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

RODRIGO FREITAS LEITE

# SELETO: Sistema de Motorista Particular

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

Orientador: Prof. Dr. Leandro Krug Wives

Porto Alegre
2017

*“If I have seen further than others,*
*it is by standing upon the shoulders of giants.“*
— Isaac Newton

## AGRADECIMENTOS

Agradeço a todas as pessoas que me apoiaram para a construção deste trabalho. Após inúmeras dificuldades que me assolaram nos últimos tempos, este trabalho não pertence somente a mim, mas, principalmente, a minha família especialmente para minha vó, mãe e tia.

Agradeço também ao professor Leandro Krug Wives, meu orientador, o qual compreendeu toda a minha situação e me apoio para conseguir concluir este trabalho.

Minha gratidão também aos meus sócios e a toda família Kobe pelo suporte e apoio ao longo dos anos. Sem o comprometimento de vocês não seria possível trabalhar e cursar esta graduação.

Gostaria de agradecer imensamente a todos os professores e funcionários do INF por me atenderem sempre em prontidão para esclarecimento de qualquer dúvida ou problema. Foi uma enorme satisfação e honra fazer parte desta instituição.

## RESUMO

Atualmente, o uso de dispositivos móveis está presente na vida de todas as pessoas e, com a distribuição de internet móvel entre os usuários, novos negócios do tipo B2C e C2C surgiram. Nesse contexto, diversas soluções puderam ser desenvolvidas para problemas que antes, ou não tinham como serem resolvidos, ou ainda não existiam de fato. Nesse sentido, nos últimos anos, a locomoção tem sido uma das maiores necessidades urbanas. Desse modo, o trabalho desenvolvido apresenta um sistema cujo objetivo é facilitar a locomoção para passageiros e auxiliar os motoristas profissionais a encontrar esses possíveis clientes. A solução é composta de dois aplicativos iOS (um para o passageiro e outro para o motorista) e um servidor desenvolvido em uma plataforma BaaS.



**Palavras-chave:** IOS. BaaS. B2B.

**SELETO - Private Driver System**

## ABSTRACT

Currently, the use of mobile devices is present in the lives of all people. With the distribution of mobile internet amongst users, new businesses of type B2B and B2C have emerged. In this context, various solutions were developed for problems that either did not exist before, or that did not have solutions yet. In the latest years, the locomotion became one of the biggest urban necessities. Thus, the work developed consists of a mobile system whose goal is to help passengers to move around and drivers to find their customers. The solution is formed by two iOS applications (one for the passenger and one for the driver) and a web server developed on a BaaS platform.

**Keywords:** iOS, BaaS.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | Application Programming Interface |
| BaaS | Backend as a Service |
| B2B | Business 2 Business |
| C2C | Consumer 2 Consumer |
| HTTP | HyperText Transfer Protocol |
| JSON | JavaScript object notation |
| MVC | Model-View-Controller |
| MVP | Minimum Value Product |
| MVVM | Model-View-View-Model |
| REST | Representational State Transfer |
| RFC | Request For Comments |
| SQL | Structured Query Language |
| UFRGS | Universidade Federal do Rio Grande do Sul |
| UML | Unified Modeling Language |

**SUMÁRIO**

# 1 INTRODUÇÃO

A utilização de dispositivos móveis como meio principal de conexão à internet é crescente. De acordo com dados do PNAD (2015), em 2014, 80,4% das casas brasileiras se conectaram à internet através de telefones móveis. Em razão dessa facilidade de acesso à rede, os celulares já se tornaram um dos principais meios de interação entre as pessoas e, através de aplicativos, auxiliam em diversas tarefas do dia-a-dia, desde as mais básicas, como ler um e-mail, até as mais complexas, como realizar transações bancárias. Segundo estudo de Flurry (2017), em 2015, o uso de aplicativos mobile cresceu 58% em relação a 2014. O estudo também mostrou que os consumidores gastam cerca de 4,9 horas em *smartphones* e *tablets*, e, desse tempo, 92% é gasto em aplicativos.

Dentro desse contexto, tem-se uma gama enorme de negócios e serviços sendo realizados entre consumidores, o *Client to Consumer* (C2C), ou seja, transação em que dois ou mais usuários (um que necessita de determinado serviço e outro que presta o serviço) são aproximados, nesse caso, por um aplicativo.

Nesse modelo, diferentes tipos de problemas podem ser estudados e solucionados, como, por exemplo, a locomoção de pessoas em uma grande cidade. Segundo a pesquisa KantarTNS (2017), até 2050, 70% da população mundial estará vivendo em centros urbanos. Dessa forma, a mobilidade urbana é uma das grandes questões a serem resolvidas devido as dificuldades que as pessoas e as cidades já enfrentam com congestionamentos, custo de veículos ou problemas ambientais. A pesquisa também traz o dado de que 75% das pessoas que vivem em áreas urbanas utilizam aplicativos para organizar ou orientar seu trajeto. Os aplicativos usados têm foco em facilitar a navegação das pessoas, gerando benefícios como preço e conveniência.

Entretanto, apesar dos diversos aplicativos criados para atender essa necessidade de mobilidade urbana, a questão da falta de segurança, pelo aumento da violência, como mostra o estudo do IPEA (2017), ainda é um fator preocupante, tanto para os motoristas que oferecem o serviço, quanto para os passageiros que vivem nos grandes centros urbanos.

## 1.1 Motivação

Conforme relatado na introdução deste trabalho, o número de pessoas que utilizam dispositivos móveis vem aumentando nos últimos anos, assim como também cresceu o uso de aplicativos. Em relação a mobilidade urbana especificamente, segundo a pesquisa KantarTNS (2017), 3/4 dos cidadãos das grandes cidades usam aplicativos para organizar ou orientar seu trajeto e a possibilidade de acessar um serviço *pague por viagem* com um simples toque em um aplicativo de celular acelerou a mobilidade global.

Levando em conta todos os dados apresentados até aqui, KantarTNS (2017) e PNAD (2015) mostram que, mesmo com a proliferação de aplicativos de mobilidade urbana, ainda há um mercado significativo para ser explorado por esse modelo de negócio. Além disso, existe a percepção de que os aplicativos existentes ainda não conseguem diminuir a sensação de insegurança de passageiros e motoristas que utilizam estas ferramentas.

## 1.2 Objetivo

Este trabalho apresenta a construção de um aplicativo na plataforma iOS assim como o processo de desenvolvimento aplicado durante todo o trabalho, que compreende o estudo das tecnologias envolvidas, a apresentação da solução, e a implementação das funcionalidades discutidas durante o trabalho.

## 1.3 Organização do texto

O trabalho está organizado nos seguintes capítulos que nortearam os assuntos relevantes ao processo de construção da solução:

**Capítulo 1** corresponde a esta introdução que apresentou a motivação que levou a construção desta solução, os objetivos a serem tratados durante o trabalho e as escolhas tecnológicas utilizadas.

**Capitulo 2**. Apresenta os conceitos, ferramentas e tecnologias utilizados durante o desenvolvimento da aplicação, assim como os fatores de escolha delas.

**Capítulo 3**. Apresenta trabalhos relacionados com o tema proposto, a plataforma móvel escolhida para o desenvolvimento do aplicativo e outras tecnologias escolhidas para o desenvolvimento da aplicação.

**Capítulo 4**. Apresenta uma visão geral sobre o desenvolvimento do aplicativo, como sua metodologia, aspectos arquiteturais, modelagem do sistema, *business model* e servidor.

**Capitulo 5**. Apresenta a conclusão do trabalho e todas as considerações sobre o aplicativo Seleto, assim como pontos a melhorar e a serem explorados futuramente no projeto.

# 2 CONCEITOS E TECNOLOGIAS RELACIONADAS

Neste capítulo são apresentados os conceitos, as ferramentas e as tecnologias utilizadas para o desenvolvimento da solução proposta. Os fatores para a escolha das ferramentas e tecnologias estão relacionados com a difusão das mesmas no desenvolvimento de projetos para aplicativos móveis, suas características relacionadas com as necessidades do projeto e o custo atribuído a sua utilização.

Como a solução proposta caracteriza-se no desenvolvimento de um aplicativo mobile para o sistema operacional iOS, será utilizando o Xcode, que é o ambiente oficial de desenvolvimento da Apple. Será utilizado, também, um servidor Parse Server, para processar e armazenar todas as informações existentes no sistema. O aplicativo será desenvolvido de forma nativa, possibilitando explorar ao máximo todos os recursos disponibilizados pelo sistema operacional iOS.

Além disso, também serão utilizadas as ferramentas Crashlytics e Answers, disponíveis na plataforma Fabric (detalhes a seguir). Essas ferramentas são importantes para monitorar possíveis *bugs* ou defeitos, o desempenho da aplicação e o comportamento dos usuários. Já para o desenvolvimento do servidor, será utilizado o Parse Server hospedado no Back4App.

Diversas outras tecnologias poderiam ser utilizadas na elaboração do projeto, porém, as tecnologias mencionadas foram definidas devido a sua importância no mercado e pelo conhecimento técnico adquirido ao longo do tempo para o desenvolvimento da solução.

