 - Exemplo de arquitetura em camadas



Fonte: https://engsoftmoderna.info/cap7.html

2.5. Plataforma de Experiência Digital

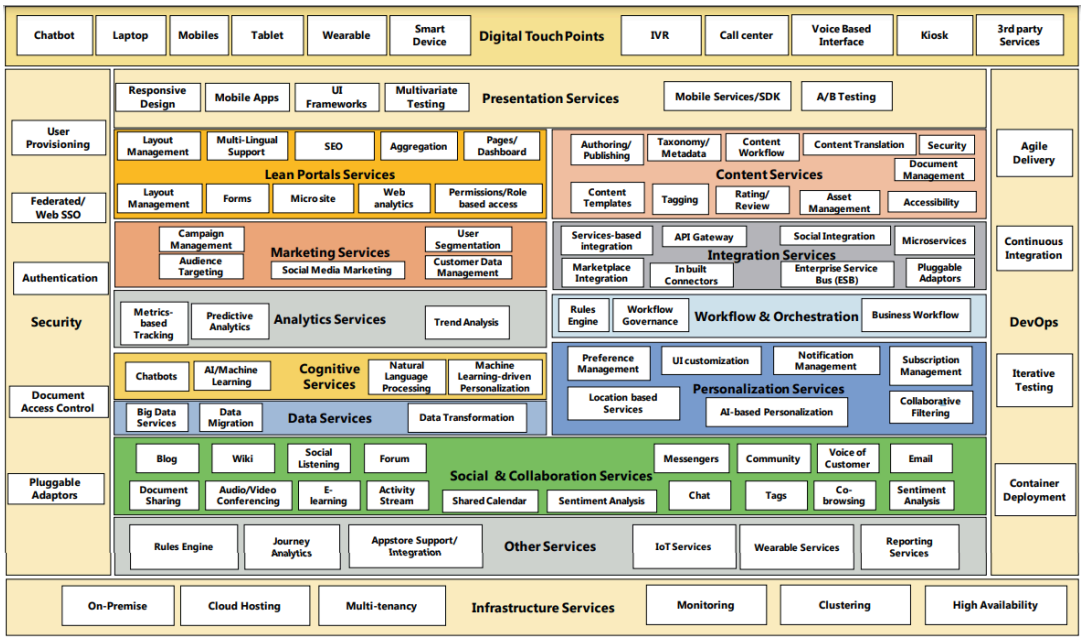
Segundo Shivakumar e Sethii (2019), as Plataformas de Experiência Digital (DXPs) são um conjunto de ferramentas tecnológicas que ajudam a conhecer o risco do uso das tecnologias e criar plataformas escaláveis com tecnologias modernas dentro da arquitetura de camadas. As DXPs são principalmente plataformas de engajamento centradas no usuário que fornecem uma visão unificada, com interface de usuário rica para uma experiência aprimorada do usuário final.

Ainda seguindo a visão dos autores, o principal benefício das plataformas digitais é seu enfoque na integração entre diferentes tecnologias e partes interessadas, tão usual em plataformas web hoje em dia. Por exemplo, o fluxo de um usuário entre diversas ferramentas interligadas como uma transferência digital de PIX, na qual, após comprovado o pagamento, o usuário pode facilmente compartilhar pelo whatsapp o comprovante de pagamento. De forma ininterrupta e

simples, o usuário transita entre duas aplicações diferentes e não diretamente relacionadas, mas integradas dentro do contexto aqui explicado.

Para chegar em uma plataforma de experiência digital, existe uma arquitetura de referência que pode ser seguida, e que contempla os principais pontos existentes. A figura abaixo mostra essa arquitetura, e em seguida, têm-se a explicação dos principais componentes de cada camada.

Figura 4 - Arquitetura de referência de uma DXP



Fonte: SHIVAKUMAR, S. K.; SETHII, S. Building Digital Experience Platform: A Guide to Developing Next-Generation Enterprise Applications. California: Apress Berkeley, 2019

- Pontos de contato do usuário: são dispositivos como smartphones, pontos onde o usuário se conecta com o serviço.
- Serviços de apresentação: interface e experiência de usuário.
- Serviços de portal Lean: serviços de apresentação complementares para experiência de usuário personalizados, como indicação de músicas no Spotify.
    - Controles simples para gerenciar páginas;
    - Visão centralizada do serviço personalizado;
    - Análise do comportamento do usuário no site;
- Serviços de Conteúdo e serviços de gerenciamento de conteúdo: como autoria de conteúdo.

    - Fornecem ferramentas de gerenciamento de conteúdo durante o ciclo de vida do conteúdo (criação, alteração, exclusão e qualquer edição);
- Campanha e Serviços de marketing: um dos principais recursos da DXP são campanhas de marketing. A DXP usa dados do usuário para direcionar melhor o público das publicidades.
- Serviços de Análises: análise web usando métricas predefinidas, análise de tendências e análise preditiva.
- Serviços de integração: componente mais importante de uma DXP. A plataforma deve ser capaz de integrar, escalabilizar com flexibilidade e extensão. Exemplos: *REST, JSON* e *ERP*.
    - A integração melhora a produtividade do usuário final e otimiza o retorno do lucro;
    - A integração é realizada através de padronização do uso das tecnologias;
- Serviços Sociais e de Colaboração: ferramentas de troca de experiência entre os usuários, como fóruns, blogs, wikis.
- Fluxo de trabalho e orquestração: as DXPs permitem projetar e implementar processos de negócios ágeis, automatizados e dinâmicos por meio de modelagem de fluxo de trabalho, um mecanismo de regras configurável e governança de fluxo de trabalho.
- Serviços de Pesquisa: as DXPs também oferecem suporte a recursos de pesquisa avançados, como filtragem de resultados ou pesquisa facetada.
    - Aumentam a produtividade do usuário final através de descoberta de informação eficiente;
- Serviços comerciais (opcional): serviços comerciais como gerenciamento de pedidos (a DXP pode gerenciar os pedidos de uma loja, por exemplo).
- Serviços Cognitivos (opcional): recomendações baseadas em IA e machine learning.
- Serviços de dados (opcional): serviços de gerenciamento de dados.
- Serviços de infraestrutura: recursos de suporte, como implantação em nuvem e conteinerização.
- Serviços de Personalização: serviços que personalizam a DXP de acordo com o usuário. Inclui a possibilidade de alteração do layout de página até recomendação de produtos e gerenciamento de assinaturas.

- Segurança: recursos de autenticação e autorização.
- DevOps: integração e implantação contínuas, testes iterativos e utilização de *Containers* .

3. Método de Trabalho

Com a percepção de que os meios de transporte em vigência são insuficientes e é preciso melhorar o deslocamento na Cidade Universitária, foi necessária uma aproximação científica sobre o problema, a fim de entendê-lo para além da percepção e achismos. Portanto foi necessário utilizar como base de pesquisa a metodologia de Design Science, o que implica nessa fase, reconhecer o problema e entender intimamente as dificuldades associadas a ele (o que no Design Thinking é também conhecido como etapa de imersão) e entender a relevância desse problema pra sociedade, quem ele afeta, como afeta e se há modo de mudança. Todo esse processo se deu por meio: 1) da leitura de artigos que comprovassem a existência do problema, 2) o entendimento sobre os movimentos universitários recentes como a rediscussão do plano diretor da Cidade Universitária 3) a recém mudança das rotas dos circulares e pesquisas de opinião sobre essas mesmas rotas promovidas pela Prefeitura do Campus Butantã, 4) a percepção de outros alunos e também do conhecimento íntimo e cotidiano dos próprios membros da equipe que enfrentaram esses mesmos problemas de forma recorrente ao longo de suas graduações.

Portanto, se utilizando desse princípio reconhecido no mercado, verificou-se de forma rigorosa não só a relevância do problema que afeta milhares de estudantes, funcionários e professores todos os dias, mas que havia potencial para uma solução de engenharia e tecnologia capaz de ativamente afetar e solucionar esse problema.

Tendo essa visão sobre o problema mais clara, iniciou-se o processo de levantamento de hipóteses sobre como uma solução tecnológica poderia melhorar a qualidade de deslocamento na universidade, especificamente no campus Butantã. Nessa fase, que na metodologia de Design Thinking é chamada de ideação, foram levantadas diversas ideias pela equipe, das quais cada uma, foram avaliados o potencial de mudança no campus, vantagens e desvantagens relevantes para a questão. Dentro das que mais atendiam essas expectativas, foram feitas pesquisas de soluções estados da arte similares às ideias e que poderiam comprovar seu

potencial e viabilidade. No final desse processo, alcançou-se a ideia atual do BusCar, de uma plataforma multimodal, uma solução integradora que dá ao usuário da plataforma uma visão ampla e em tempo real dos diversos recursos disponíveis no campus(oferecendo acompanhamento em tempo real de ônibus, locais de bicicleta e disponibilidade, tempo de deslocamento, tudo isso em uma única plataforma que unifica essas soluções) e não só isso, otimiza esses recursos para o usuário, oferecendo possibilidades que antes não estariam disponíveis, como a funcionalidade de organizador de caronas.

Essa solução, ao unificar todas essas e possivelmente outras soluções relacionadas ao deslocamento em uma única plataforma de fácil acesso ao usuário permite: 1)uma redistribuição melhor de recursos de mobilidade na cidade universitária, 2)uma visão ampla de métodos de transporte que não são normalmente considerados e facilidade do usuário de acessar esses meios. Isso, por sua vez, contribui para o conforto dos usuários, velocidade em deslocamento, contribui para uma visão de transporte mais condizente com a crescente preocupação com o meio ambiente e também com a saúde do usuário.

Percebendo essa ser a solução mais condizente com o problema e de maior potencial, portanto, iniciou-se o projeto mais rigoroso do que essa solução deveria ser na prática e para isso, entendeu-se que a primeira versão de um projeto dessa magnitude precisaria de um escopo menor, dado o tempo e restrições de equipe da disciplina de Projeto de Conclusão de Curso. Portanto, a equipe acabou decidindo projetar para a primeira versão apenas o recurso de organizador de caronas, dada que esse recurso sozinho já demonstrava ser um grande catalisador de mudanças na mobilidade do campus e oferecia um playground no qual as principais características do BusCar poderiam ser já visualizadas.

Visto isso, o processo de projetar a versão inicial do BusCar se utilizou de diversas técnicas já conhecidas na área de engenharia, dentre elas, os casos de uso, user stories, diagramas BPMN, (todos com intuito de entender e documentar as funcionalidades pertinentes ao organizador de caronas, - de acordo também com os princípios da metodologia de Design Science - dessa forma ampliando o entendimento da plataforma nessa versão e fornecendo ferramentas precisas de organização e de projeto para os seguintes passos de desenvolvimento.) Concomitantemente com a aplicação dessas técnicas e em função delas, foi-se determinando os requisitos funcionais e não funcionais da aplicação.

Determinados os requisitos funcionais e não funcionais, partiu-se para a próxima etapa de desenvolvimento: determinar uma arquitetura que melhor se aplicasse a todos os conceitos desenvolvidos e necessidades reconhecidas. Para isso, dentre as opções de mercado, como arquitetura de microsserviços, arquitetura MVC, arquitetura client-server, decidiu-se para a versão primeira a arquitetura em camadas dada sua simplicidade de abstração, alta modularidade e alta escalabilidade, características essas que atenderam muito bem os requisitos e atenderão muito bem as necessidades futuras da plataforma.

Sendo determinada a arquitetura de camadas e projetando cada um de seus elementos, foi então possível a etapa de pesquisa de tecnologias que servissem à arquitetura. Nessa etapa, a partir das necessidades de cada elemento descrito, procurou-se as tecnologias mais utilizadas pelo mercado e que atendessem às características.

Finalizada a etapa de pesquisa e determinação de tecnologias que permitissem a execução do organizador de caronas que aloca motoristas e passageiros em tempo real, com o uso concomitante de diversos usuários em alta escala, simplicidade de uso e processos de gamificação que incentivem os usuários ao uso e permanência na plataforma, foi então necessário comprovar a viabilidade da solução, e a terceira etapa do Design Thinking, a prototipação.

Para a prototipação, a primeira fase foi entender o que um protótipo desse produto intenta demonstrar, quais os parâmetros que levariam a equipe a acreditar na viabilidade da solução, em seu potencial de ser executada e iniciar o desenvolvimento prático deste produto. Por fim, demonstrar uma plataforma de fluxo de usuário intuitiva e que consegue conectar em tempo real dois usuários buscando e oferecendo carona. Com esse intuito uma série de testes intermediários e que servissem a esse objetivo foram criados e que serviram como metas durante a implementação do protótipo.

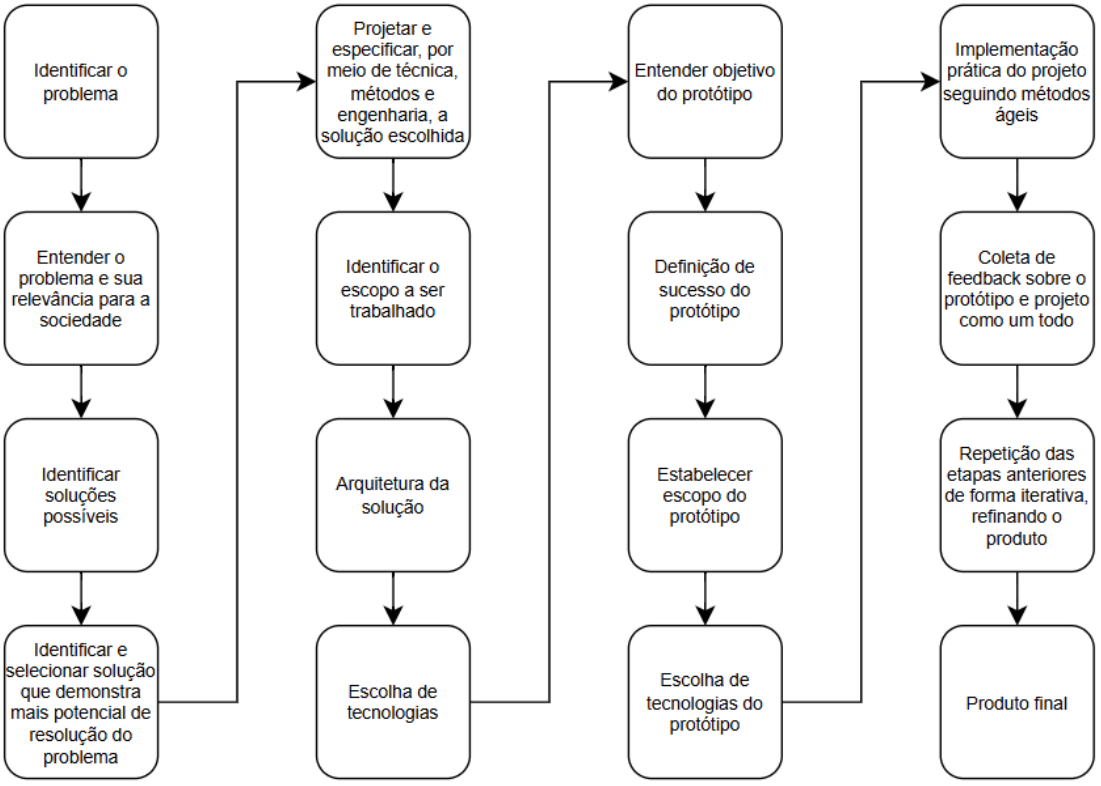
A determinação desses testes que avaliassem o objetivo, foi essencial como parte da definição do escopo de trabalho para a implementação do protótipo, que ocorreu em paralelo com a determinação dos testes.

Direcionado o caminho da implementação do protótipo, a equipe então seguiu algumas das tecnologias determinadas após a etapa de arquitetura e alterou algumas devido a constrições de tempo, recursos e complexidade. As tecnologias

foram escolhidas priorizando ambas proximidade com o projeto original e ter um protótipo finalizado dentro do prazo determinado.

Após analisarmos as tecnologias escolhidas, por fim, a etapa de implementação do protótipo se iniciou com a escrita de código e terminou com a validação dos testes, verificando o que poderia ser considerado ótimo e o que necessitaria de mais tempo para ser concluído, e assim tendo métricas que avaliassem o sucesso ou não do protótipo em relação ao objetivo declarado anteriormente.

Figura 5 - Fluxograma do método de trabalho



Fonte: elaborada pelos autores

Dessa forma, as etapas seguintes ao projeto, caso haja continuidade, seriam a quarta etapa do método de Design Thinking: 1) receber feedbacks do protótipo e da solução como um todo, 2) reavaliar se os objetivos continuam os mesmos, 3) avaliar se o produto continua sendo relevante como solução para o problema, 4) definir quais mudanças podem ocorrer para encaminhá-lo para seus objetivos, e dessa forma iterativamente repetir todos os processos anteriores e refinar cada vez mais o produto até que ele possa agir de forma concreta sobre a sociedade, (nesse

caso, a comunidade USP) melhorando a forma de transporte, tornando-a mais saudável, verde, confortável e rápida.

4. Desenvolvimento do Trabalho

Neste capítulo, será explicitado os diversos métodos utilizados ao longo do desenvolvimento da solução para entender suas características essenciais e desenvolver um produto capaz de entregar a solução para o problema já explicitado. Isso significa, entender e definir os requisitos funcionais e não funcionais da solução tecnológica, determinar suas funcionalidades, estudar, escolher e desenvolver uma arquitetura, escolher as tecnologias capazes de satisfazer as características desenvolvidas, definir o objetivo, parâmetros de sucesso e metas de implementação para um protótipo, escolher suas tecnologias, desenvolver o protótipo e por fim testar e validar o protótipo de forma a verificar se ele alcançou o objetivo proposto.

4.1. Especificação de Requisitos

Como visto nas aulas de Engenharia de Software (PCS3413 e PCS3213) , os requisitos funcionais e não funcionais de um sistema definem as funcionalidades e qualidades que um sistema deve apresentar a fim de atender ao escopo do projeto. O escopo, por sua vez, define a abrangência das funcionalidades do sistema, e o que ele deverá ou não atender. Estes dois podem ser obtidos através das pesquisas de trabalhos próximos já realizados para possíveis comparações, e entendimento claro do domínio em que o sistema está inserido.

Como citado anteriormente no tópico 1.3 deste documento, realizamos um estudo de trabalhos e projetos que traziam algo semelhante, e entendemos melhor as necessidades dos usuários de aplicativos de carona no geral, e os requisitos obtidos pelos autores dos trabalhos para desenvolvimento do sistema.

O entendimento do domínio foi obtido através de alguns fatores: vivências e experiências dos integrantes do grupo, que passaram pelos problemas mencionados e que entendem as características do campus e as necessidades do passageiro; e entrevistas com colegas próximos para entender mais a fundo as demandas dos usuários.

Com o conjunto de informações adquiridas, conseguimos identificar os seguintes itens que auxiliam no desenvolvimento do software:

Stakeholders: Motorista, passageiro e Cidade Universitária.

Problema: Ônibus circulares com filas grandes, cheios e com grande intervalo de tempo entre saídas do ponto de ônibus (frota abaixo do necessário).