## 2.1 iOS

Para o desenvolvimento do aplicativo, será utilizada a plataforma iOS, pois existe um conhecimento prévio adquirido pelo autor na plataforma. Portanto, aplicar modificações durante o processo de desenvolvimento, caso seja necessário, pode ser executado de maneira rápida. Além disso, há uma alta aceitação de novas versões do sistema operacional pelos usuários, como pode ser analisado em (MIXPANEL, 2017), 96% dos usuários tem a última versão do sistema operacional instalado, logo é possível utilizar as últimas tecnologias e frameworks oferecidas pela plataforma, facilitando o desenvolvimento do aplicativo.

## 2.2 Fabric

Fabric[1] é uma plataforma desenvolvida pelo Twitter, recentemente, adquirida pela Google. Entre seus principais clientes estão Spotify, SoundCloud, Ebay e outras empresas. Através desse serviço, diversas ferramentas são disponibilizadas, como monetização, autenticação, distribuição entre outros serviços podem ser encontrados em (FABRIC, 2017). Neste projeto, serão utilizadas as ferramentas Crashlytics e Answers, as quais auxiliam no mapeamento de erros e na detecção do comportamento dos usuários. O Crashlytics destaca-se pela identificação e análise de erros ocorridos na aplicação. Já o Answers consiste no mapeamento do comportamento dos usuários utilizando o aplicativo. Desse modo, é possível identificar quais partes do sistema são mais utilizadas e como os componentes da solução podem ser alterados para fornecer uma melhor usabilidade. Esse serviço é indispensável para se obter *feedback* de como o usuário está utilizando o sistema, assim como detectar erros que podem ocorrer à medida que os usuários utilizam o sistema.

## 2.3 Parse Server

Parse é uma plataforma de código aberto que provê serviços de *backend* para aplicações. Desse modo, desenvolvedores podem abstrair a criação e configuração do servidor, administração do banco de dados, desenvolvimento de APIs, armazenamento de

[1]Disponível em https://fabric.io

dados, controle de sessão, autenticação de usuários e outras responsabilidades atribuídas ao *backend* (MAROTTO, 2016). O Parse Server foi construído utilizando Node e MongoDB. O Parse server é uma plataforma BaaS (*Backend as a service*), ou seja, um conjunto de bibliotecas e frameworks, o qual provê uma solução de banco de dados e lógica na nuvem. O Parse Server foi escolhido como solução de *backend* para o aplicativo, pois desenvolver e validar toda criação do servidor demandaria um tempo considerável na criação do MVP. Todavia, isso não invalida no futuro o desenvolvimento de um *backend* customizado para o Seleto.

## 2.4 MongoDB

MongoDB é um banco de dados *open-source* que se caracteriza por guardar documentos em formato JSON[2] provendo alta performance e disponibilidade. Outra característica importante está relacionada com os dados armazenados no banco de dados. O MongoDB possui um esquema flexível, ou seja, é diferente de sistemas de gerenciamento de bancos dados SQL, onde é necessário definir as estruturas das tabelas antes de inserir os dados. Dessa forma, é possível adicionar e alterar as estruturas existentes de acordo com o desenvolvimento de novas funcionalidades no sistema. Apesar de o MongoDB não ser acessado diretamente pelo desenvolvedor, uma vez que o Parse Server apenas mostra os dados através de um portal web, é necessário entender de que se trata de um banco de dados do tipo *NoSQL*, o que pode afetar o seu desenvolvimento, em especial o projeto e a modelagem dos dados. Portanto, deve-se modelar as coleções de maneira correta, evitando o máximo de *joins* entre os documentos contidos no banco de dados.

## 2.5 Back4app

Back4app é um serviço oferecido para a hospedagem e configuração de servidores do tipo Parse Server. Com a utilização desse serviço, os usuários não precisam criar e realizar a manutenção de servidores para hospedar a sua aplicação. A plataforma é responsável pelo gerenciamento e monitoramento dos aplicativos criados na mesma. Todos os sistemas são hospedados na Amazon. Além de garantir o funcionamento dos aplicativos, o Back4app também dispõe de outros mecanismos para auxiliar os desenvolvedores,

[2] *JavaScript Object Notation*

como a configuração e envio de notificações para smartphones e a criação de API[3] customizadas. Esse gerenciamento de infraestrutura é altamente complicado, e depende, muitas vezes, de um time de profissionais que cuide de diversos aspectos, tais como espaço disponível para dados, memória utilizada pelo servidor, número de requisições, entre outros. Portanto a utilização do Back4app é essencial para focar no desenvolvimento da aplicação.

## 2.6 Swift

A linguagem de programação utilizada para o desenvolvimento da plataforma iOS é a Swift. Criada em 2014 pela Apple, a linguagem possui foco na eficiência e em auxiliar os desenvolvedores a escreverem códigos mais seguros e confiáveis. Segundo seus desenvolvedores, ela possui uma sintaxe limpa e moderna, oferece acesso contínuo aos códigos e estruturas em $C$ e $Objective-C$ existentes. Swift proporciona características de linguagens modernas como, por exemplo, protocolos, *closures* e *generics*, de acordo com (SWIFTGITHUB, 2017). Além disso, apresenta um ótimo desempenho se comparada com sua antecessora, a linguagem $Objective-C$, também da Apple. Segundo (APPLE, 2017), em um algoritmo de pesquisa de dados, o Swift apresentou até 2,6x mais eficiência em relação à sua precursora Objective-C. Outra característica importante é a disponibilidade da linguagem em código aberto, possibilitando a expansão de novas funcionalidades frequentemente.

## 2.7 WebSocket

WebSocketRFC é um protocolo para uso de comunicação full-duplex entre servidor e cliente sobre uma única conexão TCP (WEBSOCKETRFC, 2011). O protocolo de WebSocket habilita a interação entre cliente e servidor com pouco *overhead*, facilitando aplicações em tempo real, seja enviando ou recebendo dados do cliente. Logo, o servidor pode enviar dados para o cliente sem ele ter solicitado e permitir que dados sejam enviados enquanto a conexão permanece aberta.

[3] *Application Programming Interface*

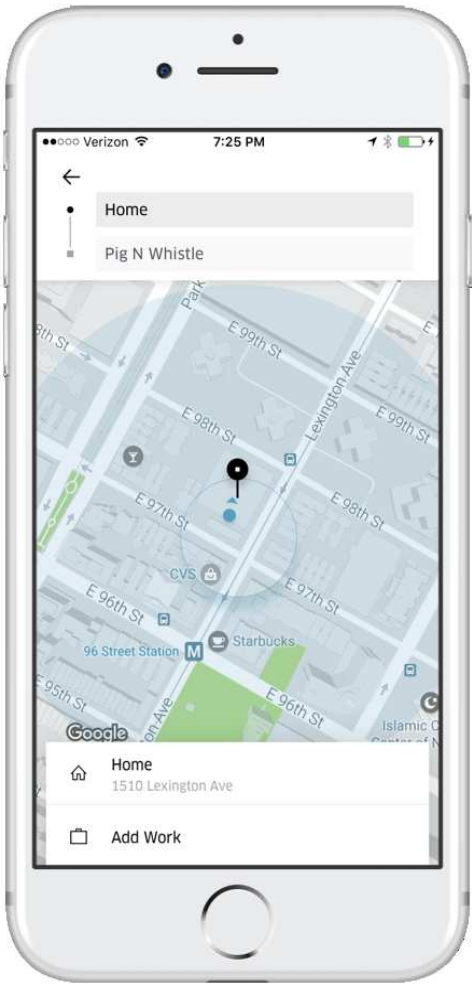
# 3 TRABALHOS RELACIONADOS

Após as pesquisas sobre mobilidade realizadas pela KantarTNS (2017) e a disponibilização do índice de cidades violências do IPEA (2017), outras soluções já foram propostas para ajudar a resolver o problema da mobilidade urbana, uma vez que a população em grandes centros urbanos aumentará no futuro. Logo, neste capítulo são apresentados alguns sistemas similares que podem ser utilizados como referência para a solução proposta e para elencar suas principais características.

## 3.1 Uber

Uber é um aplicativo com sede nos Estados Unidos, disponível para as plataformas iOS, Android e Windows Phone e cujo objetivo é conectar motoristas e passageiros. O Uber foi um dos precursores neste ramo e atualmente é usado mundialmente. O sistema funciona com segmentação de serviços, um preço econômico para carros populares e um preço custoso para carros de alto valor. O Uber aceita pagamentos por cartão de crédito e dinheiro. No Brasil, o custo da tarifa para os passageiros quando a operação começou já era atrativa, todavia após algum tempo de funcionamento esta tarifa diminui, o que aumentou a procura pelo serviço. Todavia a taxa cobrada sobre as corridas continua o mesmo desde a implantação do serviço, criando uma insatisfação perante aos motoristas que acabaram faturando menos a cada corrida.