Solução: Pensou-se em um organizador de caronas, que combina motoristas e passageiros que têm como destino o mesmo local dentro da Cidade Universitária. Tendo em vista que os passageiros estarão esperando nos pontos de ônibus dos circulares como sempre fizeram, mas que agora aceitam a possibilidade de uma carona como meio alternativo. Com essas possibilidades apresentadas ao usuário membro da comunidade, se espera que, diversificando os métodos de transporte, incentivando por meio de processos de gamificação e recompensas, além do claro benefício social implicado no uso da plataforma, a comunidade inteira se beneficie de um transporte mais confortável, seguro, rápido, saudável e concordando melhor com as crescentes necessidades ambientais.

Escopo: Organização de caronas para membros da comunidade USP, que fizeram cadastro no sistema, podendo atuar como motoristas ou passageiros. O motorista tem sua localização determinada e define o destino final (fixo e sendo pontos conhecidos da USP, como institutos e restaurantes universitários). Da mesma forma, a localização do passageiro é determinada pela sua geolocalização e o passageiro define o ponto de destino (com as mesmas opções e características dos citados para o motorista). O sistema se encarrega de combinar os passageiros com o motorista que possuírem igualdade nos pontos selecionados, e tem a responsabilidade de auxiliar o embarque e garantir a correta execução das etapas. Ao fim da carona, ambos têm a possibilidade de avaliar o outro, com o intuito de estabelecer qualidade dos usuários do sistema e coletar os dados de cada carona, para análise da mobilidade interna ao campus. Para além disso, oferecer uma experiência de gamificação para o usuário que incentive a continuidade na plataforma, como recompensas de avatares, skins, features diferentes e personalizadas e possivelmente recompensas que a própria universidade e outros parceiros comerciais poderiam oferecer como créditos nos restaurantes universitários, ingressos a museus associados e outras recompensas que não seriam tão onerosas à Universidade.

Requisitos:

- Cadastro: Usuário cadastra uma única vez no sistema, que será utilizado para próximas utilizações. Deve conter elementos importantes para aumentar

a segurança dos motoristas e passageiros como: nome; email com domínio “@usp.br”; número usp; senha única; instituição que estuda; foto; *whatsapp;* modelo, cor, foto e placa do carro para os motoristas.

- **Login:** Usuário cadastrado pode se autenticar no sistema e utilizar os serviços. A autenticação deve fornecer um token que valida a sessão do usuário por um tempo determinado.
- **Gerência de usuários:** O sistema deve possuir ferramentas que permitem ao administrador e aos usuários gerenciar (adicionar, remover, alterar e consultar) os perfis.
- **Pontos de embarque por localização:** O sistema deve identificar a localização, por meio de geolocalização do dispositivo móvel, do passageiro e do motorista, de forma a determinar o ponto de embarque da carona.
- **Pontos de desembarque fixos:** O sistema deve possuir pontos de desembarque fixos, como institutos, bandejões e lugares conhecidos e frequentemente acessados na USP.
- **Gerência de pontos de desembarque:** O sistema deve permitir que o administrador gerencie (adicione, remova, altere e consulte) os pontos de desembarque.
- **Pedido de carona:** Passageiro cadastrado deve ser capaz de pedir carona, especificando seu destino.
- **Oferecimento de carona:** O motorista cadastrado insere os diferentes locais de destino que considera plausível entregar os passageiros para a carona.
- ***Match:*** Sistema capaz de armazenar os passageiros que estão solicitando caronas, e os motoristas que estão oferecendo. Assim, ele deve ser capaz de identificar possíveis combinações de motoristas e passageiros que tiverem ponto de embarque e desembarque iguais.
- **IHC com alta usabilidade:** É de extrema importância que o sistema possa indicar aos usuários o estado em que se encontram os pedidos e oferecimentos de carona, a escolha de passageiros para seus motoristas, bem como meios de cancelar qualquer ação e dar sensação de controle e segurança para eles.
- **Avaliação:** Ao fim de cada carona, tanto passageiro como motorista tem a possibilidade de avaliar a corrida. Assim o sistema será capaz de armazenar essas avaliações e direcionar para melhorias futuras.

- **Coleta de dados de fluxo:** O sistema deve ser capaz de coletar e armazenar dados (de forma anônima) relativos às caronas realizadas, a fim de gerar relatórios e gráficos úteis à comunidade USP.
- **Coleta de dados de negócio:** O sistema deve coletar dados relativos à experiência do usuário para que possa estar em constante melhoria. Deve também coletar dados que possam ser úteis a integração com outros serviços que possam entregar uma experiência personalizada ao usuário.
- **Processo de gamificação:** O sistema deve oferecer um processo que estimule o usuário por meio de uma estrutura similar a jogos. Isso é, oferecer métricas e pontuações que incentivem o usuário a continuar na plataforma. Por meio desse processo, seria possível também determinar recompensas das quais o usuário possa se aproveitar e assim incentivar o uso e permanência da plataforma.
- **Supervisão dos usuários:** O sistema deve ser capaz de receber e avaliar denúncias e analytics de cancelamento, a fim de manter no aplicativo apenas usuários de qualidade, que não comprometem o ecossistema da plataforma digital.

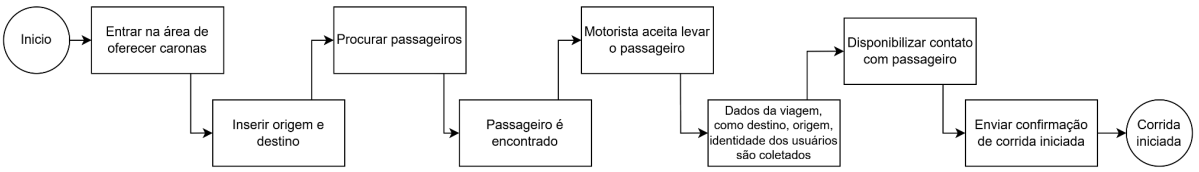
4.2. Transformação dos Requisitos em Funcionalidades

Tendo ambos os requisitos funcionais e não funcionais descritos, houve então o enfoque em transformá-los em funcionalidades dentro do projeto. Para isso foi necessário o conjunto de diversas técnicas e métodos reconhecidos na engenharia de forma a entender as funcionalidades necessárias.

Dentre essas técnicas, como casos de uso, user stories, a de maior destaque ambos por sua facilidade e clareza de documentação e também por ser um meio muito útil no entendimento dessas funcionalidades, foram os diagramas BPMN, que estão elencados abaixo, detalhando o fluxo de informações e de um usuário ao longo de uma tarefa na plataforma.

- **Oferecer Carona**

Figura 6 - BPMN do oferecimento da carona



Fonte: Elaborado pelos autores

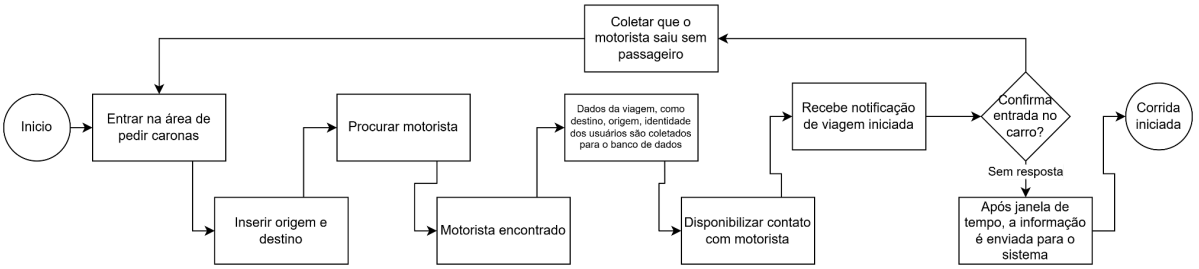
A funcionalidade de oferecer carona acima é assumida apenas pelo papel do motorista que deseja oferecer sua carona a outros usuários com o papel de passageiros. Logo, apenas um usuário com carro registrado na plataforma tem acesso a essa funcionalidade e pode cumprir esse papel. Ao entrar na área de caronas, o motorista indica o local de onde iniciará a carona, e portanto onde permitirá o embarque de passageiros (apelidado como origem); o motorista indica também onde deixará os passageiros dentre as opções de destino. Além dessas opções, o motorista pode indicar quantos lugares ele está disposto a disponibilizar por meio do número de passageiros. Com esses dados disponibilizados, o serviço irá buscar dentre a lista de passageiros buscando uma carona naquele instante que cumpra os requisitos de mesma origem e destino, procurando também encaixar um pedido que tenha o mesmo número de passageiros ou menos e priorize os pedidos feitos em ordem crescente no tempo.

Ao encontrar um passageiro que melhor atenda às condições dentre os da lista, ambos, passageiro e motorista, são alocados um ao outro. O motorista confirma a viagem, os dados dessa viagem (origem, destino, usuários, número de passageiros, tempo de espera, tempo de alocação) são guardados no banco de dados para estudos de fluxo da plataforma, estudos de qualidade e melhoria de serviço, dentre outros.

A plataforma oferece a possibilidade de contato (outro serviço explicado em detalhes abaixo), e o motorista confirma à plataforma que iniciou a corrida, assim iniciando o serviço.

- **Pedir Carona**

Figura 7 - BPMN do pedido da carona



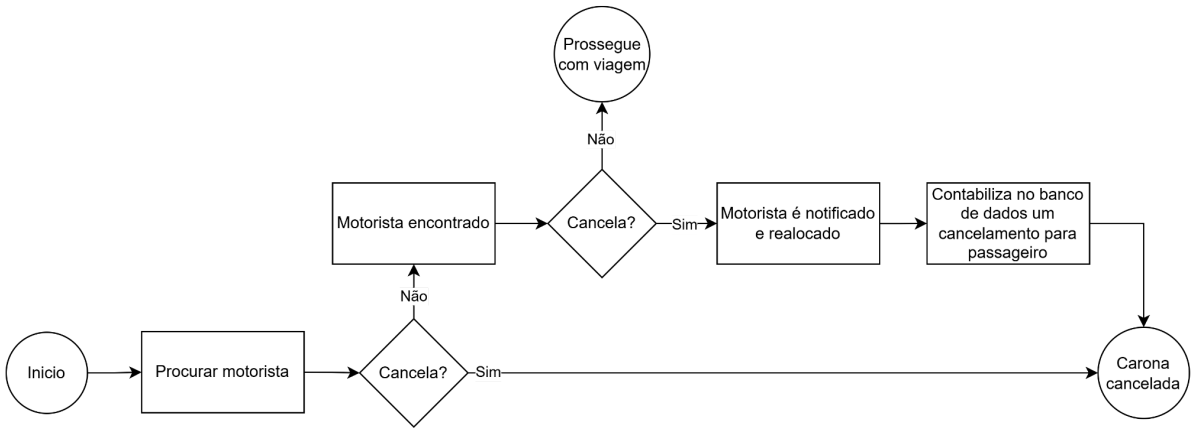
Fonte: Elaborado pelos autores

A funcionalidade de pedir carona é assumida pelo papel do passageiro, que insere em seu pedido, o local de origem, de onde embarca, e o local de destino, onde será deixado. O algoritmo da plataforma procura um motorista que esteja oferecendo carona no momento e que melhor atenda as requisições do passageiro. Ao ser encontrado, ambos motorista e passageiro são alocados um ao outro. Os dados da viagem (mesmos do serviço anterior) são enviados ao banco de dados, o contato do motorista é disponibilizado.

O passageiro recebe uma notificação de que a viagem foi iniciada quando o motorista confirma o início da corrida, podendo então confirmar para o sistema de que está dentro do carro, ou negar a entrada e procurar por outro motorista. Caso o passageiro não confirme sua entrada, após um tempo determinado, a plataforma considera que o passageiro entrou no veículo e percorreu a viagem conforme esperado.

- **Passageiro cancela corrida**

Figura 8 - BPMN do cancelamento da carona pelo passageiro

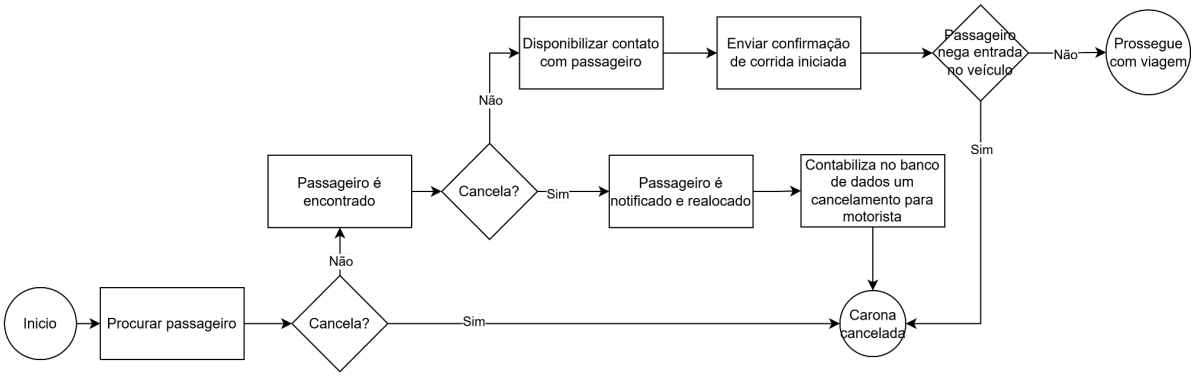


Fonte: Elaborado pelos autores

O passageiro busca uma carona, podendo cancelar durante o estágio de procura de um motorista, ou podendo cancelar mesmo após uma viagem ser atribuída. Caso seja cancelada a viagem, o motorista é notificado e realocado para um novo passageiro. Além disso, quando um motorista já foi alocado, o sistema salva e contabiliza que o passageiro cancelou a viagem e quando, para que o passageiro seja avaliado pelo sistema em caso de muitos cancelamentos em uma curta janela de tempo.

- **Motorista cancela corrida**

Figura 9 - BPMN do cancelamento da carona pelo motorista

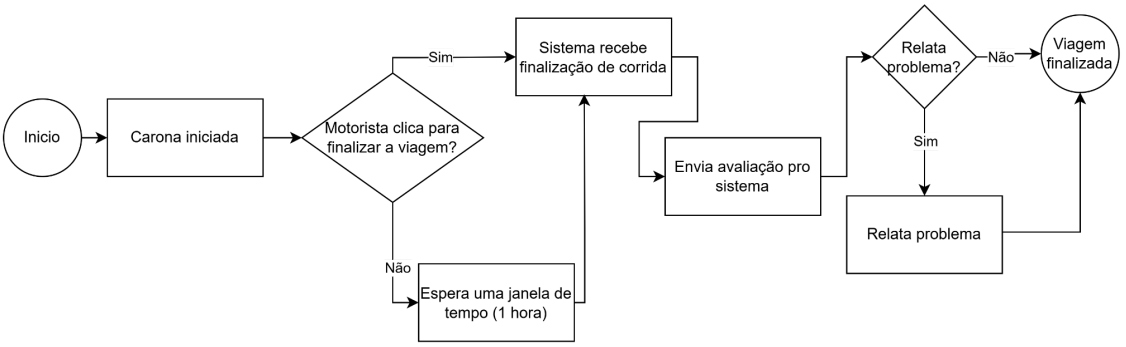


Fonte: Elaborado pelos autores

O motorista busca um passageiro, podendo cancelar durante a fase de procura, ou mesmo após um passageiro ser encontrado. Caso cancele com um passageiro já alocado, esse cancelamento e quando ocorreu são salvos no banco de dados para avaliação do motorista caso haja uma grande quantidade de cancelamentos em curta janela de tempo. Caso não cancele, o contato com o passageiro é disponibilizado e o motorista deve então indicar o início da corrida, caso o passageiro não negue que está no veículo, a viagem prossegue normalmente, caso contrário, a carona é cancelada.

- **Concluir corrida**

Figura 10 - BPMN da conclusão da carona

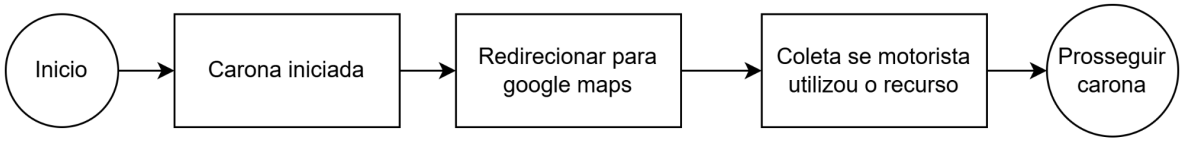


Fonte: Elaborado pelos autores

Com a carona já em andamento, o motorista, ao chegar ao seu destino, pode finalizar a viagem, indicando que chegou ao destino. Caso o motorista não finalize, após uma janela de tempo determinada, a viagem é finalizada por timeout e o fluxo continua da mesma forma para a outra opção. Após essa bifurcação, o sistema coleta uma avaliação sobre a experiência da viagem tanto do passageiro quanto do motorista. Após isso, os usuários podem escolher relatar algum problema que tenha ocorrido durante a viagem, que é salvo na base de dados e continuar o fluxo normal, finalizando de vez a viagem, ou não relatar nada e obtendo o mesmo resultado de finalizar a viagem.

- **Motorista redireciona rota já programada para o Google Maps**

Figura 11 - BPMN do redirecionamento para Google Maps



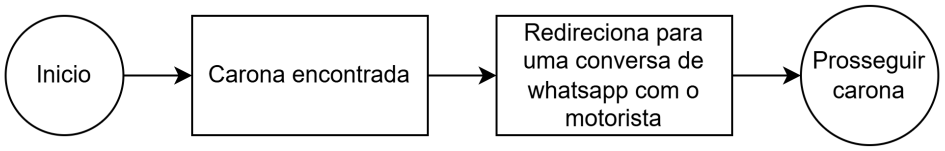
Fonte: Elaborado pelos autores

Com a carona iniciada, o motorista recebe a opção de redirecionar-se da plataforma para o aplicativo google maps com a rota para o seu destino já inserida e pode prosseguir com a carona. Se o motorista opta por utilizar esse recurso ou não, é contabilizado no banco de dados, como um dos parâmetros para inferir a familiaridade do motorista com a rota. Essa é uma das funcionalidades que utilizam o conceito já mencionado de integração entre diferentes tecnologias e plataformas,

facilitando a transição do usuário entre os diversos recursos tecnológicos, de forma intuitiva e muitas vezes imperceptível.

- **Usuário é redirecionado para conversa no Whatsapp**

Figura 12 - BPMN do redirecionamento para Whatsapp

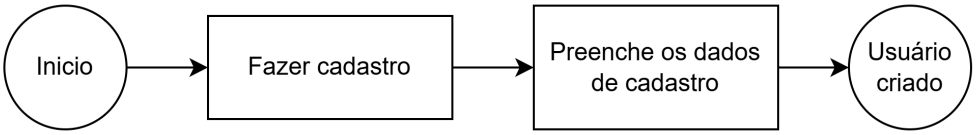


Fonte: Elaborado pelos autores

Antes de iniciar a carona, após encontrar uma alocação de carona, a plataforma oferece um redirecionamento para uma conversa de whatsapp entre os usuários para que, caso precisem, possam se comunicar até o encontro. Mais um exemplo de um recurso ligado ao conceito de Plataforma de Experiências Digitais, tal qual mencionado acima, essa funcionalidade foi pensada se utilizando desses princípios.

- **Fazer cadastro no sistema**

Figura 13 - BPMN do cadastro no sistema

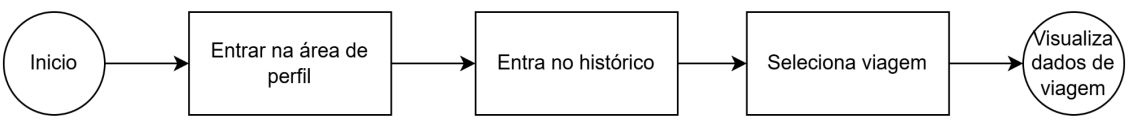


Fonte: Elaborado pelos autores

O usuário em sua primeira entrada na plataforma precisa se cadastrar, preenchendo os dados de identificação, necessários antes de utilizar os recursos oferecidos pela plataforma.

- **Verificar dados de uma viagem no histórico**

Figura 14 - BPMN da verificação do histórico de caronas

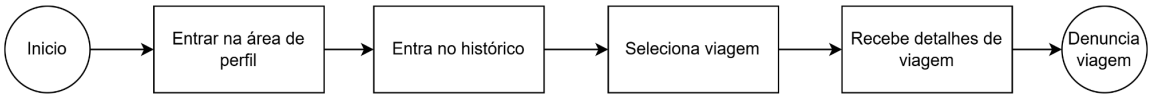


Fonte: Elaborado pelos autores

Um usuário pode verificar ao entrar em um histórico, as viagens que realizou que são identificadas por origem, destino, data e horário que ocorreram. Ele pode também selecionar a tal viagem e receber mais informações a respeito da viagem, como motorista, placa e modelo do carro.

- **Denúncia viagem pelo histórico**

Figura 15 - BPMN da denúncia de viagem

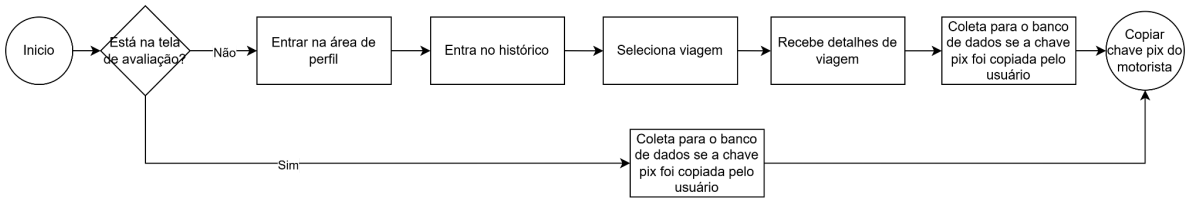


Fonte: Elaborado pelos autores

Ao selecionar uma viagem em específico pelo histórico de viagens, o usuário pode escolher denunciar o passageiro/motorista parceiro.

- **Adquirir chave pix do motorista**

Figura 16 - BPMN da cópia do PIX do motorista

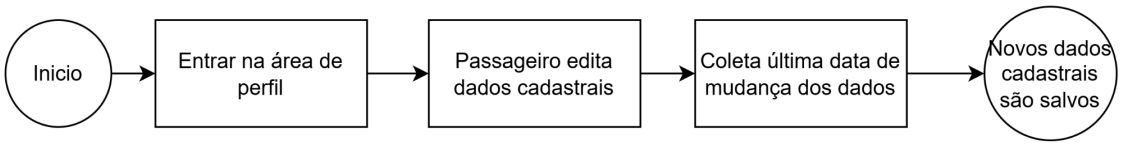


Fonte: Elaborado pelos autores

Ao selecionar uma viagem em específico pelo histórico de viagens, o passageiro pode escolher copiar uma chave pix fornecida pelo motorista, para que, caso queira, possa agradecer ao motorista oferecendo compensação financeira. Caso o usuário copie a chave, esse dado é coletado, servindo como parâmetro indicativo de que o motorista fez um bom trabalho.

- **Editar dados cadastrais**

Figura 17 - BPMN da edição dos dados cadastrais



Fonte: Elaborado pelos autores

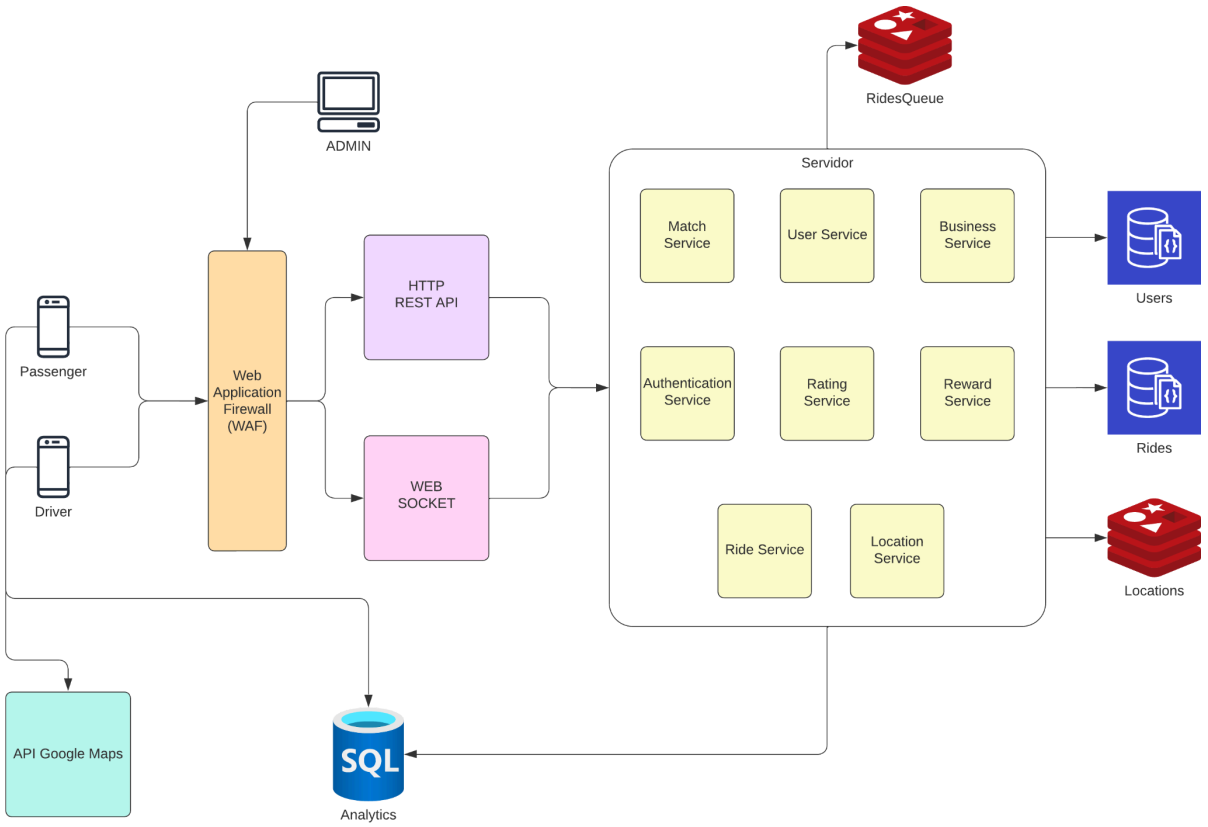
O usuário pode verificar os seus dados pessoais em seu perfil, escolhendo alterar algum dos campos. Caso isso aconteça, a data da última alteração do campo é salva, impedindo o usuário de alterar aquele campo por uma janela de tempo definida.

4.3. Projeto do Sistema

Segundo Sam Newman, no livro "*Monolith to Microservices" (2019),* os desenvolvedores tendem sempre a querer usar a última tecnologia, sem pensar no contexto e os requisitos do sistema antes. Neste livro, o autor demonstra que deve -se implementar microsserviços apenas se for realmente necessário. A prática recomendada é desenvolver um monolito modular, que possui os benefícios de divisão das responsabilidades entre os serviços, mas sem abraçar todas as complexidades de implementação do microsserviço. A partir deste monolito modular, caso seja necessário escalar a aplicação, é válido transicionar para a arquitetura de microsserviços futuramente, desacoplando módulo a módulo para um novo microsserviço. Assim, o desenvolvimento no início do projeto é mais rápido, e ainda é possível escalar sem um esforço tão grande.

Seguindo esse pensamento, foi decidido implementar a arquitetura de um monolito modular para o aplicativo de caronas. Abaixo está o diagrama da arquitetura, bem como uma explicação de cada uma das partes que a compõe:

Figura 18 - Arquitetura do BusCar



Fonte: Elaborado pelos autores

Passenger/Driver: Representam o usuário do aplicativo e constitui o Front-end da aplicação. O Front-End será feito com PWA (Progressive Web App), que é uma maneira de unificar características de aplicações web e aplicativos nativos, com os seguintes benefícios:

- Mesmo sendo em essência, uma aplicação web(site), é possível definir um ícone para ficar nos aplicativos do celular (assim como o *Pinterest* e *Instagram);*
- Traz funcionalidades offline, como recebimento de notificações e mensagens;
- Não precisa saber das especificidades do sistema operacional que o aplicativo rodará, pois o browser se encarrega disso. Isso facilita o desenvolvimento e é possível utilizar o framework web mais conveniente para o grupo;
- É possível obter dados de geolocalização do dispositivo.

ADMIN: Representa o administrador do sistema. Terá uma UI básica para exercer as funcionalidades de gerência, que dentre elas incluem obter uma relação de usuários inscritos, uso da aplicação, denúncias e avaliações de usuários,

pontuações do sistema de recompensas da plataforma, e outros. O administrador poderá também tomar ações, como adicionar pontos de desembarque, avaliar denúncias e excluir usuários da plataforma caso seja necessário.

API Google Maps: Representa a obtenção dos recursos fornecidos com a API do Google Maps. Com essa API, é possível obter informações para renderizar um mapa para o usuário, assim como fornecer dados de rota para o motorista.

Verificou-se previamente a necessidade de dados de geolocalização em diversos dos serviços fornecidos. Desde a coleta de informações sobre a localização do usuário (para a chamada entre passageiro e motoristas próximos e acompanhamento), até a coleta do posicionamento dos usuários para futuras análises de deslocamento dentro do campus.

Para a coleta dessas informações, foram consideradas duas ferramentas muito próximas em funcionamento, a biblioteca Google S2 para javascript e a API Maps Javascript (que é construída em cima da biblioteca mencionada anteriormente).

Por fim, acabou-se escolhendo utilizar a API Maps JS no projeto, isso pois, apesar de não ser gratuita como a biblioteca do Google S2 e da necessidade de estar conectada com a rede para processar os dados geoespaciais (o que não é um problema, já que todo o público alvo tem acesso à rede interna da USP, o eduroam), ela oferece grandes vantagens de projeto que se demonstraram mais vantajosas em relação a Google S2, como oferecer ferramentas de visualização, renderização de mapas e marcadores, facilidade de configuração, e possui uma base de dados global pronta, que inclui, pontos de interesse, informações de vias, imagem satélite, entre outros.

WAF: Representa o Firewall em nuvem que será utilizado para proteger a aplicação de injeção e negação de serviço. Este é um serviço de segurança fornecido pelos principais serviços de cloud do mercado, tal qual Google Cloud, Azure e AWS. Cada um desses serviços tem suas próprias características e benefícios e a necessidade desse tipo de serviço em aplicações web é inegável.

Firewalls servem uma barreira configurável entre a rede interna e o restante da rede, de maneira a prevenir ataques e acessos não autorizados por meio de políticas de segurança. Dessa forma, seria possível inspecionar o tráfego que entra

e sai da aplicação, tornando o tráfego auditável e facilitando a análise de possíveis ataques não previstos pelas já existentes políticas de segurança. Outra funcionalidade é filtrar o tráfego com base em IPs, proteção contra ataques maliciosos e explorações de vulnerabilidades conhecidas, com port scanning e flooding.

As vantagens de se utilizar dos sistemas de firewall em serviços de cloud para além das já mencionadas são sua elasticidade de escalonamento, isso é, como ela escala de forma automática para lidar com o aumento do tráfego na rede, garantindo performance mesmo durante picos de uso. Também por serem distribuídos e gerenciados pelo serviço de nuvem, sua infraestrutura distribuída minimiza latências ou quedas de desempenho.

RidesQueue: Funciona como uma estrutura de armazenamento diferente das demais devido às suas características diferentes dos demais banco de dados da plataforma. Para a fila de caronas que estão em espera de um match foi avaliado que o seu serviço de armazenamento precisaria ser rápido, funcionando como um cache, para que as diversas entradas e saídas de usuários procurando caronas pudessem ser lidas e escritas de forma rápida e não relacional.

Devido à alta compatibilidade com diversas linguagens e com os principais serviços de nuvem e a preferência do mercado de seu uso para problemas que requisitam memória cache e pub/sub, (que são temas pertinentes à solução proposta) o Redis é a melhor opção para a solução. Aqui ficarão armazenados os passageiros que estão à procura de carona, e os motoristas que estão online oferecendo carona, bem como seus pontos de destino:

FIgura 19 - Modelo 1: Passengers

```
{
  "ride_id": "ObjectId", // ID único da corrida, gerado automaticamente pelo MongoDB
  "user_id": "string", // ID do usuário
  "origin": {
    "latitude": "number", // Latitude da origem
    "longitude": "number" // Longitude da origem
  },
  "destiny": "string", // Destino
  "time_ask_ride": "ISODate" // Timestamp do início da corrida
}
```

Fonte: elaborada pelos autores

Figura 20 - Modelo 2: Drivers

```
{
  "user_id": "string", // ID do usuário
  "destiny": "string", // Destino
  "time_ask_ride": "ISODate" // Timestamp do início da corrida
}
```

Fonte: elaborada pelos autores

Users: Banco de dados não-relacional que armazena os usuários cadastrados na plataforma e suas informações.

Figura 21: Modelo dos usuários

```
{
  "user_id": "ObjectId", // ID único do usuário gerado automaticamente pelo MongoDB
  "name": "string", // Nome do usuário
  "telephone_number": "string", // Número de telefone
  "role": "string", // Papel do usuário
  "institute": "string", // Nome do instituto
  "is_driver": "boolean", // Indica se o usuário é motorista
  "img_path": "string", // Path para a imagem de perfil
  "car_img_path": "string", // Path para a imagem do carro
  "driver_info": { // Informações do motorista (opcional, só se is_driver = true)
    "car_brand": "string", // Marca do carro
    "car_model": "string", // Modelo do carro
    "car_color": "string", // Cor do carro
    "car_license_plate": "string" // Placa do carro
  },
  "pixKey": "string", // Chave PIX (e-mail, CPF, telefone, ou chave aleatória)
  "email": "string", // E-mail do usuário
  "password": "string" // Senha do usuário (armazenada de forma segura, com hash)
}
```

Fonte: elaborada pelos autores

Rides: Banco de dados não-relacional que armazena as corridas em andamento em uma tabela, e as corridas finalizadas em outra. Em outro modelo de dados, há também os destinos fixos possíveis para os usuários (Destinations).

Figura 22 - Modelo1: RidesHistory

```
{
  "ride_id": "string", // ID único da corrida, gerado automaticamente pelo MongoDB
  "passenger_id": "string", // ID do passageiro (fornecido externamente)
  "driver_id": "string", // ID do motorista (fornecido externamente)
  "status": "number", // Identifica o status da carona (Em andamento, cancelada, concluída)
  "origin": {
    "latitude": "number", // Latitude da origem
    "longitude": "number" // Longitude da origem
  },
  "destiny": "string", // Instituto de destino
  "time_start_ride": "ISODate", // Timestamp do início da corrida
  "time_end_ride": "ISODate", // Timestamp do término da corrida
  "ratings": {
    "was_fast": "boolean", // Avaliação da velocidade da carona
    "was_confortable": "boolean", // Avaliação do conforto da carona
    "facility_to_offer": "number", // Avaliação da facilidade do oferecimento da carona
    "probability_to_offer_again": "number" // Avaliação da chance de oferecer mais vezes
  },
  "complaint": "string" // Denúncia associada à viagem
}
```

Fonte: elaborada pelos autores

Figura 23 - Modelo 2: Destinations

```
{
  "destination_id": "ObjectId", // ID único de destino, gerado automaticamente pelo MongoDB
  "institute": "string", // Instituto associado ao local
  "name": "string" // Nome do ponto fixo de destino
}
```

Fonte: elaborado pelos autores

Locations: Esse banco de dados tem o intuito de constantemente atualizar a posição de usuários durante chamadas de caronas na plataforma. Ele periodicamente é atualizado com a última localização do usuário, informação essa usada para verificar o par mais próximo de match e facilitar localização do passageiro (avaliando também se o passageiro se afastou demais do ponto de embarque programado) e oferecer um acompanhamento do motorista durante a corrida. Devido à constante atualização do banco, se escolheu como definição de projeto que uma memória cache, mais rápida de atualizar seria mais apropriada e por isso se projetou que também seria utilizado o Redis, tal qual para o RidesQueue.

Figura 24 - Modelo da fila de requisições

```
{
  "_id": "ObjectId", // ID único gerado pelo MongoDB
  "user_id": "string", // Referência ao usuário
  "last_location": {
    "latitude": "number", // Latitude do local (fornecido pelo Google Maps)
    "longitude": "number" // Longitude do local (fornecido pelo Google Maps)
  },
  "last_location_time": "ISODate" // Timestamp da última atualização de localização
}
```

Fonte: elaborada pelos autores

Analytics: O banco de dados de Analytics tem como função registrar os diversos eventos ocorridos ao longo do tempo de uso da plataforma, registrando quando uma carona é pedida, quando um match é encontrado, quando ela é finalizada, local de origem e destino, localizações de motorista e passageiro durante todo o processo de pedido até a finalização da corrida.

Esse banco serve como um grande registro dos eventos gerados na plataforma, gerando informações de negócio que podem ajudar a melhorar o serviço, como os institutos que mais pedem, diferença de tempo entre o pedido e encontro de um match, e a diferença de tempo entre um match e o ínicio de uma viagem, quantas viagens foram canceladas, e muitas outras análises que podem ser recolhidas desse reservatório.

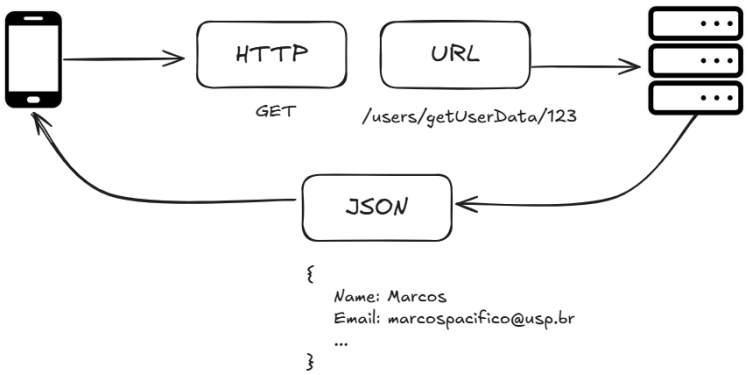
Nesse caso, foi escolhido um banco relacional, SQL Server, isso pois, sendo o grande retentor de informações de negócio e fluxo, é necessário que se possa fazer buscas complexas com condições específicas, o que condiz mais com um banco estruturado e relacional.