Figura 3.1: Chamada de corrida no aplicativo do Uber

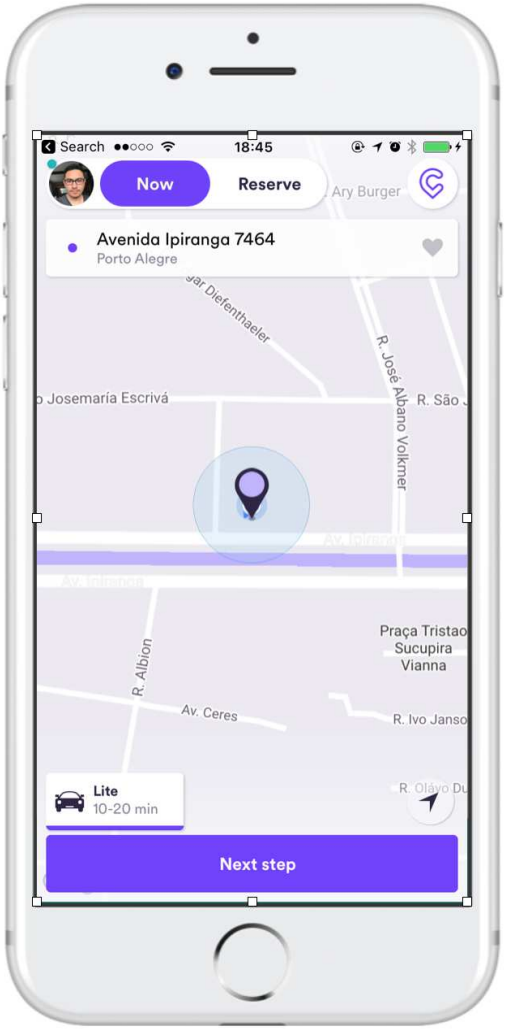



Fonte: App Store

## 3.2 Cabify

Cabify é um aplicativo com atuação na América latina disponível para iOS e Android. Um grande diferencial do aplicativo é que a tarifa não é variável. Uma vez que seja calculado o valor a ser cobrado entre os dois pontos, este será aplicado ao cliente, não importando quanto tempo a corrida levará para acabar. A plataforma aceita somente pagamento com cartão de crédito e o cadastramento dos motoristas é mais rigoroso que as outras plataformas, a fim de manter a excelência prestada pelos motoristas. O aplicativo não atende a todos os lugares da cidade, ele atua somente em algumas zonas e gradativamente aumenta seu alcance dependendo do seu interesse.

Figura 3.2: Chamada de corrida no aplicativo do Cabify

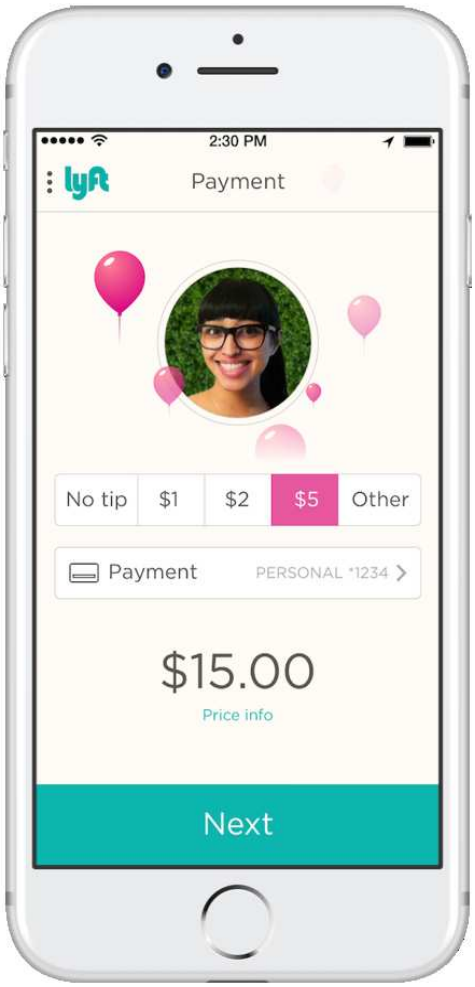



Fonte: App Store

## 3.3 Lyft

Lyft é um aplicativo com atuação no Estados Unidos disponível para iOS, Android, Windowns Phone e Amazon. Assim como o Uber, o Lyft funciona com a categorização de carros e preços. O pagamento fornecido pelo aplicativo é somente em cartão de crédito e o serviço pelo qual é reconhecido é o *pool*, no qual dois ou mais passageiros podem ou não serem levados para o mesmo destino pelo motorista e a tarifa é compartilha entre todos que utilizam o serviço. Os passageiros podem partir da mesma localização ou podem ser encontrados ao longo do percurso.

Figura 3.3: Lyft - Tela de pagamento

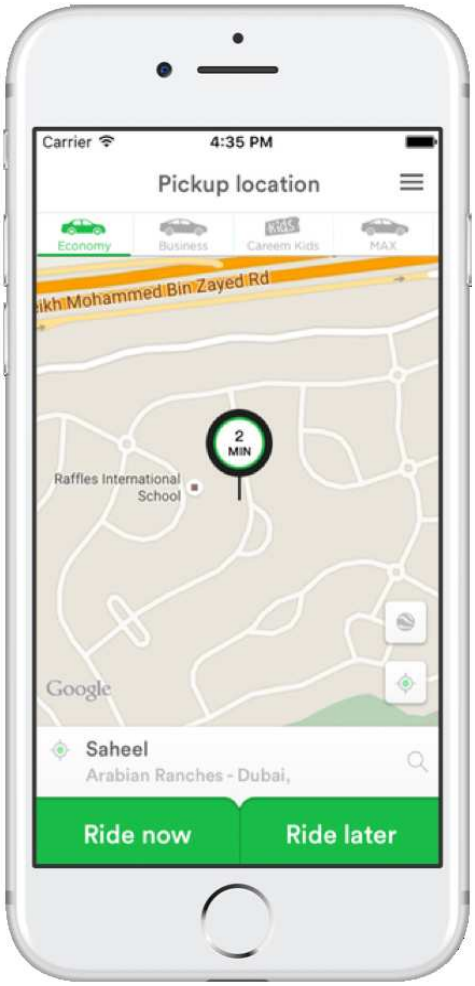



Fonte: App Store

## 3.4 Careem

Careem é um aplicativo com atuação no oriente e atua em diferentes cidades como Dubai, Alexandria, entre outras. Além de oferecer serviço de viagem segmentado, atua também no mercado de entrega de objetos, logo o passageiro pode escolher transportar e entregar encomendas. O aplicativo aceita pagamento tanto em cartão de crédito quanto em dinheiro. O aplicativo após registrar os motoristas e automóveis prove um treinamento para entender e utilizar o aplicativo como motorista.

Figura 3.4: Careem - Pesquisar corrida




Fonte: App Store

## 3.5 Funcionalidades

Baseado nos aplicativos analisados, criou-se uma tabela 3.1 comparativa dos serviços prestados por cada aplicativo. Analisando-se as funcionalidades expostas acima, percebe-se que nenhum dos aplicativos avalia se o passageiro é confiável ou não. A validação do passageiro é imprescindível para dar mais segurança para o motorista que trafega pela cidade, sendo eles considerados locais violentos ou não. Portanto, foi definido que o aplicativo Seleto será criado incluindo validação do cadastro dos passageiros e dos motoristas. Na avaliação dos aplicativos estudados também se constatou que dois apresentam a funcionalidade de pagamento em dinheiro. Fator que gera sensação de insegurança, tanto para o motorista quanto para o passageiro, o aplicativo Seleto terá pagamento somente pelo cartão de crédito para que seja um sistema mais seguro e confiável.

Tabela 3.1: Quadro comparativo das funcionalidades dos aplicativos

| Funcionalidades | | | | |
|---|---|---|---|---|
| | Uber | Cabify | Lyft | Careem |
| Categorização | Sim | Sim | Sim | Sim |
| Preço variável | Sim | Não | Sim | Sim |
| Pagamento em dinheiro | Sim | Não | Não | Sim |
| Pagamento em cartão | Sim | Sim | Sim | Sim |
| Validação de passageiros | Não | Não | Não | Não |

Fonte: Autor

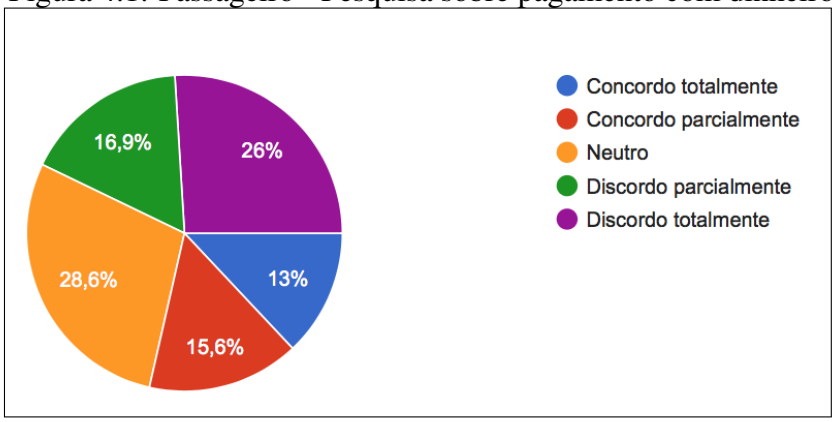
# 4 PROJETO SELETO

O projeto Seleto consiste em outra solução perante os serviços existentes. A insegurança dos motoristas e dos passageiros é uma reclamação constante, assim como a cobrança variável ao longo do dia, de acordo com a Associação de Motoristas Autônomos por Aplicativo (AMAA, 2017). Para ratificar essa percepção, foi feita uma pesquisa sobre o uso de aplicativos de mobilidade urbana na cidade de Porto Alegre, no mês de junho de 2017, com um total de 54 motoristas e 87 passageiros. A pesquisa dividida em duas: denominadas de ”Pesquisa Passageiro"e ”Pesquisa Motorista". Ambas foram realizadas no *Google Forms*, mas podem ser encontradas nos Apêndices $A$ e $B$ deste trabalho.

## 4.1 Pesquisa Passageiro

Em relação aos passageiros, foi constatado que a forma de pagamento é um fator que influencia na sensação de segurança quando se utiliza o serviço. Na afirmação: *Eu me sinto seguro pagando em dinheiro quando utilizo aplicativos de mobilidade urbana*, 42,9% dos passageiros responderam que se sentem parcialmente ou totalmente inseguros fazendo pagamentos em dinheiro. Já 25,6% informaram que se sentem totalmente ou parcialmente seguros e 28,6% disseram ser indiferentes ao tipo de pagamento, conforme demonstrado no gráfico da Figura 4.1 seguinte.

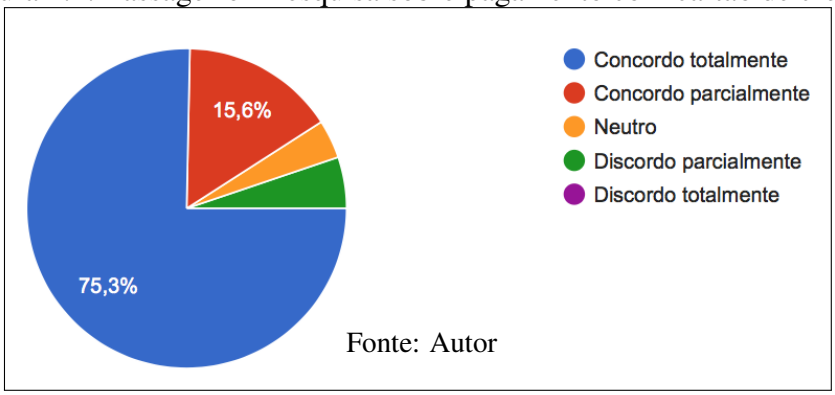
Figura 4.1: Passageiro - Pesquisa sobre pagamento com dinheiro




Fonte: Autor

Já na afirmação: *Eu me sinto seguro pagando com cartão de crédito quando utilizo aplicativos de mobilidade urbana*, 90,9% informaram que se sentem total ou parcialmente seguros ao final da corrida, enquanto que 5,6% disseram que não se sentem totalmente seguros pagando desta forma, e apenas 4,2% disseram ser indiferentes a essa situação (ver Figura 4.2).

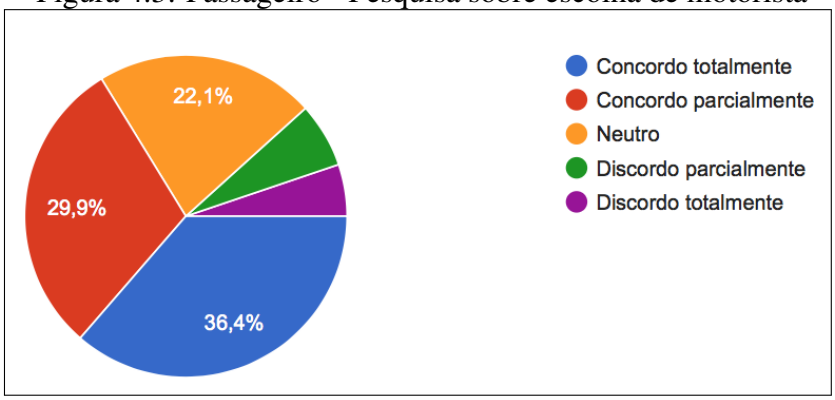
Figura 4.2: Passageiro - Pesquisa sobre pagamento com cartão de crédito




Fonte: Autor

Outro fator abordado na pesquisa foi em relação a possibilidade de escolha do motorista pelo passageiro. Das 87 pessoas pesquisadas, 66,3% gostariam que o aplicativo permitisse a escolha do motorista. Além disso, das 44 mulheres que responderam à pesquisa, 41% disseram que gostariam de poder escolher o motorista, enquanto apenas 20% dos homens se importaram com essa escolha (ver Figura 4.3).

Figura 4.3: Passageiro - Pesquisa sobre escolha de motorista

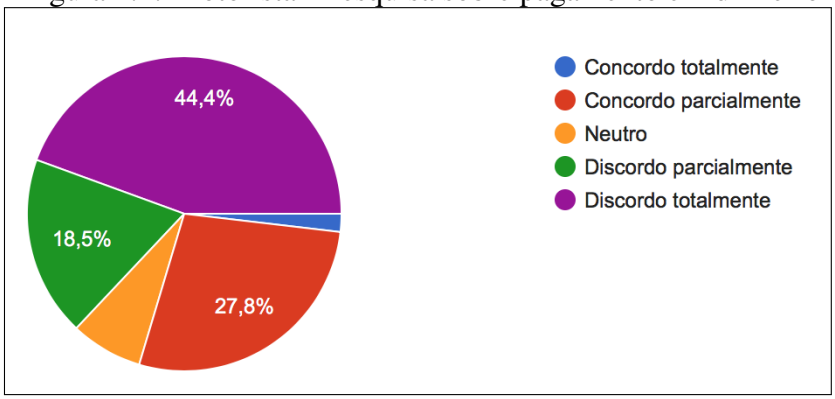



Fonte: Autor

### 4.1.1 Pesquisa Motoristas

Na pesquisa direcionada aos motoristas, a forma de pagamento também foi uma preocupação relevante. Dos 54 motoristas entrevistas, conforme pode ser verificado na Figura 4.4, 18,5% não se sentem parcialmente seguros, enquanto 44,4% não se sentem totalmente seguros quando têm que pegar uma corrida em que o pagamento será feito em dinheiro, já 27,8% se sentem parcialmente seguros.

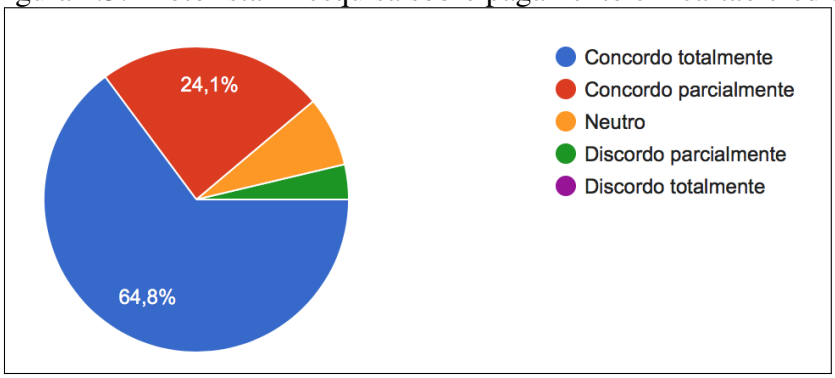
Figura 4.4: Motorista - Pesquisa sobre pagamento em dinheiro




Fonte: Autor

No entanto, quando o pagamento é ofertado com cartão de crédito, 64,8% dos entrevistados se sentem totalmente seguros, 24,1% se sentem parcialmente seguros e 7,4% são indiferentes nesta situação (conforme Figura 4.5).

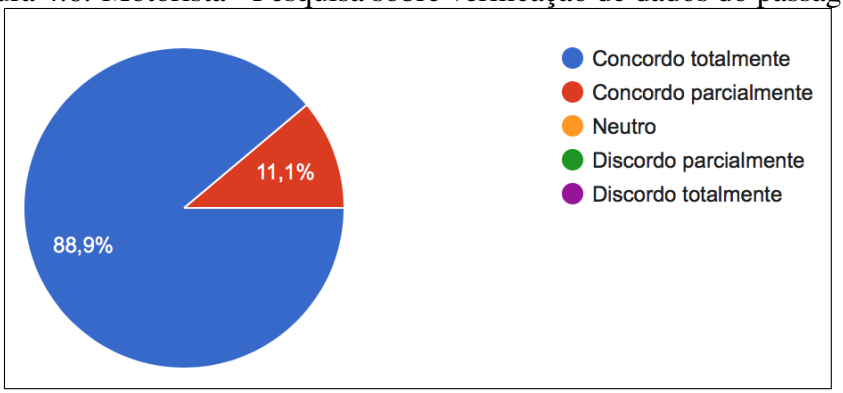
Figura 4.5: Motorista - Pesquisa sobre pagamento em cartão crédito




Fonte: Autor

A Figura 4.6 ilustra que outro fator que pode influenciar na sensação de segurança dos motoristas é a checagem das informações dos passageiros. Quando perguntado aos motoristas se sentiriam-se seguros caso o aplicativo checasse os dados do usuário antes de incluí-lo na plataforma, 88,9% concordaram totalmente e 11,1% concordaram parcialmente.