HTTP REST API: Dada a necessidade do projeto de se comunicar com o servidor, podendo enviar e receber respostas de forma segura e confiável, verificou-se a necessidade de um método de comunicação, e pra isso, foi escolhido a REST API, uma interface já conhecida por sua confiabilidade e alta performance em escala.

O bloco no diagrama representa uma das formas que os usuários têm para se comunicar com o servidor (usando a REST API). Nessa forma, o usuário envia uma requisição e recebe uma resposta do servidor. É usada, por exemplo, quando o

usuário entra na tela de perfil, onde o front envia uma requisição para o back informando o ID do usuário, e este retorna as informações do usuário, assim como mostra o exemplo abaixo para um usuário chamado “Marcos” com ID “123”:

Figura 25: Exemplo de REST API



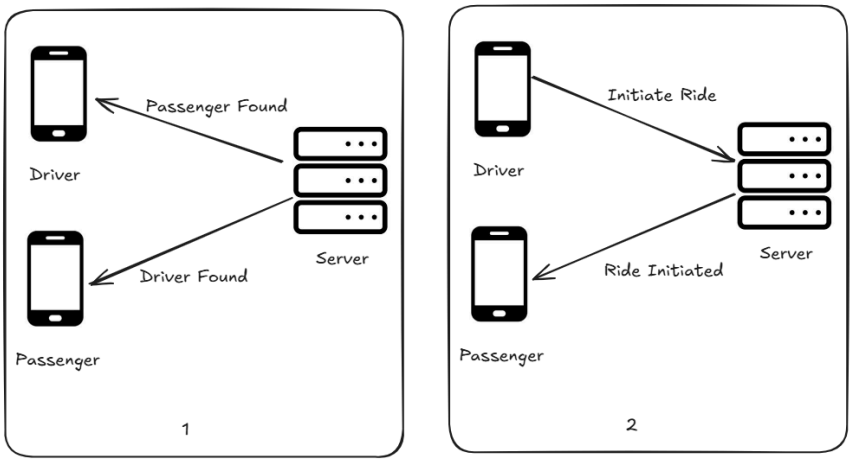
Fonte: Elaborado pelos autores

WEB SOCKET: Dentro do projeto, verificou-se a necessidade de manter uma conexão persistente que permitisse a comunicação entre clientes em tempo real durante uma carona, mantendo ambos os clientes cientes de qualquer nova condição da viagem (como o início, fim e cancelamento de uma carona).

Para isso, foram buscados recursos que mantivessem a conexão, dentre os quais, foi escolhido o websocket, que oferece como vantagens pertinentes ao projeto, sua baixa latência e comunicação bidirecional. Dessa forma, dentre métodos como Server-Sent Events(SSE) e Long Polling, que descumprem um ou mais desses requisitos, a escolha, apesar de mais complexa em questão de implementação, se demonstrou a mais apropriada para os requisitos do projeto.

Essa comunicação é persistente (não é interrompida ao final da requisição como a REST API), bidirecional (tanto o cliente como o servidor podem enviar e receber mensagens) e assíncrona (o servidor pode enviar mensagens com o cliente sem que ele precise requisitar essa informação), permitindo a implementação da funcionalidade de acompanhamento da carona, que notifica em tempo real o usuário o estado da carona. Um exemplo é mostrado na figura a seguir com o servidor notificando os usuários sobre ter encontrado uma carona (1) e com o motorista notificando o início da corrida para o servidor, que notifica o passageiro (2).

Figura 26 - Exemplo de comunicação Web Socket



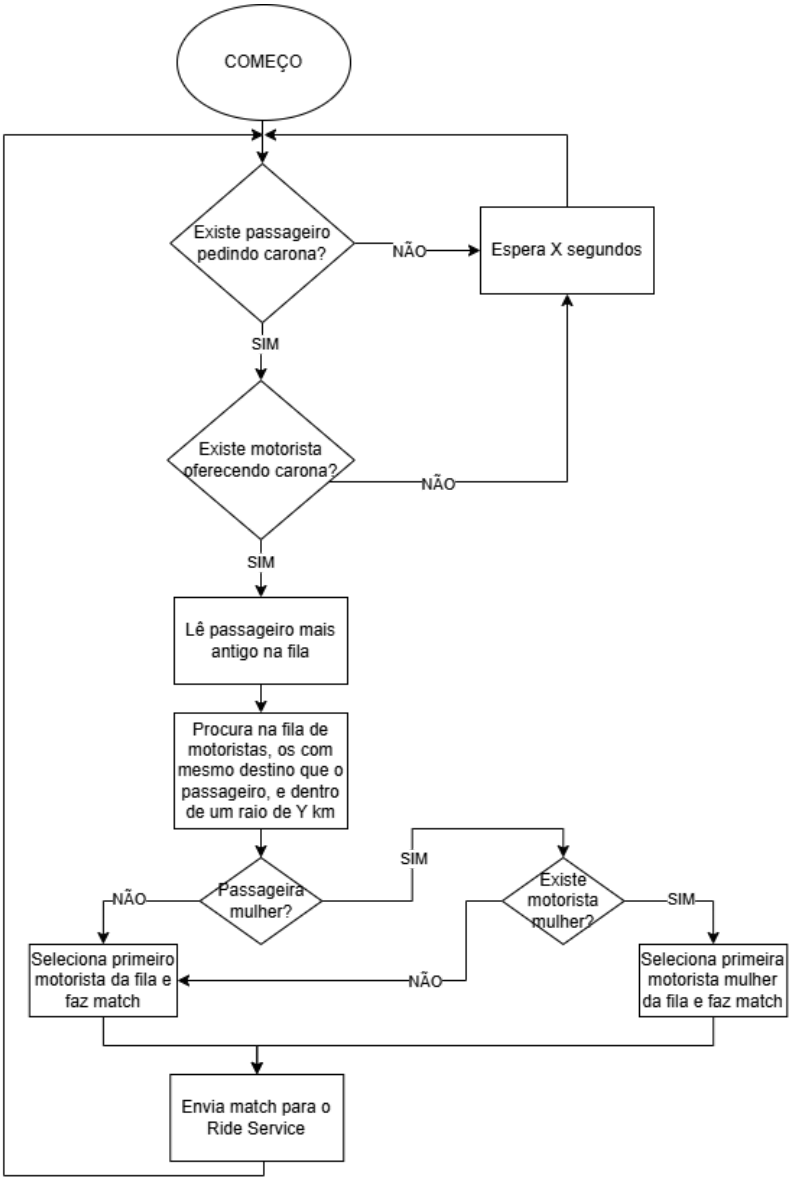
Fonte: elaborada pelos autores

Match Service: Módulo do servidor responsável por arranjar passageiros e motoristas com destino em comum para realizar uma carona (lê as requisições do serviço de cache “RidesQueue”). Esse módulo utilizará os seguintes parâmetros para fazer o *Match:*

- Destino do passageiro e motorista;
- Localização do passageiro e motorista;
- Tempo em que passageiro e motorista estão esperando o match;
- Gênero dos usuários (para priorizar alocação de passageiras mulheres com motoristas mulheres);

O algoritmo que implementa a lógica do match é o que se segue na figura abaixo:

Figura 27: Fluxograma do algoritmo de Match



Fonte: Elaborada pelos autores

Authentication Service: Responsável por verificar se o email e senha do usuário que está tentando logar no aplicativo são válidos (acessando o banco de dados “Users”). Caso sejam válidos, este serviço retorna ao usuário um token contendo o ID do usuário, que vale por um tempo determinado. Esse token é utilizado em todas as requisições do aplicativo para garantir que apenas usuários autenticados possam utilizá-lo.

User Service: Responsável por realizar os serviços relacionados aos usuários, como: Criar, Editar, Deletar e Ler os dados de um usuário da aplicação (utilizando o banco de dados “Users”). Aqui também fica a lógica de destacar os

usuários com um comportamento indesejado, baseado em suas avaliações e número de cancelamentos, a fim de que o administrador possa fazer uma análise mais detalhada e tomar alguma medida para controlar o uso indevido do usuário.

Rating Service: Responsável por receber as avaliações dos usuários ao fim de cada corrida e armazená-las juntamente às informações da carona no banco de dados “Rides”. Com isso, têm-se a capacidade de gerenciar usuários com comportamento indesejado, e também entender pontos de melhoria da aplicação.

Ride Service: O principal serviço da aplicação, responsável por:

- Notificar, por meio da comunicação Websocket, os usuários presentes no match, e manter essa comunicação aberta durante toda carona, até que se inicie o processo de Rating. Durante a carona, ele é o responsável por receber a mensagem de que a corrida foi iniciada pelo motorista, e enviar para o passageiro. Também é o Ride Service que recebe a mensagem de que a corrida foi finalizada e a envia para o passageiro;
- Faz a atualização do status da carona de acordo com os eventos ocorridos (as informações da carona ficam armazenadas no banco de dados “Rides”);
- Armazena a rota da carona no banco de dados “Rides”, a fim de que se possa ter um mapa de calor das rotas mais utilizadas pelos motoristas, gerando informação de valor para a Prefeitura da Cidade Universitária.

Business Service: Responsável pelas operações de consulta feitas pelo administrador. Essas operações visam entender os dados de rota das caronas e os perfis dos usuários associados a estes dados, gerando dados de valor para a Cidade Universitária.

Reward Service: Responsável apenas por controlar a nota do usuário. Essa nota será incrementada sempre que oferecer ou receber uma carona, receber uma avaliação positiva do outro usuário e afins. Cada uma destas ações trará uma pontuação diferente, de acordo com o balanceamento feito pelo grupo. Essas informações de pontuação ficam armazenadas no banco de dados “Users”.

Location Service: Responsável por receber as localizações dos motoristas e passageiros, e armazená-las no banco de dados “Locations”. Os motoristas que se colocam disponíveis a oferecer carona, enviam sua localização a cada X segundos, e este serviço faz o gerenciamento das localizações atualizadas, bem como a deleção das localizações de motoristas que se desconectam. O passageiro envia a área de possibilidade de match baseada na localização no momento em que pediu a carona.

4.4. Tecnologias Utilizadas

Para começar a implementação, foi necessário estudar tecnologias existentes no mercado que correspondiam às necessidades do projeto Levamos em conta quatro principais fatores para escolher as tecnologias: curva de aprendizado, visto que o tempo era curto para finalizar o TCC; adesão da tecnologia, visto que quão mais utilizada é uma tecnologia, mais documentação é possível de se encontrar; integração entre as tecnologias, pois algumas ferramentas são mais fáceis de se comunicar entre si; e por fim, a familiaridade dos integrantes do grupo com as ferramentas, a fim de acelerar ainda mais o desenvolvimento.

Abaixo são descritas as principais tecnologias utilizadas no protótipo:

- **Vue 3**

Vue 3 é um framework com possibilidade de utilizar TypeScript (como foi no nosso caso, para desenvolver um código mais robusto), voltado para o desenvolvimento de interfaces de usuário reativas e modulares, ideal para aplicações de página única (SPAs). A escolha por esse framework se deu por sua curva de aprendizado amigável.

Um outro ponto positivo é a existência de bibliotecas desse framework que facilitam o processo de desenvolvimento, como Vuetify, que facilita a implementação de telas com visual moderno e amigável, assim como ajuda no desenvolvimento de interfaces responsivas, que se adequam aos diferentes tamanhos de telas que podem acessar o BusCar.

Outras duas bibliotecas deste framework utilizadas para o desenvolvimento do Front-end, foram: Axios, para ajudar na implementação de chamadas de API; e Pinia, para passar informações importantes entre telas.

- **ASP.NET Core**

O ASP.NET é uma estrutura de código aberto, multiplataforma e alto desempenho para a criação de aplicativos modernos conectados à Internet e habilitados para nuvem, fazendo com que atenda os requisitos do nosso backend. A capacidade de rodar em diferentes plataformas traz uma maleabilidade para escolher as configurações do servidor.

Um dos grandes motivos da escolha desta tecnologia foi também a familiaridade dos integrantes do grupo com a escrita de código utilizando C#. Com isso, ganha-se tempo para desenvolver o protótipo. Outro ponto forte é a integração nativa com ferramentas modernas e protocolos, como autenticação com JWT, comunicação em tempo real com SignalR, e acesso facilitado a bancos de dados por meio do Entity Framework Core. Além disso, a organização modular promovida pelo ASP.NET Core (como a separação por camadas e uso de Services) encaixa-se perfeitamente na proposta do monolito modular, tornando o código mais limpo e fácil de manter. Por ser um framework amplamente utilizado no mercado, ele também proporciona acesso a uma comunidade ativa, ampla documentação e suporte de longo prazo pela Microsoft, garantindo confiabilidade para o desenvolvimento acadêmico e possibilidades para futuras expansões.

- **SignalR**

O SignalR é uma biblioteca da ASP.NET que facilita a criação de aplicações em tempo real, permitindo comunicação bidirecional entre servidor e cliente. Ela é usada para adicionar recursos interativos, como notificações ao vivo, em aplicações web. O SignalR abstrai as complexidades do WebSocket e, quando necessário, faz fallback para outras tecnologias compatíveis, como “*Long Polling”,* garantindo compatibilidade ampla com os diversos navegadores onde o BusCar rodará.

- **SQL Server**

O SQL Server é um sistema de gerenciamento de banco de dados relacional (SGBDR) desenvolvido pela Microsoft, projetado para gerenciar e armazenar grandes volumes de dados com segurança e eficiência. Ele oferece suporte completo para transações ACID, garantindo consistência e confiabilidade dos dados.

A escolha desta tecnologia para o banco de dados se deu pela forte integração nativa ao ecossistema Microsoft, como a biblioteca “Entity Framework Core”, para facilitar a execução de operações CRUD no SQL Server.

- **MongoDB**

O MongoDB é um banco de dados NoSQL orientado a documentos, projetado para lidar com grandes volumes de dados de forma escalável e flexível. Ele armazena informações em documentos no formato JSON (ou BSON), permitindo esquemas dinâmicos, o que facilita a manipulação e evolução dos dados sem necessidade de migrações complexas. Ideal para aplicações modernas, o MongoDB suporta consultas avançadas, índices e agregações, além de integração com diversas linguagens de programação. Ele também é distribuído por design, oferecendo alta disponibilidade e replicação. Essa tecnologia é frequentemente usada em cenários como big data, aplicativos em tempo real e sistemas distribuídos.

- **Redis**

O Redis é um banco de dados em memória de estrutura de dados chave-valor, amplamente utilizado para cache, filas e armazenamento temporário de dados. Ele oferece suporte a tipos de dados como strings, listas, conjuntos e hashes, permitindo operações rápidas de leitura e gravação. Por ser baseado em memória, o Redis proporciona alta performance e baixa latência. Ele também suporta persistência opcional dos dados em disco, além de replicação e clustering para alta disponibilidade e escalabilidade. Redis é frequentemente utilizado para melhorar o desempenho de sistemas e aplicações que exigem respostas rápidas.

- **PWA**

PWA (Progressive Web App) é uma abordagem para criar aplicativos web que oferecem uma experiência similar a de um aplicativo nativo. Ele utiliza tecnologias web modernas como HTML, CSS, JavaScript e APIs para fornecer funcionalidades como trabalho offline, notificações push e carregamento rápido. PWAs são instaláveis, ou seja, podem ser "adicionados à tela inicial" de um dispositivo sem a necessidade de passar por uma loja de aplicativos. Além disso, eles são responsivos e funcionam em diferentes dispositivos e plataformas, garantindo uma experiência consistente. As PWAs combinam a acessibilidade da web com a performance de aplicativos nativos.

Verificou-se o seu grande uso em aplicativos móveis como Instagram, Facebook, Twitter e outros. Ao utilizar essa tecnologia, é possível transpor o

problema do desenvolvimento de aplicativos que funcionam para apenas um tipo de sistema operacional ou precisam dedicar mais esforço para serem compatíveis com os diversos tipos de sistema mobile. Por esses motivos, ficou clara a vantagem de se implementar a tecnologia por meio de um PWA.

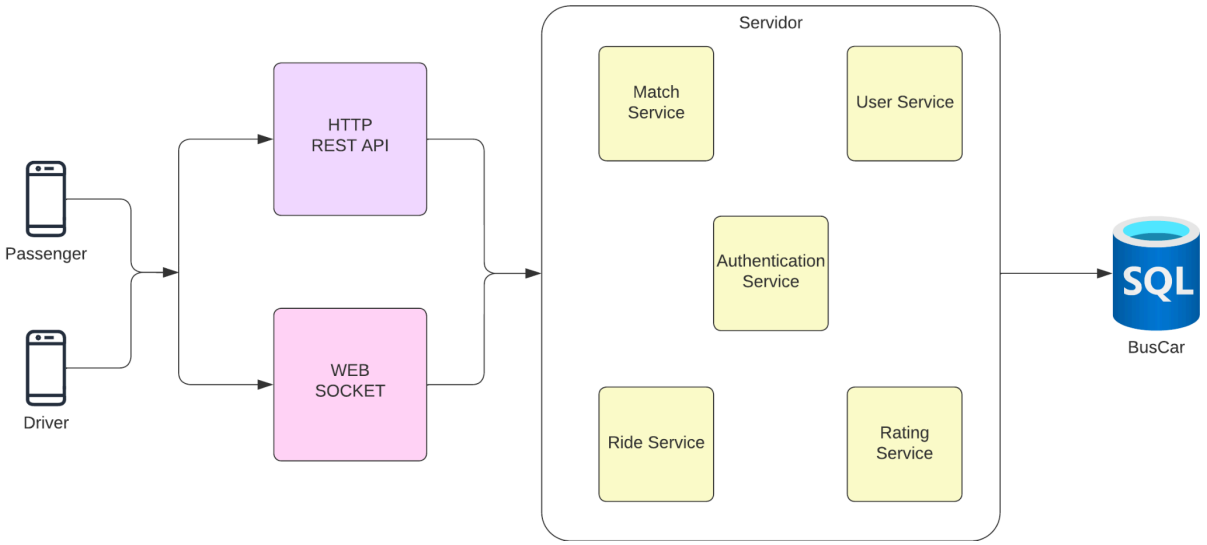
- **GitHub**

O GitHub é um serviço baseado em nuvem que hospeda o Git, um sistema de controle de versões. É extremamente útil para colaboração em equipe no desenvolvimento do código. Foi escolhido esta tecnologia pela familiaridade dos integrantes, assim como por ser de uso gratuito.

4.5. Implementação do Protótipo

A solução proposta e projetada tem diversas funcionalidades, assim como a necessidade de ser implantada com serviços em nuvem. Considerando o tempo hábil para o desenvolvimento do TCC, foi decidido implementar um protótipo que roda localmente, e com uma quantidade de serviços reduzidos, assim como mostra a figura a seguir:

Figura 28 - Arquitetura do Protótipo



Fonte: Elaborado pelos autores

Embora o escopo para implementação tenha sido reduzido, as funcionalidades principais ainda se mantêm, assim como a divisão modular dos serviços para que o desenvolvimento futuro fique mais ágil.

O protótipo contará com a implementação do Front-end (seguindo o design das telas descrito no capítulo seguinte), dos dois modos de comunicação entre Front-end e Back-end (REST API e WEB SOCKET), os cinco serviços listados na figura acima implementados no Back-end, fundamentais para o aplicativo de carona, além de uma primeira estrutura do Banco de Dados.

O protótipo serve apenas como base para começar o desenvolvimento da solução completa e não tem todas as funcionalidades que foram idealizadas. Mas apesar disto, ainda permaneceu a vontade de desenvolver algo funcional e que já pudesse amenizar os problemas da mobilidade na USP. Dito isto, teve-se que passar novamente pela etapa de ideação do Design Thinking, para superar as dificuldades de não utilizar a API do Google Maps e Geolocalização.

Agora que a localização dos usuários não é mais atualizada em tempo real, valeu-se da ideia de implementar pontos de embarque fixos para a carona, sendo estes os pontos de ônibus da Cidade Universitária. Como o público-alvo do aplicativo são membros da comunidade USP que utilizam os ônibus circulares, foi pensado que estes utilizariam o BusCar como uma maneira secundária de se movimentar. Sendo assim, o caso de uso seria um estudante na fila do ônibus circular, assim como sempre fez, mas agora buscando uma carona pelo aplicativo, na tentativa de chegar mais rápido e com mais conforto até o seu destino.

4.5.1. Desenvolvimento das telas

Depois de descrito em mais detalhes as funcionalidades que constituem o nosso protótipo e analisado como o usuário iria interagir com o sistema em seu cotidiano, foi possível desenvolver as telas do aplicativo.

Para desenvolver as telas, começou-se com os “*wireframes”*[8], e após especificadas as interações entre sistema e usuário, foram criados a identidade visual e um logotipo para tornar a implementação do Front-end mais ágil, para não ser necessário se preocupar com design na etapa de escrita do código.

Figura 29 - Logotipo do aplicativo

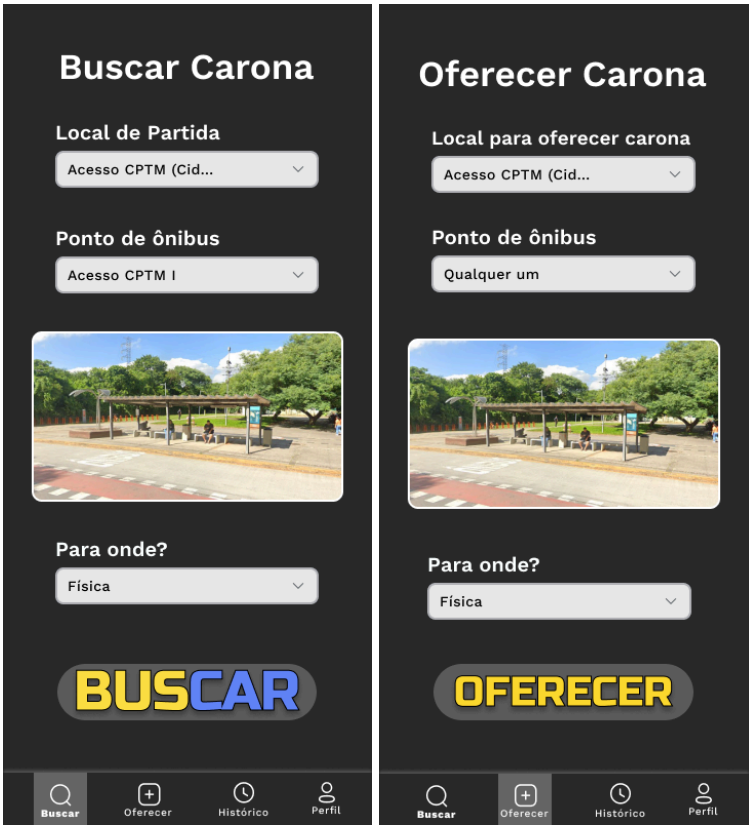
[8] Telas simples que focam apenas na interação do usuário com o sistema, sem se preocupar com cores e identidade visual.

Fonte: elaborada pelos autores

A seguir, serão demonstradas os designs das telas, desenvolvidos no Figma[9], que carregam as principais funcionalidades do sistema:

- **Telas de Busca e Oferecimento de Carona**

Figura 30: Principais telas do app



Fonte: elaborada pelos autores

Essas telas são as responsáveis por receber os dados da carona, origem e destino, tanto para o motorista quanto para o passageiro. As duas são acessadas pela barra de navegação inferior, e contém campos iguais para preenchimento, sendo que as únicas diferenças são os títulos e o texto do botão.

O passageiro acessa a tela de buscar carona, inserindo o local de partida (instituto), depois o ponto de ônibus em que se encontra. Os pontos de ônibus disponíveis são alterados dinamicamente de acordo com o local de partida. Ao

[9] Ferramenta de design online que permite criar, colaborar e testar interfaces, protótipos e wireframes.

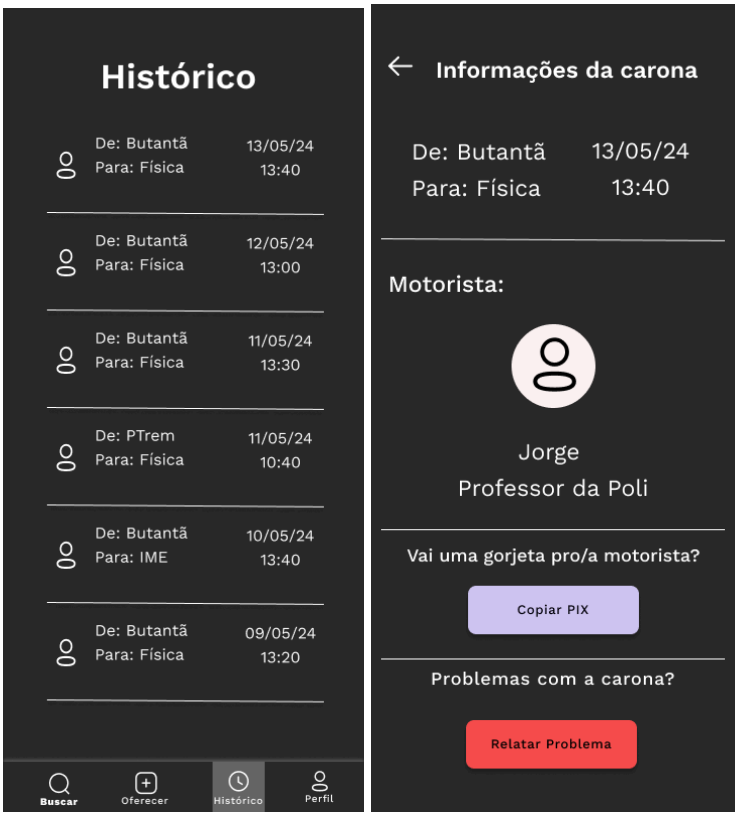
escolher o ponto de ônibus, é carregada uma imagem do mesmo para que o usuário possa confirmar que está no local correto.

Depois de confirmado o local de partida, o passageiro escolhe o local de destino (instituto), e esse é genérico, sem um ponto de ônibus específico, pois o passageiro e motorista poderão decidir o melhor para os dois durante o trajeto. Com isso, basta que o passageiro clique no botão “Buscar”, para iniciar a procura por um motorista com o mesmo percurso.

Do ponto de vista do motorista, a experiência utilizando o aplicativo é basicamente a mesma. Ele escolhe o ponto de ônibus em que deseja pegar algum passageiro (também com a imagem do ponto para o auxiliar no momento de embarque), e então o seu local de destino. Após isso, basta clicar no botão “Oferecer” para iniciar a procura por um passageiro com o mesmo percurso.

- **Tela de Histórico**

Figura 31: Telas para acessar informações de caronas feitas



Fonte: elaborada pelos autores

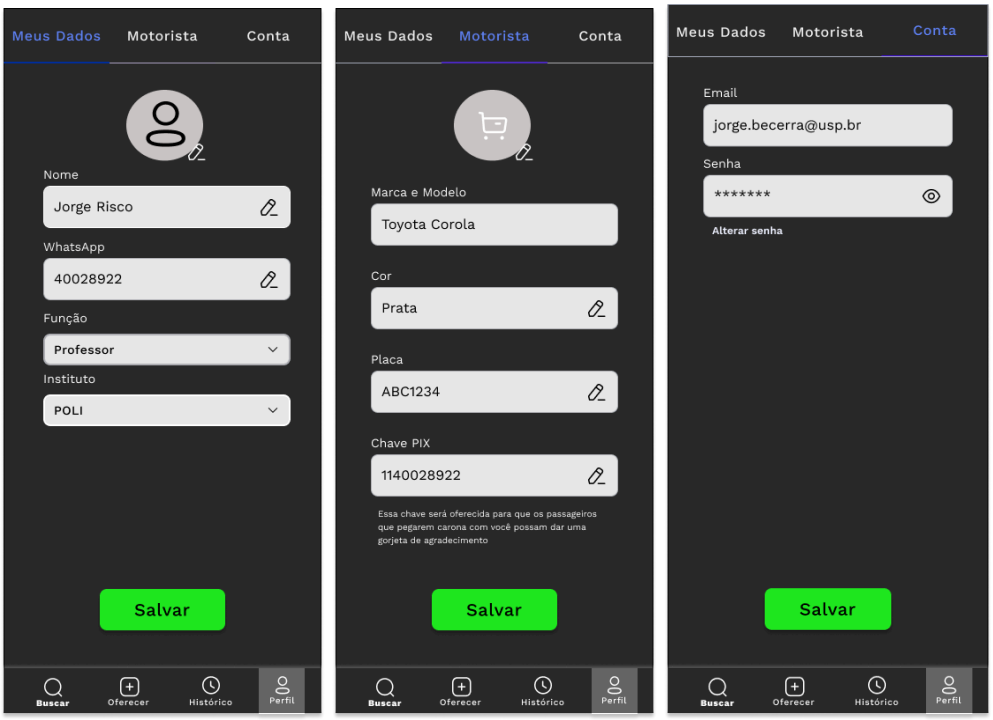
A tela de histórico, também acessada pela barra de navegação inferior, possui as caronas realizadas nos últimos sete dias, organizadas da mais atual para a mais

antiga. Nesta tela, existem informações básicas sobre a carona, mas pode-se clicar nos cards para abrir a tela de “Informações na carona”, para acessar mais detalhes. Caso o usuário tenha sido passageiro nessa carona, nesta tela também é possível copiar o código PIX do motorista que ofereceu a carona para que possa retribuir pela carona oferecida. Caso o usuário seja motorista, este botão não aparece na tela.

Uma outra funcionalidade desta tela é oferecer ao usuário um mecanismo de reportar um problema, seja ele com o aplicativo ou com a pessoa em que esteve nesta carona. Com isso, o administrador do sistema pode consertar um problema no aplicativo, ou tomar medidas cabíveis caso o problema tenha sido com a outra pessoa na carona, já que essa outra pessoa tem seu email USP cadastrado.

- **Tela de Perfil**

Figura 32: Tela para visualizar informações de seu perfil



Fonte: elaborada pelos autores

Esta tela é acessada também pela barra de navegação inferior, e possui uma aba de navegação superior para percorrer entre os três tipos de informações referentes ao perfil do usuário: Dados pessoais relevantes para aplicação, dados do seu carro e chave PIX caso queira oferecer carona, e dados da conta. Perante qualquer atualização desta tela, o botão de “Salvar” entra em estado de habilitado e o usuário pode salvar sua alteração.

- **Telas de Andamento da Carona para Motorista**

Figura 33: Telas de progresso da carona do ponto de vista do motorista

Fonte: elaborada pelos autores

As telas acima estão apresentadas sequencialmente, após o momento em que o motorista clica para oferecer carona. A primeira é apenas um feedback para que o usuário entenda que o sistema está encontrando um passageiro. A segunda é apresentada logo que um passageiro é encontrado, com informações sobre o passageiro para auxiliar no embarque. O botão amarelo abre o aplicativo do Google Maps com a rota até o ponto de ônibus onde está o passageiro, e com destino final até o destino escolhido por ambos. O botão verde leva o motorista para o Whatsapp para que os dois possam se comunicar, facilitando o embarque. Nesta tela, o motorista pode iniciar a corrida após a entrada do passageiro, ou cancelar a carona caso tenha algum imprevisto.

A terceira tela permanece com o botão amarelo para auxiliar na rota do motorista, mas tira o verde, já que os dois já estão juntos neste momento. Também permanecem as informações do passageiro. Nessa tela, é mostrado o botão de finalizar corrida, para que o motorista indique ao sistema que o passageiro já foi entregue.

A última tela é apresentada após o término da corrida, para que o motorista possa avaliar o aplicativo, trazendo assim um feedback para os administradores do

sistema, que possibilita entender se há problemas e possibilidades de melhoria. Ao clicar em “Enviar”, a avaliação é enviada e o usuário volta para a tela inicial.

- **Telas de Andamento da Carona para Passageiro**

Figura 34: Telas de progresso da carona do ponto de vista do passageiro

Fonte: Elaborada pelos autores

As telas estão apresentadas sequencialmente, após o momento em que o passageiro clica para buscar carona. As duas primeiras são semelhantes às duas primeiras do motorista. A única diferença é que o passageiro não tem o botão de iniciar corrida, e ao invés disso, é apresentado a ele as informações do carro do motorista.

A terceira é apresentada depois que o motorista clica em iniciar corrida. Ela permanece com o botão amarelo para que o passageiro possa averiguar a rota, mas tira o verde, já que os dois já estão juntos neste momento. Nesta tela permanece as informações do motorista, mas é retirada a informação do carro. Também é apresentado um feedback para que o passageiro entenda que não precisa fazer mais nada, apenas esperar. Por fim, é apresentado o botão “Não entrei no carro”, para caso o motorista tenha iniciado a corrida sem o passageiro dentro.

A última tela é apresentada após o término da corrida, para que o passageiro possa avaliar o aplicativo, trazendo assim um feedback para os administradores do sistema, para apresentar possibilidades de melhoria. Ao clicar em “Enviar”, a avaliação é enviada e o usuário volta para a tela inicial.

4.5.2. Front-end

Como citado anteriormente, foi escolhido o framework Vue 3, se utilizando de typescript. A partir dele, foi escolhida também a implementação de uma coleção pré-desenvolvida de componentes visuais, chamada Vuetify. Essa escolha se demonstrou de grande importância, pois facilitou de forma significativa a criação de um protótipo funcional e mais próximo dos designs estipulados (aumentando a velocidade de desenvolvimento e legibilidade do código).

4.5.2.1. Funcionamento do código

O Front-end foi desenvolvido em duas etapas principais: 1. Implementação do layout, que fica dentro dos elementos “<template>” dos arquivos de páginas; 2. Adição dos scripts para deixar as telas funcionais, que ficam dentro do elemento “<script>” dos arquivos de páginas. A seguir, tem-se a explicação de como foram feitas essas duas etapas, utilizando uma das telas do aplicativo como exemplo, a fim de que seja possível entender o processo de desenvolvimento (as demais telas do projeto seguem a mesma lógica).

- **Primeira Etapa**

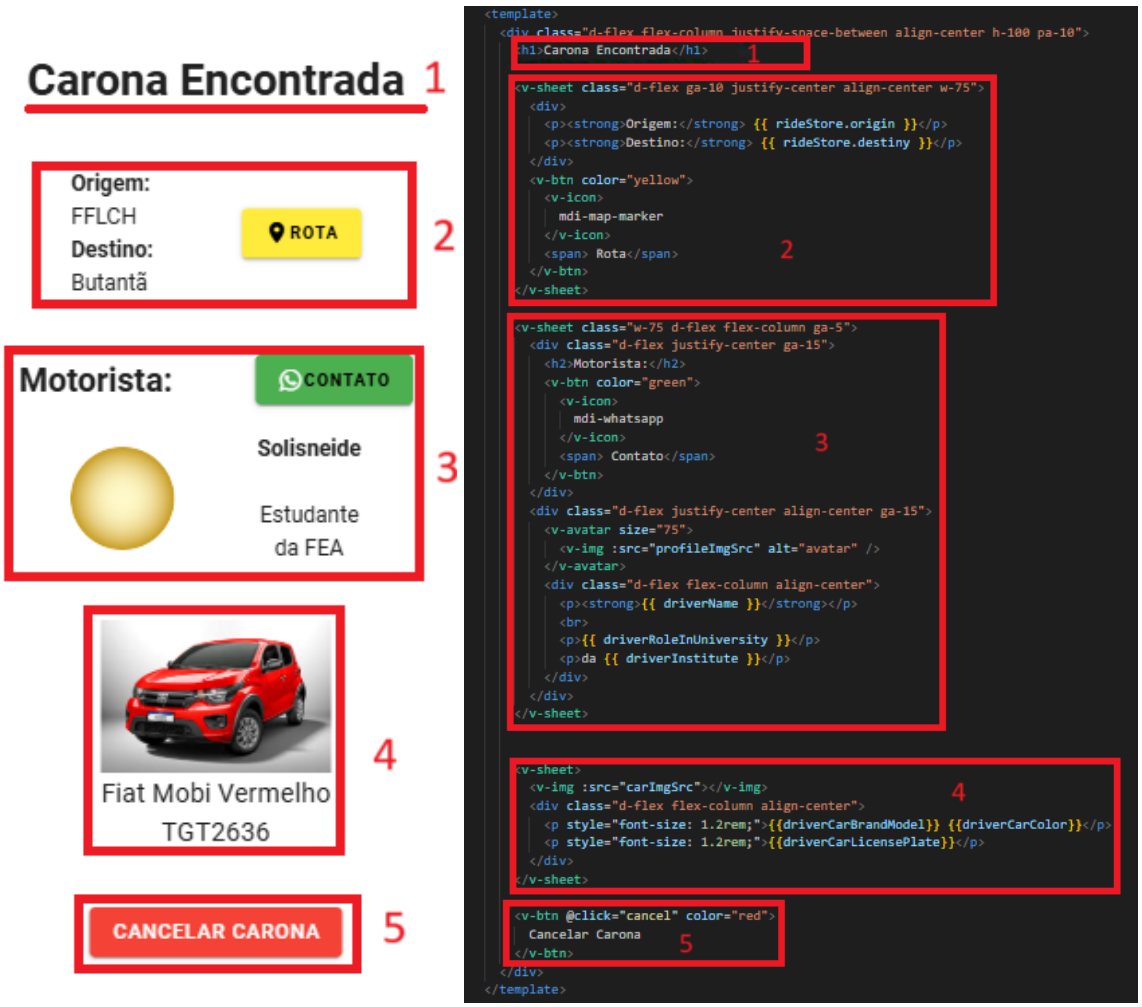
Para a primeira etapa, é necessário elencar os elementos da tela, na ordem em que aparecem, bem como instruções de posicionamento e cor, para que o *browser* possa renderizar corretamente a tela. Abaixo é dado como exemplo a tela que notifica para o passageiro que o motorista foi encontrado, bem como o código que implementa essa layout.

Pode-se dividir essa tela em cinco partes, onde a primeira é o título, a segunda as informações de localização da carona, a terceira informações referentes ao motorista, a quarta tem as informações do carro do motorista, e a quinta o botão de cancelar a carona.