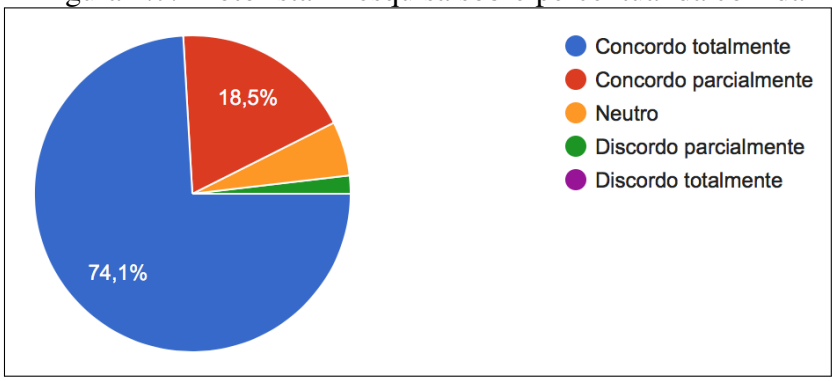
Figura 4.6: Motorista - Pesquisa sobre verificação de dados do passageiro




Fonte: Autor

Já em relação a possibilidade do aplicativo indicar rotas alternativas para evitar lugares inseguros na cidade (Figura 4.7), 18,5% informaram que se sentiriam parcialmente seguros e 74,1% não se sentiriam seguros com essa funcionalidade.

Figura 4.7: Motorista - Pesquisa sobre percentual da corrida




Fonte: Autor

Dessa forma, vê-se a possibilidade de desenvolver um aplicativo de mobilidade urbana que ofereça recursos que proporcionem mais segurança, tanto para o motorista, quanto para o passageiro que o utilizam.

## 4.2 Metodologia

Para coordenação e gerenciamento durante o desenvolvimento do trabalho, foi utilizada *Scrum*, uma metodologia de desenvolvimento ágil, descrita em sua essência (RU-

BIN, 2012). Todavia não foi utilizado tudo que a metodologia oferece, pois nesse caso, apenas alguns aspectos seriam necessários, como a criação de um *backlog*, *sprints* de tamanho fixo, em torno de duas semanas, e *sprint review*. Quando o processo foi iniciado, foi criado um *backlog* do produto a fim de gerar todas as tarefas necessárias e ao fim de duas semanas era organizada uma reunião para organizar o direcionamento da construção do aplicativo. Para delimitar e entender a evolução das tarefas em desenvolvimento foi utilizada a ferramenta Trello[1], que é um *Kanbam* em que é possível analisar as tarefas que ainda precisam ser feitas, as que estão em andamento e as que foram finalizadas.

## 4.3 Modelagem do Sistema

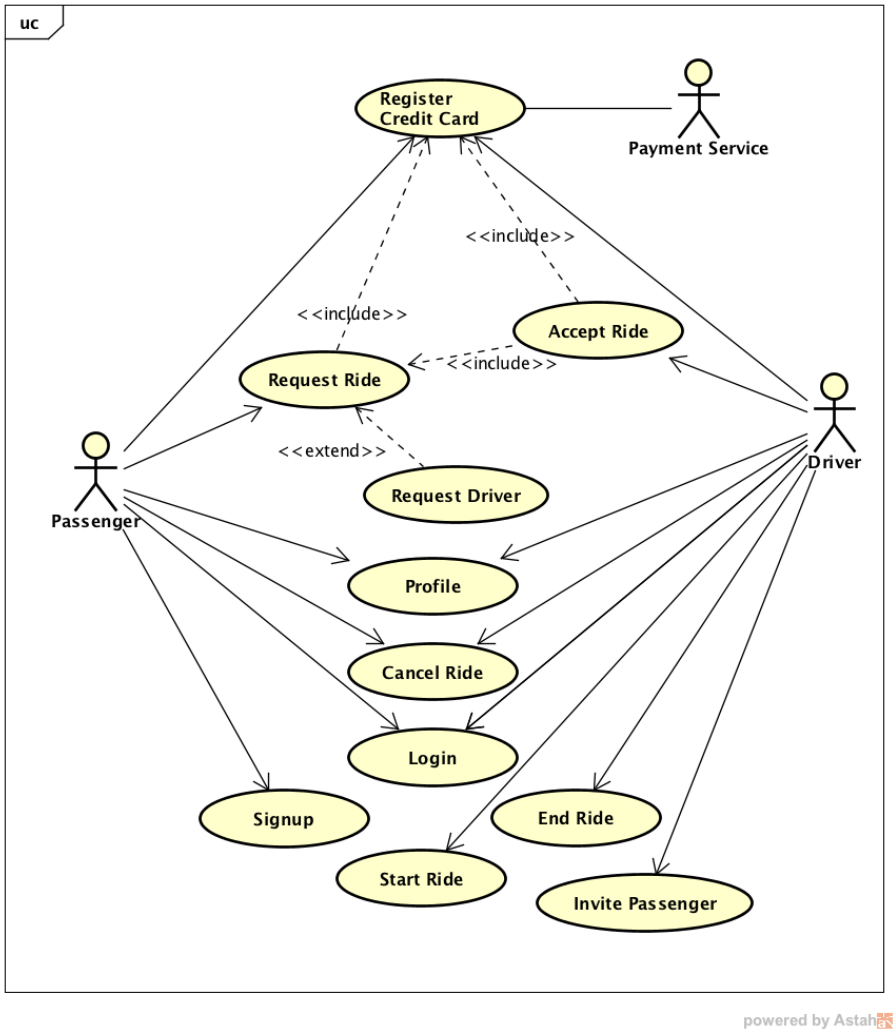
Nesta seção é apresentada a modelagem do sistema utilizando a linguagem UML. Os diagramas utilizados serão dos seguintes tipos: caso de uso e de atividade. Dessa forma, serão identificados todos os casos de uso da solução proposta, mas os principais serão descritos de forma mais completa.

### 4.3.1 Casos de Uso

O diagrama de casos de uso é responsável por identificar os atores e as funcionalidades do sistema, além das interações e perspectivas dos atores perante as funcionalidades.

[1] https://trello.com

Figura 4.8: Casos de uso




Fonte: Autor

## 4.4 Diagrama de Atividades

Esta seção apresenta diagramas de atividades para os principais casos de uso do aplicativo. Com isso, será possível identificar as atividades realizadas pelos atores, pelo sistema, além de reconhecer as pré e pós condições necessárias para entender o resultado obtido.

### 4.4.1 Diagrama de Casos de Uso - *Request Ride*

O caso de uso *Request Ride* é o evento que ocorre quando um passageiro requisita um motorista. O passageiro pode requisitar uma corrida a qualquer momento que deseje

pelo aplicativo. O passageiro pode escolher um motorista, caso este esteja disponível para ir ao seu encontro e atender ao chamado, o passageiro será atendido por este motorista. Todavia, se nenhum motorista estiver disponível, o sistema informará ao usuário e escolherá outro motorista para atendê-lo automaticamente. Para entender melhor o caso de uso *Request Ride*, este é descrito abaixo.

Tabela 4.1: Caso de Uso - *Request Ride*

| *Request Ride* | |
|---|---|
| Pré-condições | • Passageiro deve estar logado no sistema.<br>• Passageiro deve registrar o seu cartão de crédito no sistema. |
| Atores | *Passenger* |
| Pós-condições | • Motorista deve se dirigir para a localização do passageiro.<br>• Passageiro deve aguardar motorista. |
| Fluxo Principal | • Passageiro deve inserir seu local.<br>• Passageiro deve inserir seu destino.<br>• Passageiro pode escolher um motorista.<br>• Passageiro deve ver o preço da corrida.<br>• Passageiro deve requisitar a corrida.<br>• Passageiro deve ser contemplado com um motorista. |

Fonte: Autor

### 4.4.2 Caso de Uso - *Accept Ride*

O caso de uso *Accept Ride* é o evento que ocorre quando um motorista aceita uma corrida. O motorista será notificado pelo servidor quando for requisitado para uma corrida. O motorista tem a opção de aceitar ou negar o pedido. Caso o motorista venha a aceitar a corrida, existe a possibilidade de outro motorista já ter aceitado anteriormente. Caso isto venha a acontecer, o servidor informará o motorista do ocorrido.