O título (1) é apenas um elemento “h1” (*header 1,* para o browser entender como o título principal da página). Para as informações de localização (2), existem os textos “Origem” e “Destino” seguidos dos textos da origem e destino em si, que são escritos dinamicamente de acordo com o valor de “rideStore.origin” e “rideStore.destiny” (variáveis que carregam as informações de destino e origem

desde a requisição de uma carona). Nesta parte, também existe um botão amarelo, dado pelo elemento “<v-btn>”, com o ícone de localização dado pelo “<v-icon>”, e o texto de “ROTA” com o elemento “”.

Figura 35 - Relação entre layout da tela de Motorista Encontrado e código



Fonte: Elaborada pelos autores

A seção com as informações do motorista (3) foi dividida nas duas linhas que a representam. A primeira linha traz o texto “Motorista” e o botão verde escrito “CONTATO” com o ícone do WhatsApp. A segunda linha traz a imagem do motorista, seu nome, função na USP e seu instituto. A imagem é carregada dinamicamente nesta tela buscando de um servidor de imagens (explicado posteriormente). As demais informações são trazidas de uma API que retorna os dados de um usuário.

A seção com as informações do carro (4) segue a mesma lógica. Existem três linhas, onde a primeira é a imagem do carro do motorista, que também é carregada

dinamicamente buscando em um servidor de imagens. A segunda linha traz a marca e modelo do carro, e a terceira traz a placa, sendo todas essas informações trazidas da mesma API que retorna dados de um usuário.

Por fim, tem-se a sessão com o botão de “Cancelar Carona” (5), que utiliza um “<v-btn>” que ao ser clicado, executa o método “cancel”, presente dentro dos scripts desta página.

- **Segunda Etapa**

Para a segunda etapa, foi necessário estudar a fundo as bibliotecas do “SignalR” (responsável pela comunicação Web Socket), “Pinia” (responsável por armazenar informações necessárias entre as telas) e “Axios” (responsável pela implementação das chamadas de API REST) para entender os detalhes de sua utilização e poder implementar na seção de scripts das telas.

A seguir, é apresentado o código que implementa as funcionalidades da tela de “Motorista Encontrado”. Em primeiro lugar, é necessário importar as funcionalidades provenientes de outros arquivos que vão ser utilizadas (1). Para esta tela, foi importado o “useRideStore”, que armazena variáveis de estado contendo informações como, destino, origem e o ID do motorista que vai levar este passageiro. Também foram importados dois métodos do arquivo que configura as APIs (explicado melhor posteriormente), um que pega o local (URL) do servidor de imagens, e outro que é a API de requisição de dados de um usuário. Por fim, também foi importada a classe responsável pelos serviços do SignalR.

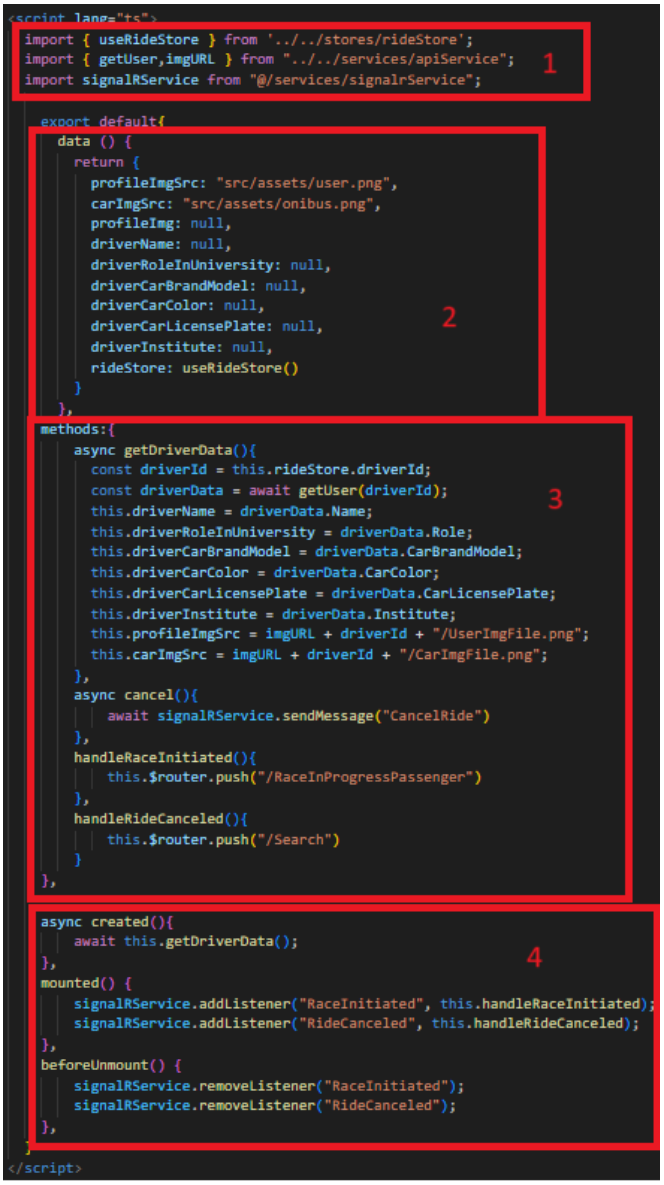
Depois de importadas essas funcionalidades, foram elencadas as variáveis reativas da tela (2), isto é, as variáveis que estão sempre à escuta de uma mudança em seu valor, e quando isso acontece, “avisam” todos os elementos do layout que utilizam elas. Com isso, têm-se uma tela com informações dinâmicas.

Em seguida, neste código são elencados os métodos e suas implementações (3). O primeiro método é o “getDriverData”, que se encarrega de buscar as informações do motorista utilizando a API “getUser”, e as imagens correspondentes utilizando a URL do servidor de imagens “imgURL”, para então atribuir estes valores às variáveis reativas. O segundo método é o “cancel”, que é o método chamado ao clicar no botão de “Cancelar Carona”. Este faz o envio de uma mensagem para o servidor, por meio do SIgnalR, que irá executar o método “CancelRide” no Back-end.O terceiro e quarto método, “handleRaceInitiated” e

“handleRideCanceled”, são chamados quando o cliente recebe a mensagem de corrida iniciada e carona cancelada, respectivamente. O primeiro faz a troca de tela para a tela “RaceInProgressPassenger” (tela de corrida em andamento) e o segundo troca para a tela “Search” (tela de busca de carona).

Na última parte deste código, tem-se a execução de funções nativas do Vue (4). A função “created” é executada logo depois das variáveis reativas terem sido instanciadas, e nela, chama-se o método “getDriverData”, descrito anteriormente. A função “mounted” é chamada após os elementos da página terem sido renderizados, e nela, são adicionadas duas mensagens que o cliente deve começar a escutar, a “RaceInitiated” e a “RideCanceled”, que acionam os métodos descritos anteriormente. A função “beforeUnmount” é chamada antes da tela ser desfeita e trocada para uma próxima, e nela, retiram-se as mensagens que o cliente passa a escutar.

Figura 36 - Script da tela de Motorista Encontrado



Fonte: Elaborada pelos autores

4.5.2.2. Organização dos arquivos e suas funções

Para um entendimento em mais alto nível, agora serão apresentados os diretórios e arquivos importantes para a implementação do Front-end. Na figura a seguir, estão os diretórios:

node_modules: Referentes aos pacotes externos adicionados ao projeto;

public: Arquivos aqui ficam visíveis para toda a aplicação;

assets: Armazena as imagens locais do projeto. No caso do BusCar, armazena imagens *placeholder* para a imagem do perfil e do carro;

components: Estão os arquivos de partes reutilizáveis do código. Como a barra de navegação inferior, e o botão de buscar carona no protótipo;

pages: Estão os arquivos das páginas do aplicativo. Para organização, deixou-se toda página dentro de um diretório próprio. Isso facilitou a leitura e trabalho em equipe;

plugins: Está o arquivo de configuração de plugins para dependências externas;

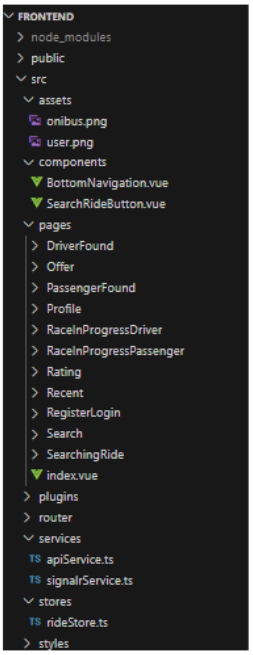
router: Responsável por mapear as páginas e os caminhos utilizados para cada uma delas. O Vue é uma SPA(aplicação de página única), que ajuda em performance, mas possui abstrações para trabalhar como se fossem múltiplas telas.

services: Neste diretório, ficam os arquivos responsáveis por fazer configurações de comunicação com o servidor. No BusCar, têm-se o “apiService.ts”, que configura os endpoints e requisições REST API para o servidor, e o “signalrService.ts”, que configura a conexão web-socket com o servidor.

stores: Aqui ficam os arquivos que utilizam a biblioteca “Pinia”, e estabelecem as variáveis reativas que podem ser acessadas ao longo de todas as telas.

styles: Aqui ficam os estilos que podem ser utilizados por toda a aplicação. Não foi utilizado este diretório, pois se fez uso da biblioteca de componentes estilizados do “Vuetify”.

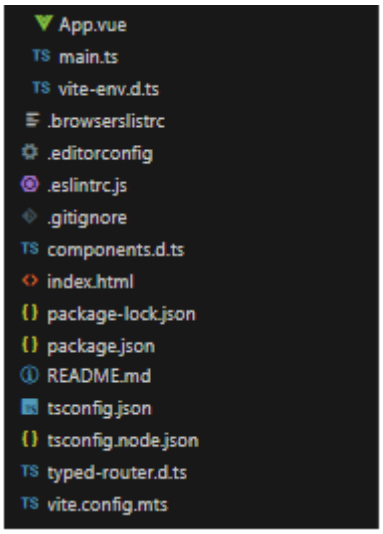
Figura 37 - Diretórios do Front-end



Fonte: elaborada pelos autores

Além dos diretórios, o projeto do Front-end também conta com diversos arquivos base e de configuração, assim como mostra a figura abaixo. Os arquivos “package.json” e “package-lock.json” configuram as dependências do projeto para com bibliotecas externas e suas versões.

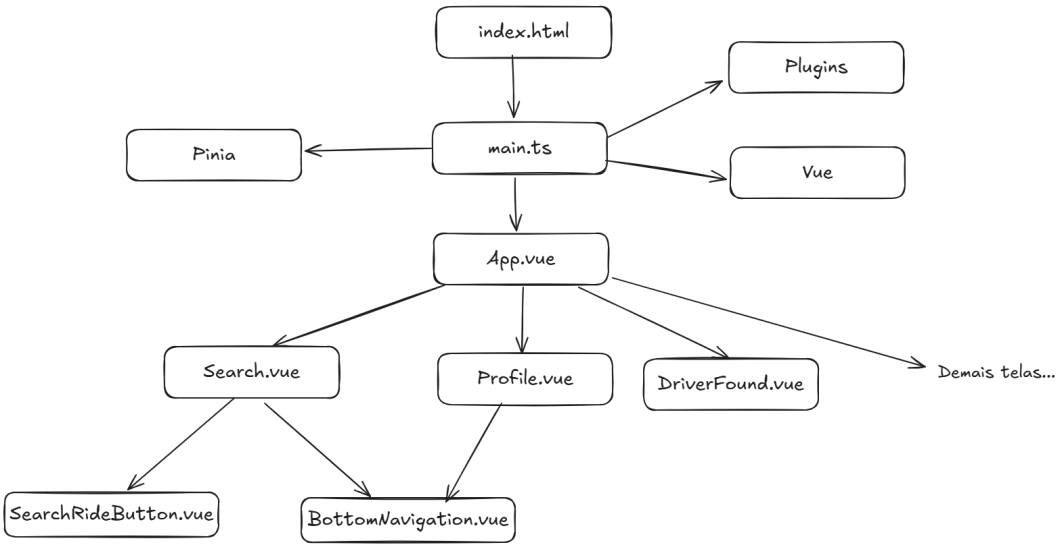
Figura 38 - Arquivos de configuração



Fonte: elaborada pelos autores

O arquivo “index.html” é o primeiro a ser identificado pelo *browser* quando uma página é carregada, e este por sua vez refere-se ao “main.ts”, que possui a lógica principal de configurar e criar a aplicação, e este por sua vez, refere-se ao “App.vue”, que trabalha como o arquivo “pai” das demais telas, que por sua vez podem possuir componentes, assim como ilustra a figura a seguir.

Figura 39: Relação entre os principais arquivos



Fonte: elaborada pelos autores

4.5.3. Back-end

O Back-end foi desenvolvido utilizando ASP.NET Core, com a linguagem de programação C#. Neste tópico, será mostrada a estrutura geral do código, os

modelos que abstraem as entidades do projeto, uma explicação dos métodos implementados para as regras de negócio e acesso ao banco de dados, e por fim, as RESP APIs e métodos do Hub do SignalR.

4.5.3.1. Estrutura do Código

A organização e estrutura do código está presente na imagem abaixo com uma explicação das principais partes que o compõem:

Properties: possui configurações gerais do projeto que são estabelecidas no momento da criação;

Controllers: Aqui ficam os arquivos que configuram os pontos de entrada da aplicação via REST API;

Data: Responsável pela configuração do banco de dados;

Hubs: Aqui ficam as configurações referentes ao Hub, que é uma abstração da comunicação Web Socket implementada pelo SignalR;

Migrations: Responsável pelas configurações da biblioteca Microsoft Entity Framework, que faz a abstração das operações envolvendo banco de dados;

Models: Aqui ficam os modelos do back-end, que são as estruturas de dados principais para realizar as regras de negócio;

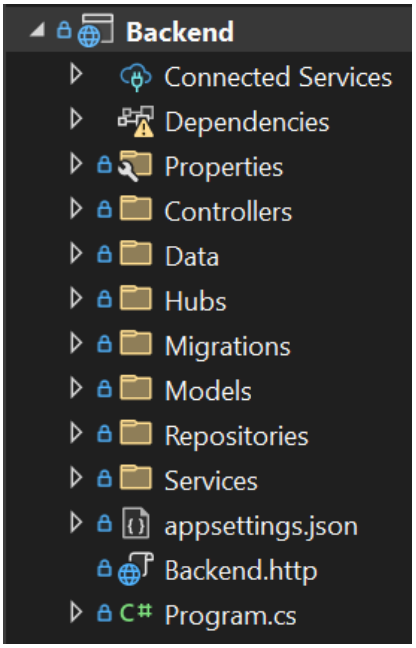
Repositories: Responsável por armazenar os arquivos referentes às operações que se têm com o banco de dados;

Services: Responsável por armazenar os arquivos que realizam as regras de negócio da aplicação;

appsettings.json: Responsável por armazenar configurações como, *string* de conexão com o banco de dados e chave secreta para ser utilizada no token de autenticação;

Program.cs: Ponto inicial da aplicação, é aqui que ficam as configurações gerais do projeto, bem como a injeção de dependências do projeto ASP.NET Core.

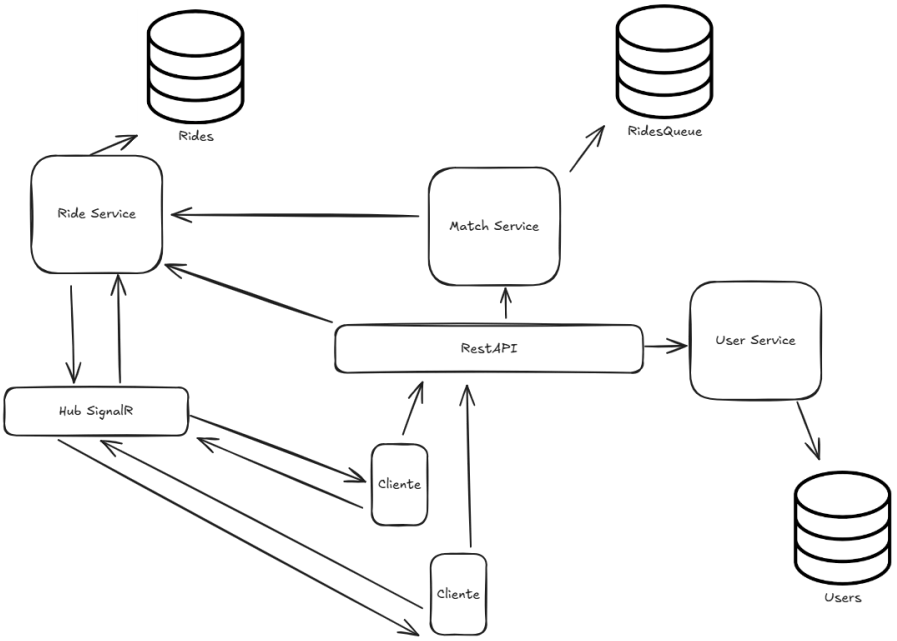
Figura 40 - Estrutura do código no back-end



Fonte: elaborada pelos autores

Para ilustrar melhor e facilitar o entendimento de como as principais partes que compõem o back-end se comunicam, foi feita uma ilustração que abstrai em alto nível essa estrutura, assim como segue:

Figura 41 - Comunicações do back-end

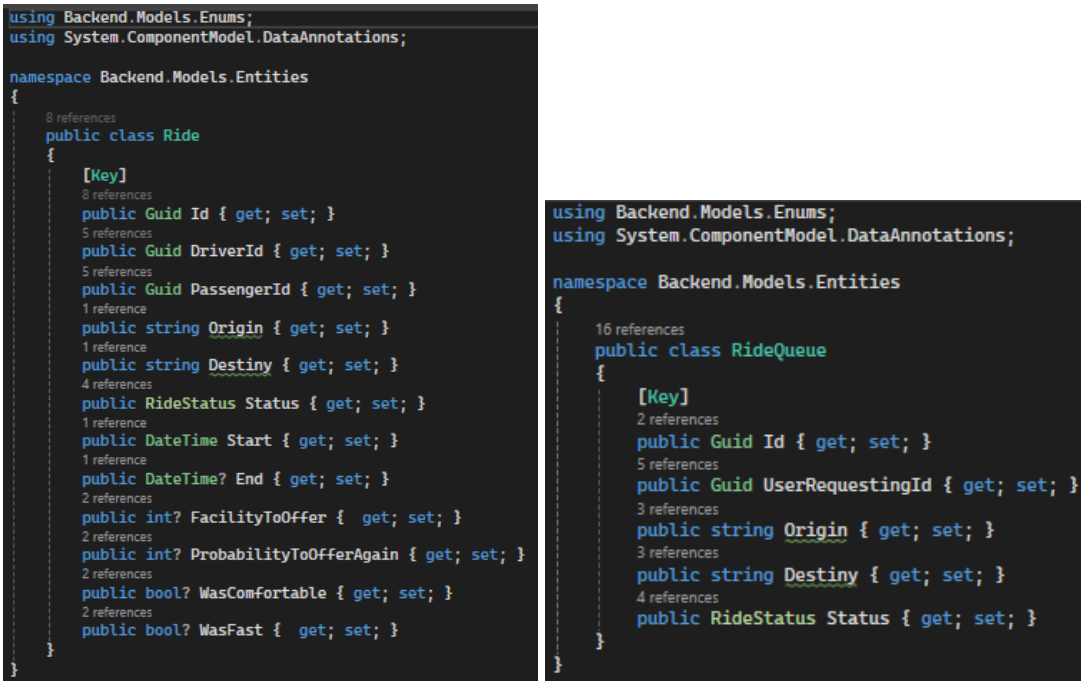


Fonte: elaborada pelos autores

4.5.3.2. Modelos

A seguir, estão listados os modelos, que são responsáveis por abstrair as entidades da nossa aplicação, com seus respectivos atributos, a fim de que possam ser implementadas as regras de negócio os utilizando.