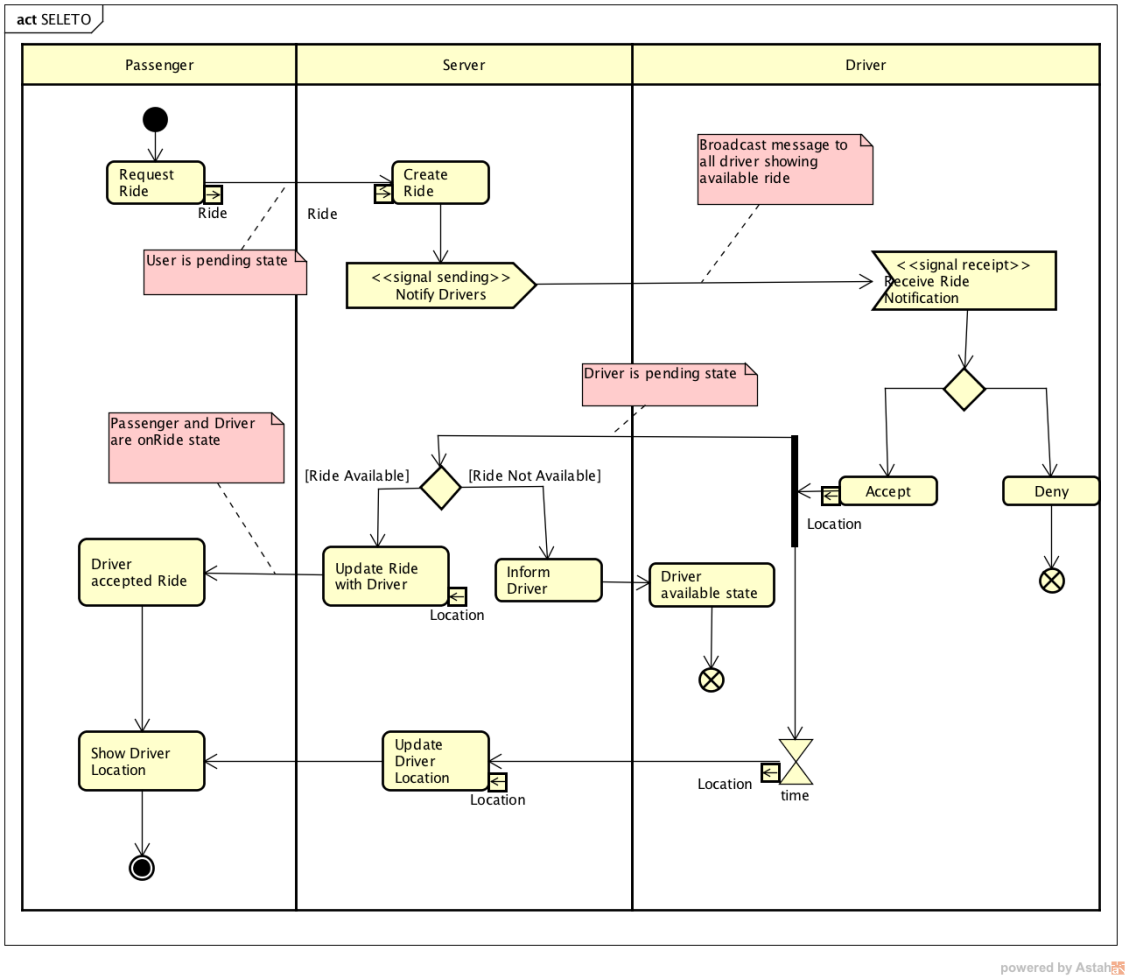
Tabela 4.2: Caso de Uso - *Accept Ride*

| *Accept Ride* | |
|---|---|
| Pré-condições | • Motorista deve estar logado no sistema.<br>• Motorista deve ter registrado seu cartão de crédito. |
| Atores | *Driver* |
| Pós-condições | • Motorista deve se dirigir para a localização do passageiro. |
| Fluxo Principal | • Sistema notifica motorista.<br>• Motorista deve aceitar corrida.<br>• Motorista deve enviar sua localização para o servidor de tempos em tempos, assim o servidor pode mostrar ao passageiro onde ele se encontra. |

Fonte: Autor

Os casos de uso *Request Ride* e *Accept Ride* são dependentes um do outro, ou seja, para que o caso de uso *Accept Ride* aconteça, é necessário que o caso de uso *Request Ride* aconteça previamente. O aplicativo tem o intuito de funcionar online, caso o passageiro ou o motorista percam a conexão com a internet, é necessário sincronizar com o servidor e atualizar o estado do aplicativo do passageiro ou do motorista. Assim sendo, o diagrama de atividades abaixo demostra o fluxo inteiro dos dois casos de uso.

Figura 4.9: *Request Ride & Accept Ride* - Diagrama de atividades




Fonte: Autor

### 4.4.3 Caso de Uso - *Cancel Ride*

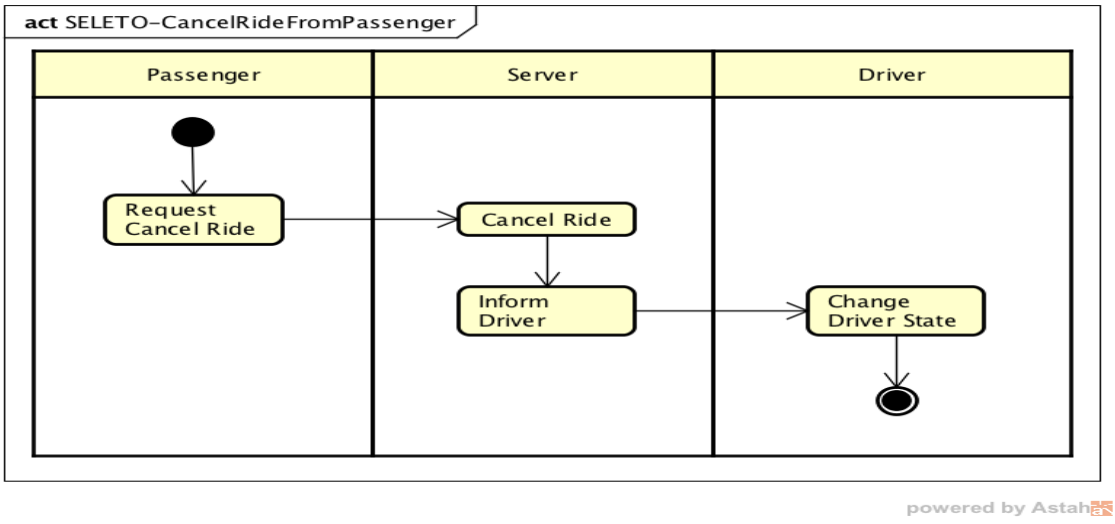
O caso de uso *Cancel Ride* pode acontecer tanto pelo passageiro quanto pelo motorista. Quando o passageiro inicia uma corrida, ele pode cancelar e o motorista, ao pegar uma corrida, também tem o direito de cancelar esta corrida.

Tabela 4.3: Caso de Uso - *Cancel Ride*

| *Cancel Ride* | |
|---|---|
| Pré-condições | • Motorista e Passageiro devem estar logados no sistema.<br>• Motorista e Passageiro devem ter registrados seus cartões de crédito. |
| Atores | *Driver ou Passageiro* |
| Pós-condições | • Motorista e passageiro devem estar no estado inicial. |
| Fluxo Principal | • Motorista e passageiro devem ser informados pelo sistema que a corrida foi cancelada. |

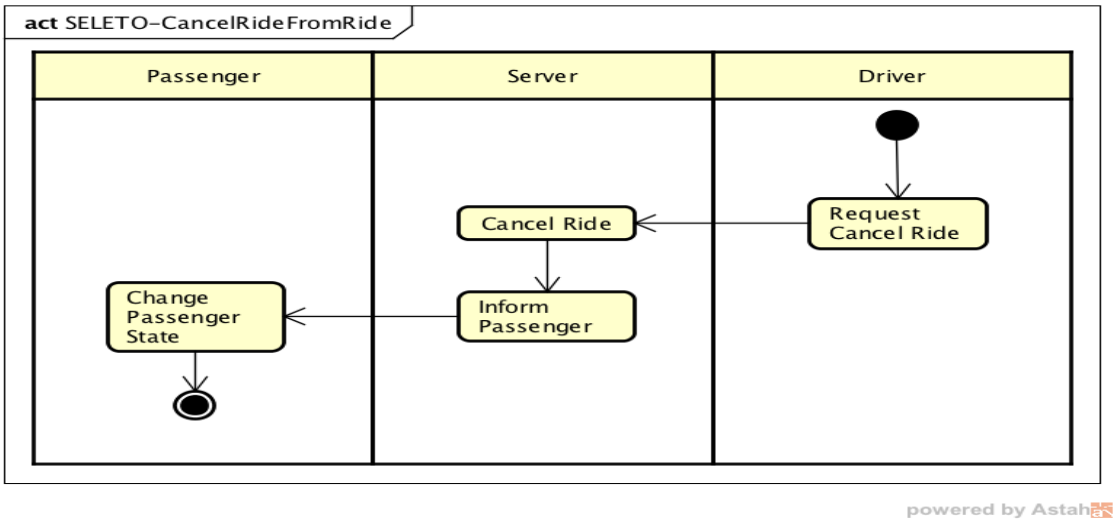
Fonte: Autor

Figura 4.10: *Cancel Ride From Passenger* - Diagrama de atividades




Fonte: Autor

Figura 4.11: *Cancel Ride From Driver* - Diagrama de atividades




Fonte: Autor

### 4.4.4 Caso de Uso - *Start Ride*

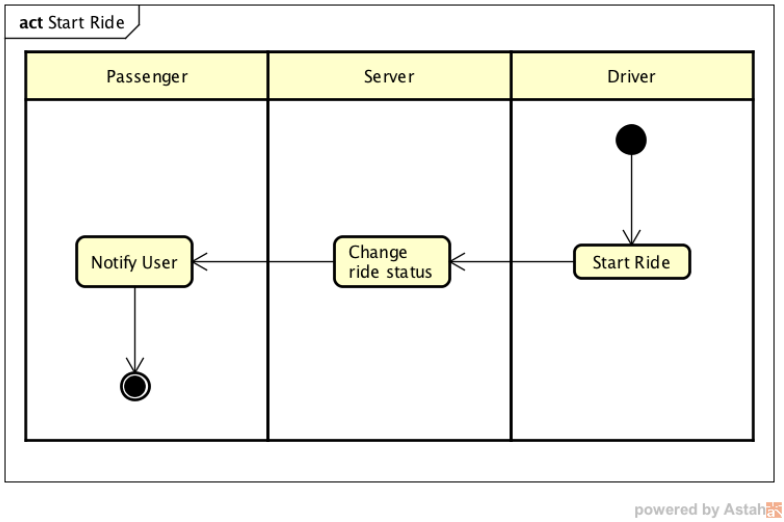
Uma vez que o motorista chega na posição em que se encontra o passageiro, o aplicativo deve informar ao motorista que pode começar a corrida, assim que o passageiro o encontrar. O aplicativo do motorista deve, após inicializar a corrida, informar ao servidor que ela está em andamento, consequentemente, o servidor informa ao passageiro que a corrida começou.