Figura 42 - Modelos da carona e solicitação de carona



```
using Backend.Models.Enums;
using System.ComponentModel.DataAnnotations;

namespace Backend.Models.Entities
{
    8 references
    public class Ride
    {
        [Key]
        8 references
        public Guid Id { get; set; }
        5 references
        public Guid DriverId { get; set; }
        5 references
        public Guid PassengerId { get; set; }
        1 reference
        public string Origin { get; set; }
        1 reference
        public string Destiny { get; set; }
        4 references
        public RideStatus Status { get; set; }
        1 reference
        public DateTime Start { get; set; }
        1 reference
        public DateTime? End { get; set; }
        2 references
        public int? FacilityToOffer {  get; set; }
        2 references
        public int? ProbabilityToOfferAgain { get; set; }
        2 references
        public bool? WasComfortable { get; set; }
        2 references
        public bool? WasFast {  get; set; }
    }
}
```

```
using Backend.Models.Enums;
using System.ComponentModel.DataAnnotations;

namespace Backend.Models.Entities
{
    16 references
    public class RideQueue
    {
        [Key]
        2 references
        public Guid Id { get; set; }
        5 references
        public Guid UserRequestingId { get; set; }
        3 references
        public string Origin { get; set; }
        3 references
        public string Destiny { get; set; }
        4 references
        public RideStatus Status { get; set; }
    }
}
```

Fonte: elaborada pelos autores

A figura acima mostra a classe “Ride”, que é o modelo para abstrair uma carona. Nela, estão presentes informações como o ID, o passageiro, motorista, origem e destino, entre outras conforme a figura. Também é possível observar a classe “RideQueue”, que implementa o modelo de uma carona na fila de caronas. Esta possui menos informações que a “Ride”, mas suas informações são utilizadas para se criar uma carona quando um *match* é feito.

Já a figura abaixo mostra como a classe “User” implementa a abstração do usuário, que carrega consigo as informações que foram inseridas por ele no momento do cadastro.

Figura 43 - Modelo do usuário

```
using Backend.Models.Enums;
using System.ComponentModel.DataAnnotations;
using System.Globalization;

namespace Backend.Models
{

    public class User
    {
        [Key]

        public Guid Id { get; set; }

        public string Email { get; set; }
        public string Password { get; set; }
        4 references
        public string Name { get; set; }
        4 references
        public string WhatsApp { get; set; }
        4 references
        public Institutes Institute { get; set; }
        4 references
        public RoleInUniversity Role { get; set; }
        2 references
        public string? ImgPath { get; set; }
        5 references
        public string? CarBrand {  get; set; }
        5 references
        public string? CarModel { get; set; }
        4 references
        public string? CarColor { get; set; }
        4 references
        public string? CarLicensePlate { get; set; }
        4 references
        public string? PixKey { get; set; }
        2 references
        public string? CarImgPath { get; set; }
    }
}
```

Fonte: elaborada pelos autores

4.5.3.3. Métodos para Regras de Negócio

Nesta seção, serão elencadas as interfaces dos serviços presentes no protótipo. As interfaces ditam os métodos existentes no serviço, bem como suas entradas e saídas. A implementação de cada um dos métodos pode ser consultada no repositório, pois na monografia serão explicadas em alto nível como os métodos funcionam e suas responsabilidades.

- **MatchService**

Figura 44 - Interface do serviço de Match

```
using Backend.Models.Entities;

namespace Backend.Services.Interfaces
{
    4 references
    public interface IMatchService
    {
        2 references
        public Task<RideQueue> RequestMatchAsync(Guid passengerId, string origin, string destiny, string rideRole);
        2 references
        public Task<RideQueue> CancelMatchRequestAsync(Guid userId);
    }
}
```

Fonte: elaborada pelos autores

RequestMatchAsync: tem como objetivo inserir uma requisição de carona (vinda do passageiro ou usuário) na fila de requisições “RidesQueue”. Seus parâmetros, obtidos na API, trazem as informações necessárias para a implementação, como o ID do usuário, a origem, destino e papel do usuário (passageiro ou motorista).

Quando terminado esse processo, o método retorna a requisição para API, que por sua vez retorna um “OK” para o cliente, sinalizando que a requisição foi feita com sucesso.

CancelMatchRequest: tem como objetivo remover a requisição do usuário da fila de requisições. Recebe como parâmetro apenas o ID do usuário, pois um usuário só pode ter uma requisição por vez, e esse é um dado que o cliente tem do lado dele o tempo todo.

- **RideService**

Figura 45 - Interface do serviço de Carona

```
using Backend.Models.Entities;
using Backend.Models.Enums;

namespace Backend.Services.Interfaces
{
    8 references
    public interface IRideService
    {
        2 references
        public Task InitiateRideAsync(RideQueue rideQueue, Guid userId);
        4 references
        public Task<List<string>> UpdateRideStatusAndGetUsers(string userrId, RideStatus rideStatus);
        2 references
        public Task SetDriverRatingAsync(string driverId, string facility, string probability);
        2 references
        public Task SetPassengerRatingAsync(string passengerId, string wasFast, string wasComfortable);
    }
}
```

Fonte: elaborada pelos autores

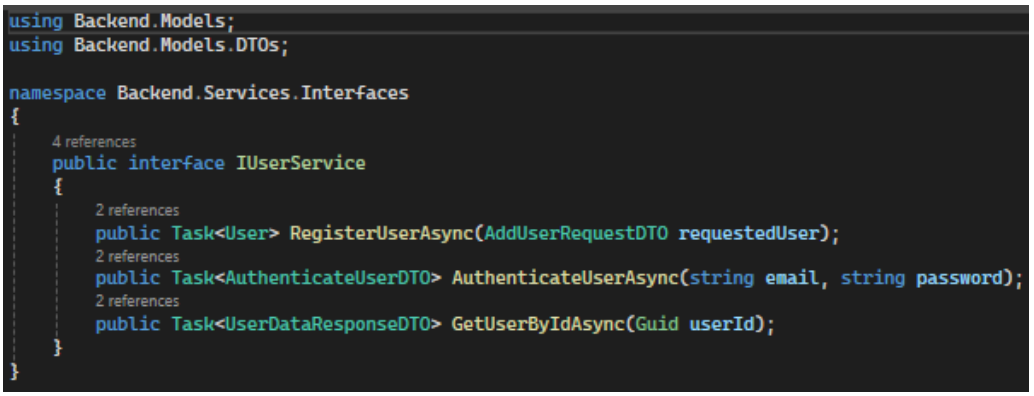
InitiateRideAsync: tem como objetivo adicionar a carona no registro de carona “Rides”. Ele é chamado pelo “MatchService” quando um match é detectado. Para isso ele recebe a requisição que havia sido feita, e o outro usuário que cumpriu com o *match.* Com isso ele consegue criar o modelo “Ride” e notificar os usuários da carona posteriormente.

UpdateRideStatusAndGetUsers: tem como objetivo atualizar o status da carona. Ele é chamado pelo “RideHub” quando a corrida é iniciada, finalizada e cancelada. Este método também retorna os dois usuários presentes na carona, para que o Hub possa comunicá-los corretamente. Para isso, o método recebe como parâmetros o ID do usuário e o Status que a corrida precisa ser atualizada.

SetDriverRatingAsync e SetPassengerRatingAsync: servem, respectivamente, para atualizar os campos de avaliação de carona do motorista e passageiro. Eles têm como parâmetros o ID do usuário fornecendo a avaliação, e sua respectiva avaliação.

- **UserService**

Figura 46 - Interface do serviço de Usuário



```
using Backend.Models;
using Backend.Models.DTOs;

namespace Backend.Services.Interfaces
{
    4 references
    public interface IUserService
    {
        2 references
        public Task<User> RegisterUserAsync(AddUserRequestDTO requestedUser);
        2 references
        public Task<AuthenticateUserDTO> AuthenticateUserAsync(string email, string password);
        2 references
        public Task<UserDataResponseDTO> GetUserByIdAsync(Guid userId);
    }
}
```

Fonte: elaborada pelos autores

RegisterUserAsync: tem como objetivo adicionar um novo usuário para o sistema. Ele recebe como parâmetros todas as informações do cadastro e as salva na tabela “Users”. Além disso, ele também faz o upload das imagens para o servidor de imagens, e então guarda o caminho das imagens para salvar no registro do usuário.

AuthenticateUserAsync: tem como objetivo autenticar um usuário no sistema. Ele recebe como parâmetros o email e senha da API, e valida se estão presentes e corretos no banco de dados. Se estiver correto, ele cria um token de autenticação com o ID do usuário e o retorna para a API.

GetUserByIdAsync: recebe o ID do usuário da API e retorna as informações de cadastro. Este método é utilizado sempre que se precisa dessas informações nas telas.

4.5.3.4. Métodos para Acesso ao Banco de Dados

Nesta seção, serão elencadas as interfaces dos repositórios presentes no protótipo. As interfaces ditam os métodos existentes no repositório, bem como suas entradas e saídas. Os repositórios são responsáveis por fazerem operações simples no banco de dados.

- **MatchRepository**

Figura 47 - Interface do repositório de Match

```
using Backend.Models.Entities;
using Backend.Models.Enums;

namespace Backend.Repositories.Interfaces
{
    6 references
    public interface IMatchRepository
    {
        2 references
        public Task<RideQueue> GetMatch(string origin, string destiny, RoleInRide userRoleInRide);
        2 references
        public Task<RideQueue> AddMatchRequest(RideQueue match);
        3 references
        public Task<RideQueue> DeleteMatchRequestByUserId(Guid userId);
    }
}
```

Fonte: elaborada pelos autores

Possui um método para verificar a existência de um *match*, dado uma origem, destino, e papel da pessoa que está solicitando um *match.* Assim como para adicionar e deletar uma requisição na tabela RidesQueue.

- **RideRepository**

Figura 48 - Interface do repositório de Carona

```
using Backend.Models.Entities;
using Backend.Models.Enums;

namespace Backend.Repositories.Interfaces
{

    public interface IRideRepository
    {
        2 references
        public Task<Ride> AddInitiatedRide(Ride initiatedRide);
        3 references
        public Task<Ride> GetRideById(Guid rideId);
        4 references
        public Task<Guid> GetRideIdInProgressByUser(Guid userId);
        4 references
        public Task UpdateRideStatus(Guid rideId, RideStatus status);
        2 references
        public Task<List<string>> GetUsersByRideId(Guid rideId);
        2 references
        public Task SetEndTime(Guid rideId, DateTime? endTime);
        2 references
        public Task SetDriverRating(Guid rideId, int facilityToOffer, int probabilityToOfferAgain);
        2 references
        public Task SetPassengerRating(Guid rideId, bool wasFast, bool wasComfortable);
    }
}
```

Fonte: elaborada pelos autores

Possui métodos que auxiliam o serviço de caronas, como: Adicionar uma nova carona no banco de dados; retornar uma carona utilizando seu ID ou o ID de um usuário; atualizar o campo de *status* de uma carona; retornar os usuários presentes em uma carona pelo seu ID; atribuir uma data de fim para uma carona cancelada ou finalizada; assim como salvar as avaliações feitas pelos usuários ao fim da carona.

- **UserRepository**

Figura 49 - Interface do repositório de Usuário

```
using Backend.Models;

namespace Backend.Repositories.Interfaces
{
    6 references
    public interface IUserRepository
    {
        2 references
        public Task<User> AddUser(User user);
        1 reference
        public void DeleteUser(Guid userId);
        2 references
        public Task<User> GetUserById(Guid userId);
        2 references
        public Task<User> GetUserByEmail(string email);
        1 reference
        public Task<User> UpdateUser(User user);
    }
}
```

Fonte: elaborada pelos autores

Possui métodos que permitem que o serviço de usuários realize suas operações, como: adicionar um usuário ao sistema; deletar um usuário (para ser feito pelo ADMIN); retornar um usuário pelo seu ID ou por seu Email; e atualizar as informações de um usuário.

4.5.3.5. REST APIs

Nesta seção, serão elencadas as classes dentro do diretório “Controller“, que implementam as APIs, disponíveis no servidor para serem utilizadas pelos clientes.

- **MatchController**

Figura 50 - Configurações dos *endpoints* de Match

```
[HttpPost]
[Route("MatchRequest/{userId}/{origin}/{destiny}/{userRoleInRide}")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
```

```
[HttpPost]
[Route("CancelMatchRequest/{userId}")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
```

Fonte: elaborada pelos autores

Nesta classe, temos dois *endpoints,* um que faz a requisição de uma carona e o outro que cancela a requisição de uma carona. Ambos são métodos POST do HTTP, e precisam que o usuário esteja autenticado com um token para chamá-los.

- **RideController**

Figura 51 - Configurações dos *endpoints* de Carona

```
[HttpPost]
[Route("DriverRating/{facility}/{probability}/{driverId}")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
```

```
[HttpPost]
[Route("PassengerRating/{wasFast}/{wasComfortable}/{passengerId}")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
```

Fonte: elaborada pelos autores

Na classe “RideController”, existem dois *endpoints* disponíveis. Os dois são referentes a realização da avaliação de uma carona, então tomam como parâmetros as avaliações e o ID dos usuários. Ambos também precisam que o usuário esteja autenticado com um token válido para que possa avaliar a carona.

- **UserController**

Figura 52 - Configuração dos *endpoints* de Usuário

```
[HttpPost]
[Route("AddUser")]
```

```
[HttpGet]
[Route("SignIn/{email}/{password}")]
```

```
[HttpGet]
[Route("GetUserData/{userId}")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
```

Fonte: elaborada pelos autores

Na classe “UserController”, existem três *endpoints* disponíveis. Seguindo a ordem da figura, o primeiro serve para adicionar um usuário no sistema, e seus dados vêm dentro do corpo da requisição. O segundo tem como objetivo fazer a autenticação do usuário no sistema, e para isso recebe email e senha. Estes dois não precisam que o usuário esteja autenticado.

O último precisa que o usuário esteja autenticado, e serve para retornar os dados de um usuário qualquer dado um ID de usuário. Este método é chamado pelas telas de Perfil, Passageiro Encontrado, Motorista Encontrado, e nas demais telas onde são necessárias informações de um usuário.

4.5.3.6. Métodos do Hub

Foram criados Hubs para fazer a comunicação entre cliente e servidor em tempo real utilizando o SignalR. Os Hubs são uma espécie de interface bidirecional, onde tanto servidor quanto cliente podem acessar métodos para realizar uma comunicação. Abaixo, estão listados os métodos presentes no RideHub, e as mensagens que são enviadas por ele.

InitiateRide: Chamado pelo “RideService” depois que a carona é criada na tabela “Rides”. Ao ser chamado, ele envia a mensagem “MatchFound” para os usuários da carona com a seguinte linha de código:

Figura 53 - Código para notificar usuários da carona encontrada

```
await hubContext.Clients.User(initiatedRide.DriverId.ToString()).SendAsync("MatchFound", initiatedRide.PassengerId.ToString());
await hubContext.Clients.User(initiatedRide.PassengerId.ToString()).SendAsync("MatchFound", initiatedRide.DriverId.ToString());
```

Fonte: elaborada pelos autores

InitiateRace: Chamado pelo motorista depois que ele clica para iniciar a corrida. Ao ser chamado, ele faz a requisição para atualizar o status da carona e notifica os clientes desta carona com a mensagem “RaceInitiated”.

Figura 54 - Método para iniciar corrida e notificar usuários

```
public async Task InitiateRace()
{
    var users = await rideService.UpdateRideStatusAndGetUsers(Context.UserIdentifier, RideStatus.Running);

    await Clients.Users(users).SendAsync("RaceInitiated");
}
```

Fonte: elaborada pelos autores

FinishRace: Chamado pelo motorista depois que ele clica para finalizar a corrida. Ao ser chamado, ele faz a requisição para atualizar o status da carona e notifica os clientes desta carona com a mensagem “RaceFinished”.

Figura 55 - Método para finalizar corrida e notificar usuários

```
public async Task FinishRace()
{
    var users = await rideService.UpdateRideStatusAndGetUsers(Context.UserIdentifier, RideStatus.Rating);

    await Clients.Users(users).SendAsync("RaceFinished");
}
```

Fonte: elaborada pelos autores

CancelRide: Pode ser chamado pelo passageiro e motorista quando estes clicam para cancelar uma carona, ou quando o passageiro notifica que não entrou no carro e o motorista iniciou a corrida sem ele. Ao ser chamado, ele faz a requisição para atualizar o status da carona e notifica os clientes desta carona com a mensagem “RideCanceled”.

Figura 56 - método para cancelar carona e notificar usuários

```
public async Task CancelRide()
{
    var users = await rideService.UpdateRideStatusAndGetUsers(Context.UserIdentifier, RideStatus.Canceled);

    await Clients.Users(users).SendAsync("RideCanceled");
}
```

Fonte: elaborada pelos autores

4.5.4. Banco de Dados

O banco de dados foi implementado com SQL Server, e as tabelas são relacionadas com os modelos descritos anteriormente. Abaixo são elencadas as tabelas com capturas de tela no “SQL Server Management Studio 20”.

Figura 57 - Estrutura da tabela Rides

Column Name	Data Type	Allow Nulls
Id	uniqueidentifier	☐
DriverId	uniqueidentifier	☐
PassengerId	uniqueidentifier	☐
Origin	nvarchar(MAX)	☐
Destiny	nvarchar(MAX)	☐
Status	int	☐
Start	datetime2(7)	☐
[End]	datetime2(7)	☑
FacilityToOffer	int	☑
ProbabilityToOfferAgain	int	☑
WasComfortable	bit	☑
WasFast	bit	☑

Fonte: elaborada pelos autores

Figura 58 - Estrutura da tabela RidesQueue

Column Name	Data Type	Allow Nulls
Id	uniqueidentifier	☐
UserRequestingId	uniqueidentifier	☐
Origin	nvarchar(MAX)	☐
Destiny	nvarchar(MAX)	☐
Status	int	☐

Fonte: elaborada pelos autores

Figura 59 - Estrutura da tabela Users

	Column Name	Data Type	Allow Nulls
🔑	Id	uniqueidentifier	☐
	Email	nvarchar(MAX)	☐
	Password	nvarchar(MAX)	☐
	Name	nvarchar(MAX)	☐
	WhatsApp	nvarchar(MAX)	☐
	Institute	int	☐
	Role	int	☐
	ImgPath	nvarchar(MAX)	☑
	CarBrand	nvarchar(MAX)	☑
	CarModel	nvarchar(MAX)	☑
	CarColor	nvarchar(MAX)	☑
	CarLicensePlate	nvarchar(MAX)	☑
	PixKey	nvarchar(MAX)	☑
	CarImgPath	nvarchar(MAX)	☑

Fonte: elaborada pelos autores

4.6. Validação e Testes

Neste capítulo, serão apresentados os testes realizados, a fim de validar os resultados obtidos com a implementação do protótipo. O plano de testes foi elaborado de acordo com o que se era esperado do protótipo no início de sua implementação.

Para elencar os testes, foram divididos em testes do front-end, do back-end e de integração. Para cada um destes, é apresentada uma tabela com o teste executado, o resultado esperado e o resultado obtido. Resultados obtidos de acordo com o esperado estão marcados em verde, enquanto os que não atenderam a expectativa estão em vermelho(falha) e amarelo(diferente da expectativa).

Testes do Front-End

Os testes de front-end foram realizados tanto pelo próprio computador rodando a aplicação, quanto por celulares conectados ao computador pelo endereço IP e porta da aplicação.

Tabela 1 - Testes do Front-end

Teste Executado	Resultado Esperado	Resultado Obtido
Abrir o aplicativo em diferentes dispositivos	Tela se adequar ao tamanho do dispositivo e mostrar todos os elementos	Todas as telas desenvolvidas se adequaram corretamente ao tamanho dos dispositivos

Interagir com todos os botões	Botões devem sempre executar uma ação e ter um feedback de interação para o usuário	Todos os botões do aplicativo funcionam de acordo com o esperado
Inserir o endereço IP e porta do computador (servidor do front) em um celular	Receber as telas corretamente no celular	Telas obtidas com sucesso
Abrir o aplicativo com celular no tema claro e no escuro	Cores do aplicativo devem estar de acordo com o tema do dispositivo	Cores alteram de acordo com tema do dispositivo

Fonte: elaborada pelos autores

Testes do Back-end

Os testes de back-end foram realizados utilizando o “Swagger”, que é uma ferramenta criada automaticamente com o projeto ASP.NET Core. Com essa ferramenta, é possível chamar todas as APIs do back-end e verificar seu retorno.

Quando o teste envolveu alterações no banco de dados, foi utilizado o “SQL Server Management Studio 20”, que é uma interface gráfica para verificar os dados da tabela utilizando comandos SQL.

Tabela 2 - Testes do Back-end

Teste Executado	Resultado Esperado	Resultado Obtido
Chamar API User/AddUser	Deve adicionar um usuário no banco de dados	Usuário adicionado com sucesso
Chamar API User/SignIn	Deve consultar os dados de email e senha inseridos, e se forem válidos, deve retornar com um Token de autenticação contendo o ID do usuário	Token só é enviado caso usuário tenha entrado com dados corretos
Chamar API User/GetUserData	Deve retornar os dados do usuário (nome, telefone,...) quando recebe um ID de usuário	Dados do usuário são retornados com sucesso
Chamar API Ride/DriverRating	Deve adicionar a avaliação do Motorista na carona em que ele acabou de participar	É inserida a avaliação do motorista no banco de dados
Chamar API Ride/PassengerRating	Deve adicionar a avaliação do Passageiro na carona em que acabou de participar	É inserida a avaliação do passageiro no banco de dados

Chamar API Macth/MatchRequest	Deve criar uma requisição de carona na fila de caronas caso não tenha match. Caso tenha, deve excluir a requisição da pessoa com quem deu match da fila	Quando um match não é identificado, é adicionado um registro da requisição na fila. Caso um match seja encontrado, a requisição que estava presente na fila do usuário que estava esperando é deletada
Chamar API Macth/CancelMatchRequest	Deve excluir a requisição da carona da fila de caronas	Requisição é apagada da fila com sucesso

Fonte: elaborada pelos autores

Testes de Integração

Os testes de integração dizem respeito ao funcionamento total do aplicativo. Com eles, estamos verificando a integração do front-end e back-end, assim como todo o fluxo do usuário.

Tabela 3 - Testes de integração

Teste Executado	Resultado Esperado	Resultado Obtido
Registrar um usuário pelas telas de criação de conta	Usuário criado com as informações inseridas na tela, e com a imagem salva separadamente em um servidor de imagens	Registro do usuário é criado no banco de dados. Imagem é salva em um servidor, e o caminho para ela é salvo junto ao registro do usuário
Registrar um usuário sem email USP	Usuário é impedido de registrar	Usuário é avisado de que email é inválido e não consegue registrar
Logar no aplicativo	Usuário utiliza seu email e senha para entrar no aplicativo e é redirecionado para a página inicial se usuário válido	Banco de usuários registrados é consultado e caso seja compatível, recebe um token temporário e é redirecionado para a tela inicial
Inserir dados de login inválidos	Usuário é negado o acesso e permanece na página de login	Usuário não recebe token de acesso e permanece na página de login
Acessar a tela de perfil	Usuário logado recebe suas informações cadastradas na tela	Usuário capaz de visualizar todas as suas informações
Acessar a tela de histórico	Receber as informações das últimas caronas realizadas	Lógica da tela de perfil não foi implementada a tempo, então aparecem

		informações estáticas apenas para preencher a tela
Buscar uma carona inserindo origem e destino (sem que haja alguém oferecendo)	Passageiro fica esperando até que clique para cancelar carona	Fluxo esperado ocorreu com sucesso
Oferecer uma carona inserindo origem e destino (sem que haja alguém buscando)	Motorista fica esperando até que clique para cancelar carona	Fluxo esperado ocorreu com sucesso
Buscar ou oferecer uma carona sem inserir origem e destino	Sistema bloqueia requisição e avisa da necessidade de inserir origem e destino	Feedback é dado com sucesso e nenhuma requisição é criada na fila de requisições
Buscar carona com dois dispositivos e oferecer apenas com um	Apenas um dos passageiros é alocado para a carona, e sua requisição é removida da fila de requisições	Um único passageiro é alocado para a carona com o motorista e o outro continua na espera. Banco de dados é atualizado corretamente
Ter dois matches ocorrendo simultaneamente	Usuários de cada match receberem as informações corretas de seus respectivos parceiros de carona	Usuários foram notificados corretamente com as informações respectivas de seu match
Cancelar uma requisição de match	Usuário requisitando volta para a tela anterior e requisição é apagada do banco de dados	Tanto passageiro quanto motorista seguem o fluxo corretamente
Passageiro ou motorista cancelar uma carona	Ambos devem ser redirecionados para a tela anterior a que estavam e status da carona deve ser atualizado para cancelado no banco de dados	Ambos os usuários foram redirecionados corretamente e o status foi alterado corretamente no banco de dados
Passageiro informar que não entrou no carro	Motorista e passageiro devem ser levados para as telas de oferecimento e busca, respectivamente. O status da carona deve ser atualizado para cancelado no banco de dados	Ambos os usuários foram redirecionados corretamente e o status foi alterado corretamente no banco de dados
Motorista iniciar e finalizar uma corrida	Passageiro deve ser notificado destas ações e ambos devem ser levados para a etapa de avaliação da carona	Fluxo da carona seguiu corretamente para ambos usuários
Passageiro e motorista avaliarem a carona	Nota do passageiro e motorista sendo salvas no banco de dados para a carona em questão	Notas salvas com êxito
Reabrir o aplicativo durante uma carona em andamento	Usuário volta para a tela do estado atual da carona e é possível seguir o andamento da carona normalmente	Usuário volta para a tela do estado atual da carona, porém não segue o andamento da carona normalmente

Recarregar a página durante uma carona em andamento	Usuário volta para a tela do estado atual da carona com as informações do outro usuário da carona disponibilizadas corretamente na tela	Usuário volta para a tela atual, porém as informações na tela estão vazias
Clicar no botão de contato do WhatsApp	Usuário é redirecionado para o WhatsApp com uma conversa aberta com o outro usuário da carona	Usuário é redirecionado corretamente e a conversa é aberta no whatsapp
Clicar no botão de rotas do Maps	Usuário é redirecionado ao aplicativo do Maps com a rota estabelecida, utilizando origem e destino	Usuário é redirecionado, mas a rota utiliza endereços genéricos como “Butantã” ao invés de “Metrô Butantã”, fazendo com que a rota não seja precisa em todos os casos

Fonte: elaborada pelos autores

Os testes que não passaram foram analisados para se entender a causa. Para corrigir o da tela de histórico, é necessário criar uma API para retornar as caronas já realizadas pelo usuário, e chamá-la quando o usuário acessar a tela de histórico. O teste de reabrir o aplicativo será corrigido quando for implementado um tratamento para reinicializar a conexão websocket após voltar para o aplicativo, para que possa ouvir as mensagens do servidor. O teste de recarregar a página será corrigido quando for feito um tratamento para requisitar os dados da carona via ID do usuário local, e não mais usando os dados armazenados pelo Pinia, já que são resetados ao recarregar a página.

O teste com resultado diferente da expectativa será corrigido quando for implementado o sistema de coordenadas para os pontos, pois este é preciso. Com a implementação atual que utiliza o nome do local é necessário fornecer um nome mais específico para o Maps.

5. Considerações Finais

5.1. Conclusões do Projeto de Formatura

Ao final do desenvolvimento deste trabalho, pode-se dizer que o projeto teve um resultado satisfatório, porém diferente do que era esperado no começo do desenvolvimento. No início, existia a vontade de ter um aplicativo já lançado e disponível para todos os membros da comunidade USP, porém foi percebido ao longo do projeto que a implementação desse aplicativo era mais complexa do que o

esperado, fazendo com que o grupo, juntamente com o orientador, reduzisse o escopo para a implementação de um protótipo.

Por meio da vivência dos integrantes do grupo na comunidade, relatos de colegas e análise de artigos e notícias, um problema social foi identificado: existe uma deficiência no sistema de transporte da Cidade Universitária. A partir disso, os integrantes do grupo se colocaram em uma imersão do problema a fim de entendê-lo a fundo, e entender as dores das pessoas afetadas. Com essa imersão em paralelo aos conhecimentos tecnológicos adquiridos no curso, foi possível identificar uma solução, integrar os meios de transporte na cidade universitária por meio de uma única aplicação de acesso fácil e intuitivo para o usuário.

Com esses conhecimentos adquiridos, foi possível arquitetar um sistema composto de múltiplas partes, que otimiza a utilização de um recurso já existente na comunidade (os automóveis privados), para que todos os requisitos obtidos por meio da imersão fossem atingidos. Utilizando o leque de tecnologias conhecidas pelo grupo, conseguiu-se selecionar cada uma delas de maneira criteriosa, elencando os pontos positivos e negativos para o contexto, para contemplar a arquitetura proposta. E assim, com a prototipação, foi validada a viabilidade do sistema com as escolhas feitas.

O protótipo possui a parte principal da solução tecnológica, a comunicação em tempo real com diversos dispositivos, juntamente com uma interface amigável e com alta usabilidade. Valendo-se dos testes elencados, foi confirmado que o projeto tem um grande potencial para continuidade.

5.2. Contribuições

A primeira contribuição, e a mais evidente, é o desenvolvimento dos primeiros passos para o sistema de alocação de caronas. Este trabalho pode ser utilizado futuramente para refinar a solução e incrementar a implementação a fim de que se possa chegar no objetivo inicial, um aplicativo disponível para todos os membros da USP.

Uma outra contribuição foi apresentar um método para transformar um problema, de caráter social, em uma solução, de caráter tecnológico, impulsionando a inovação na sociedade e melhoria da utilização de tantos recursos disponíveis.

Por fim, este trabalho contribuiu fortemente para o aprofundamento dos conhecimentos dos integrantes do grupo. Todos os integrantes foram colocados à

frente de diversos desafios que elucidaram pontos de melhoria na formação acadêmica e profissional.

5.3. Perspectivas de Continuidade

Como mencionado, o projeto possui um grande potencial, e portanto deve ser continuado para se chegar no sistema completo arquitetado. É de vontade dos integrantes do grupo e do orientador continuar com a implementação do sistema nos anos seguintes. A continuidade da implementação será dada por testes com um grupo de usuários, seguindo a última etapa do *Design Thinking*, a fim de trazer refinamentos para o produto e se mantendo fiel à metodologia de Design Science, isto é, o processo contínuo de pesquisa e melhoria da solução. As melhorias que devem ser feitas a primeiro momento para o protótipo são:

- Tratamento dos pontos de falha identificados no protótipo;
- Criação do arquivo de configuração para tornar o Front-end um PWA;
- Compra de um domínio para o site;
- Implantação do sistema na nuvem;
- Desenvolvimento das outras funcionalidades idealizadas.

Referências Bibliográficas

ADUSP. **Prefeitura do Câmpus da Capital recua de mudanças nos circulares e abre "consulta pública"; proposta inicial, que seria implantada no reinício das aulas, foi questionada pela comunidade.** Disponível em: <https://adusp.org.br/mobilidade-urbana/circulares-consulta/>. Acesso em: 05 mai. 2024

ALMEIDA, Wilson Mesquita de. **Revisitando "USP para Todos?" : desafios permanentes na inclusão dos estudantes de baixa renda no ensino superior público brasileiro**. Revista de Ciências Sociais, [S. l.], v. 51, n. 3, p. 21–62, 2020. DOI: 10.36517/rcs.51.3.d02. Disponível em: <http://www.periodicos.ufc.br/revcienso/article/view/54817>. Acesso em: 6 mai. 2024.

ARCHILLI, Stefanie. **Design Thinking: método prático acelera processo de inovação.** Disponível em: <https://blog.mbauspesalq.com/2017/03/24/design-thinking-metodo-pratico-acelera-processo-de-inovacao/?gad_source=1>. Acesso em: 1 mai. 2024.

AMAZON WEB SERVICES**. AWS Network Firewall.** Disponível em: <https://aws.amazon.com/pt/network-firewall/>. Acesso em: 15 set. 2024.

_____. **O que é uma API RESTful?** Disponível em: <https://aws.amazon.com/pt/what-is/restful-api/>. Acesso em: 5 set. 2024.

_____. **What is Redis?.** Disponível em: <https://aws.amazon.com/pt/elasticache/what-is-redis/>. Acesso em: 13 set. 2024.

FLETCHER, Patrick et al**. Introdução ao SignalR**. Disponível em: <https://learn.microsoft.com/pt-br/aspnet/signalr/overview/getting-started/introduction-to-signalr>. Acesso em: 5 out. 2024.

GIT. **Documentation.** Disponível em: <https://git-scm.com/doc>. Acesso em: 10 ago. 2024.
GITHUB**. GitHub Docs.** Disponível em: <https://docs.github.com/pt>. Acesso em: 22 set. 2024.

GOOGLE. **API Maps Javascript Documentation**. Disponível em: <https://developers.google.com/maps/documentation/javascript?hl=pt-br>. Acesso em: 10 out. 2024.

HEVNER, A. R. et al. **Design science in information systems research.** Minneapolis: MIS Quarterly, 2004. 30 p.

HOFFMAN, Michael et al**. Visão geral dos Aplicativos Web Progressivos (PWAs).** Disponível em: <https://learn.microsoft.com/pt-br/microsoft-edge/progressive-web-apps-chromium/>. Acesso em: 20 set. 2024.
INSTITUTO DE POLÍTICAS DE TRANSPORTE & DESENVOLVIMENTO**. Transporte Público.** Disponível em: <https://itdpbrasil.org/programas/transporte-publico/>. Acesso em: 18 abr. 2024.

LACERDA, Daniel Pacheco; DRESCH, Aline; PROENÇA, Adriano; ANTUNES JÚNIOR, José Antonio Valle. **Design Science Research: método de pesquisa para a engenharia de produção.** São Carlos: Scielo, 2013. Disponível em: <https://www.scielo.br/j/gp/a/3CZmL4JJxLmxCv6b3pnQ8pq/#>. Acesso em: 4 abr. 2024.

LACERDA, Guilherme Mangueira. **A influência do uso de aplicativo de carona na mobilidade urbana: um estudo de caso do Blablacar.** Paraíba, 2023. Disponível em: <IFPB - Repositório Digital: A influência do uso de aplicativo de carona na mobilidade urbana: um estudo de caso do Blablacar>

MICROSOFT. **ASP.NET Documentation.** Disponível em: <https://learn.microsoft.com/pt-br/aspnet/core/?view=aspnetcore-8.0>. Acesso em: 30 set. 2024.

_____. **Azure Firewall documentation.** Disponível em: <https://learn.microsoft.com/pt-br/azure/firewall/>. Acesso em: 15 set. 2024.

_____. **Documentação Técnica do SQL Server**. Disponível em: <https://learn.microsoft.com/pt-br/sql/sql-server/?view=sql-server-ver15>. Acesso em: 28 set. 2024.

MDN WEB DOCS**. Gerenciamento de Conexão em HTTP/1.x** Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Connection_management_in_HTTP_1.x>. Acesso em: 10 nov. 2024.

MONGODB**. Documentação do MongoDB.** Disponível em: <https://www.mongodb.com/pt-br/docs/>. Acesso em: 10 set. 2024

NEWMAN, Sam. **Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith**. Sebastopol: O'Reilly Media, 2019.

OMG STANDARDS DEVELOPMENT ORGANIZATION. **Business Process Model and Notation.** Version 2.0**,** 2011. Disponível em: <https://www.omg.org/spec/BPMN/2.0/>. Acesso em: 30 mai. 2024.

OLIVEIRA, Miguel. **Webhook vs WebSocket: comparação detalhada que você deve saber.** Disponível em: <https://apidog.com/pt/blog/webhook-vs-websocket/>. Acesso em: 5 set. 2024.

PM3. **Design Thinking: o que é, etapas e como aplicar.** Disponível em: <https://www.cursospm3.com.br/blog/design-thinking-guia-o-que-e-etapas-como-aplicar/>. Acesso em: 1 mai. 2024.

REDIS. **Develop with Redis.** Disponível em: <https://redis.io/docs/latest/>. Acesso em: 23 set. 2024.

REIS, Paulo. **Ciência do Artificial e Design Science Research**. Agência UFRJ de Inovação, 2019. Disponível em: <https://inovacao.ufrj.br/images/vol_22_ciencia_artificial_design_science_research_2019.pdf>. Acesso em: 9 dez. 2024.

REN, Frank. **Geosharded Recommendations Part 1: Sharding Approach**. Disponível em: <https://medium.com/tinder/geosharded-recommendations-part-1-sharding-approach-d5d54e0ec77a>. Acesso em: 15 ago. 2024.

SHIVAKUMAR, S. K.; SETHII, S. **Building Digital Experience Platform: A Guide to Developing Next-Generation Enterprise Applications**. California: Apress Berkeley, 2019

SOUZA, Alex Sandro Alves de. **Estudo de viabilidade para implantação de um aplicativo de carona compartilhada na Universidade Federal do Ceará do Campus Quixadá**. Ceará, 2023.

VALENTE, Marco Tulio. **Engenharia de Software Moderna**. Publicação independente, 2022.

VUE**.** ***The Progressive JavaScript Framework***. Disponível em: <https://vuejs.org/guide/introduction.html>. Acesso em: 5 ago. 2024.

VUETIFY**. Get started with Vuetify 3**. Disponível em: <https://vuetifyjs.com/en/introduction/why-vuetify/#what-is-vuetify3f>. Acesso em: 5 ago. 2024.

YAMAMOTO, Erika. **Circulares da Cidade Universitária transportam 28 mil passageiros por dia**. Jornal USP, 2024. Disponível em: <https://jornal.usp.br/universidade/circulares-da-cidade-universitaria-transportam-28-mil-passageiros-por-dia/>. Acesso em: 9 dez. 2024.