Tabela 4.4: Caso de Uso - *Start Ride*

| *Start Ride* | |
|---|---|
| Pré-condições | • Passageiro deve ter requisitado a corrida.<br>• Motorista deve ter aceitado a corrida. |
| Atores | *Driver, Passenger, Server* |
| Pós-condições | • Motorista deve iniciar a corrida. |
| Fluxo Principal | • O aplicativo deve informar ao motorista que se encontra próximo ao passageiro.<br>• O motorista deve inicializar a corrida.<br>• O aplicativo deve informar ao servidor que a corrida foi iniciada.<br>• O servidor deve informar ao passageiro que a corrida foi iniciada. |

Fonte: Autor

Figura 4.12: *Start Ride* - Diagrama de atividades




Fonte: Autor

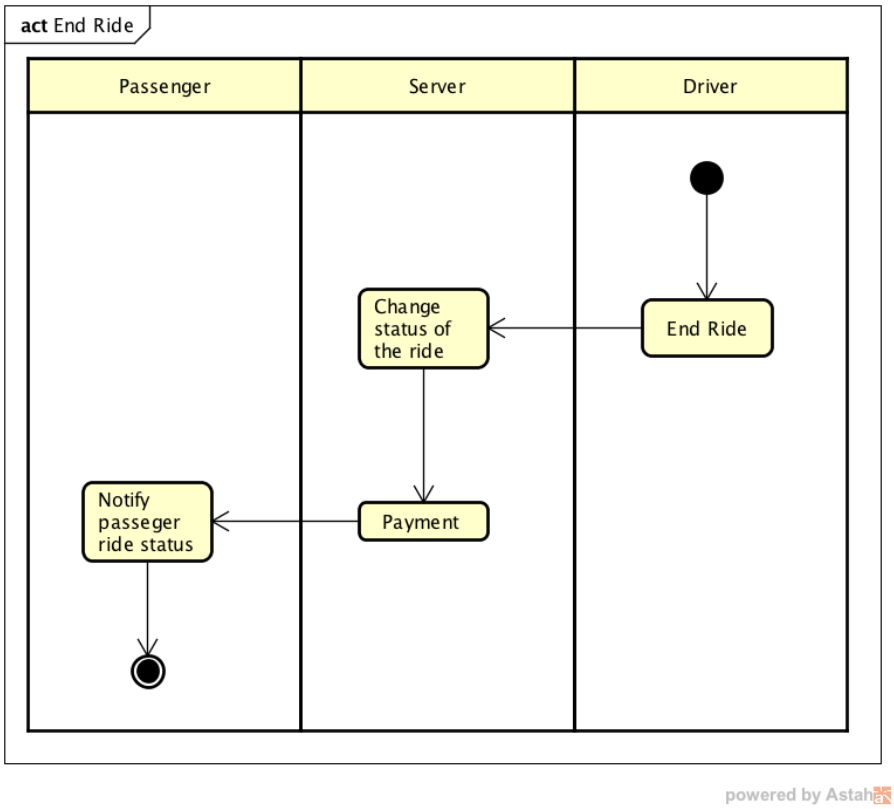
### 4.4.5 Caso de Uso - *End Ride*

O motorista, ao atingir o destino do passageiro, deve finalizar a corrida. Uma vez que o motorista performe esta ação, o aplicativo deve informar ao servidor que o status da corrida foi finalizado e, consequentemente, notificar o passageiro sobre o fim da corrida e confirmar o valor cobrado por ela.

Tabela 4.5: Caso de Uso - *End Ride*

| *End Ride* | |
|---|---|
| Pré-condições | • Motorista deve já ter inicializado a corrida. |
| Atores | *Driver, Passenger, Server* |
| Pós-condições | • Corrida deve ser finalizada.<br>• Motorista deve ter status livre para aceitar novas corridas.<br>• Passageiro deve ter status livre para requisitar novas corridas. |
| Fluxo Principal | • O aplicativo deve informar ao motorista que se encontra próximo ao destino do passageiro.<br>• O motorista deve encerrar a corrida.<br>• O aplicativo deve informar ao servidor que a corrida foi finalizada.<br>• O servidor deve gerar o pagamento.<br>• O servidor deve notificar que a corrida foi finalizada. |

Figura 4.13: End Ride - Diagrama de atividades




Fonte: Autor

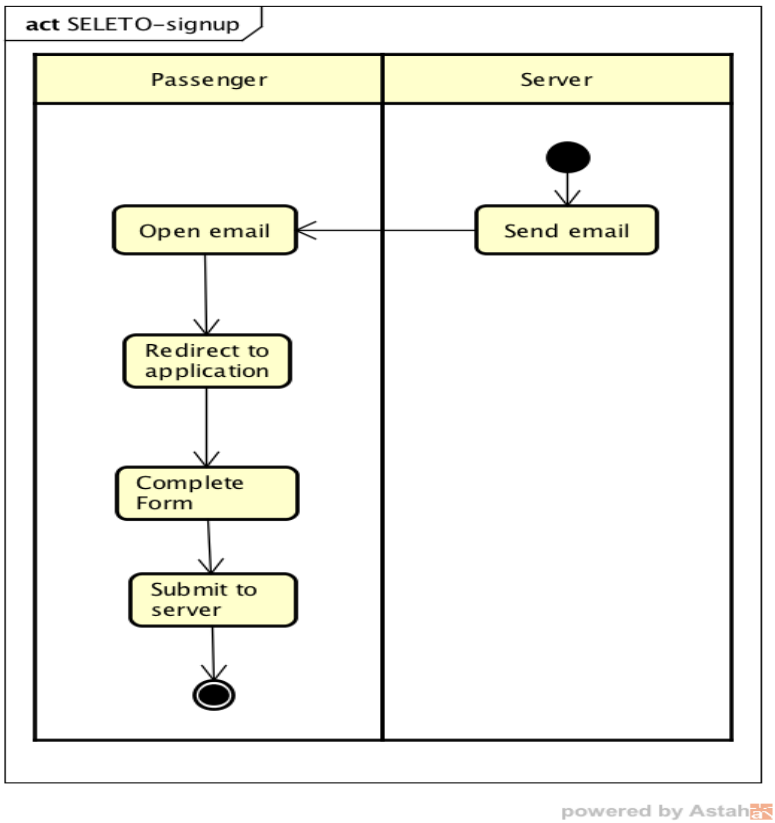
### 4.4.6 Caso de Uso - *Signup*

O caso de uso *Signup* ocorre quando o passageiro se cadastra na plataforma. O passageiro consegue se cadastrar na plataforma através de um convite que o sistema enviará para o seu email. Quando o passageiro abre o *link* no email, ele será redirecionado para o aplicativo e poderá proceder com o preenchimento de suas credenciais, enviando suas informações ao servidor. Uma vez que ele envie seus dados ao servidor, será feita a validação da informação do usuário. O foco durante o desenvolvimento foi obter os dados de maneira fidedigna do usuário. A tarefa de avaliação dos dados do usuário contidos na base de dados é um trabalho futuro no desenvolvimento do aplicativo.

Tabela 4.6: Caso de Uso - *Signup*

| *Signup* | |
|---|---|
| Pré-condições | • Servidor envia email de criação de conta para o passageiro. |
| Atores | *Passenger, Server* |
| Pós-condições | • Passageiro está cadastrado no sistema. |
| Fluxo Principal | • Passageiro abre email.<br>• Passageiro abre link do email e é redirecionado.<br>• Passageiro preenche formulário com suas informações.<br>• Passageiro submete informações ao servidor. |

Figura 4.14: *Signup* - Diagrama de atividades




Fonte: Autor

## 4.5 Arquitetura

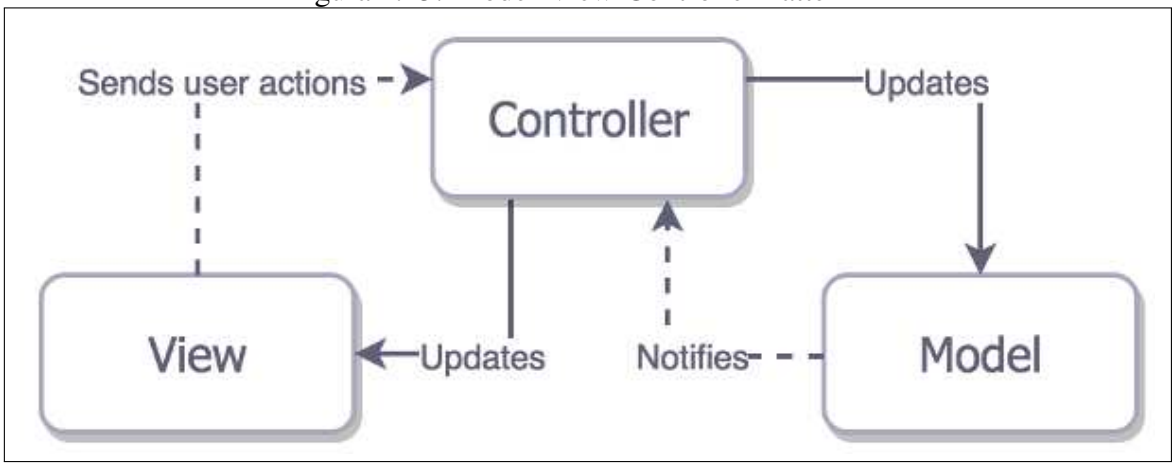
A arquitetura em que reside as aplicações iOS e Apple em geral é o MVC (*Model-View-Controller*). Esse padrão arquitetural foi criado nos meados da década de 70 e contêm três camadas com responsabilidades distintas:

**Model:** consiste nos dados da aplicação, regras de negócios, lógica e funções.

**View:** pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama.

**Controller:** faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão.

Figura 4.15: Model-View-Controller Pattern

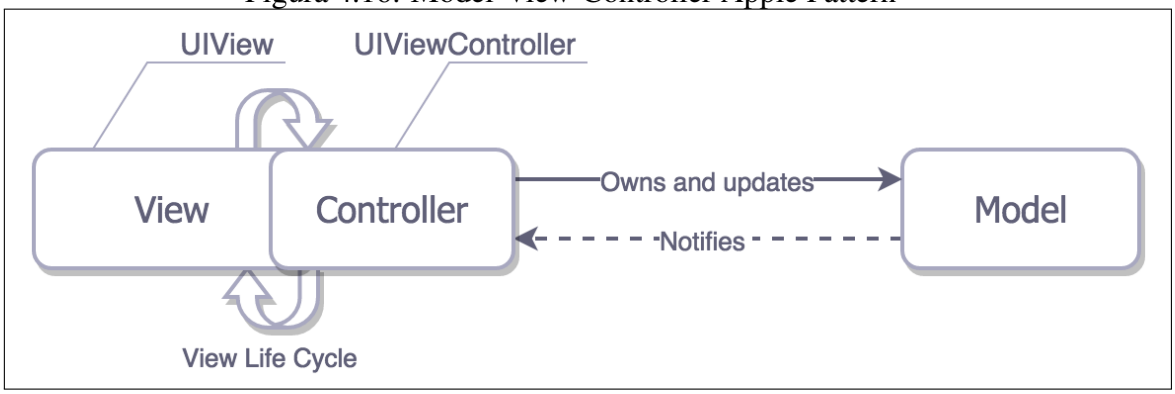



Fonte: <https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52>

Apesar de ser um padrão arquitetural conhecido e a Apple encorajar a sua adoção, o *framework* de desenvolvimento utilizado não funciona seguindo as regras do tradicional MVC. Na figura seguinte, pode ser analisada a arquitetura como realmente é aplicada por aplicações iOS. O *controller* é altamente acoplado à *View* e grande parte das responsabilidades da *View* são delegadas ao *controller*. Logo, o *controller* acaba por conter toda lógica e responsabilidades, assim o *controller* acaba por se tornar uma classe extensa e de difícil testabilidade.

Afim de contornar essa característica provida pelo *framework* da Apple, foi decidido utilizar outra arquitetura chamada Model-View View-Model (MVVM), explicado em (SORENSEN; MIHAILESC, 2010). Uma vez que a *view* e o *controller* são altamente

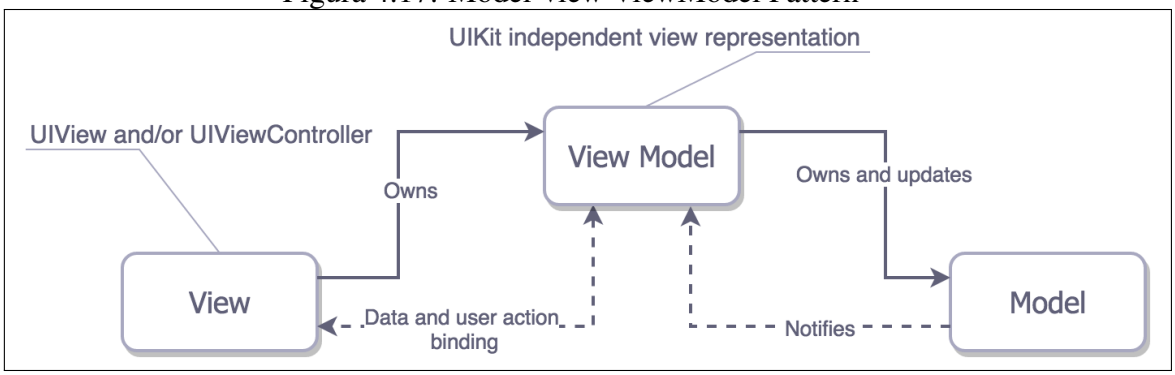
Figura 4.16: Model-View-Controller Apple Pattern




Fonte: <https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52>

acoplados no MVC da Apple, no MVVM, eles são considerados como a *view* propriamente dita. O *ViewModel* é o mediador entre a *view* e o *model*, o qual é responsável por receber ações e guardar estados da *view*. O *ViewModel*, por sua vez, também invoca ações na camada de modelo. Essas ações atualizam a camada de modelo e este atualizará o *ViewModel*. Entre o *ViewModel* e a *View* existe um *binding* (ligação), que atualiza a *view* imediatamente quando um dado é alterado no *ViewModel*. Um benefício é o isolamento dos elementos da UI, facilitando testes.

Figura 4.17: Model-view ViewModel Pattern




Fonte: <https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52>

### 4.5.1 Binding

Previamente, vimos que a atualização da *View* acontece pelo *binding* entre a *View* e o *ViewModel*. Uma vez que os dados são atualizados no modelo e passados para o *ViewModel*, esse, por sua vez, atualiza imediatamente a *View*. Para sinalizarmos para a *View* que mudanças ocorreram no *ViewModel*, como, por exemplo, a mudança de um valor em

uma variável, precisamos registrar esses eventos na *View*. Logo, a *view* sempre receberá os valores atualizados do *ViewModel*. A fim de atingir esse objetivo, utilizamos o design de projeto comportamental *Observer*[2], descrito no livro (GAMMA et al., 1994), em que a *view* se registra perante o *ViewModel* a fim de receber as notificações quando alguma mudança ocorrer. Portanto, no projeto, criamos uma classe responsável pelo *binding* entre a *View* e o *ViewModel*. Qualquer modificação no valor da variável, a classe notificará essa mudança.

Figura 4.18: Dynamic class implementa o Binding

```
class Dynamic<T> {
    typealias Listener = (T) -> Void
    var listener: Listener?

    func bind(_ listener: Listener?) {
        self.listener = listener
    }

    func bindAndFire(_ listener: Listener?) {
        self.listener = listener
        listener?(value)
    }

    var value: T {
        didSet {
            listener?(value)
        }
    }

    init(_ val: T) {
        value = val
    }
}

```

[2]|Padrão de projeto de software que define uma dependência entre objetos de modo que quando um objeto muda o estado, todos seus dependentes são notificados e atualizados automaticamente.

O *ViewModel*, neste caso PassengerProfileViewControllerModel, implementa o *binding* perante uma variável que, uma vez modificada, notificará quem se registrou.

Figura 4.19: Exemplo de View Model

```
protocol PassengerProfileViewModel {

    var user: Dynamic<User?> { get }
    var photoData: Dynamic<Data?> { get }

    func fetchUserImage()

}

class PassengerProfileViewControllerModel: PassengerProfileViewModel {

    let user: Dynamic<User?>
    var photoData: Dynamic<Data?>

    init() {
        if let currentUser = try? User.getCurrentUser()?.fetchIfNeeded() {
            user = Dynamic(currentUser)
        } else {
            user = Dynamic(User.getCurrentUser())
        }
        photoData = Dynamic(nil)
    }

    func fetchUserImage() {
        user.value?.profilePicture.getDataInBackground(block: { [unowned self] (data, error) in
            self.photoData.value = data
        })
    }
}

```

A *View* se registra perante a *ViewModel* para receber notificações quando alguma mudança ocorreu. Uma vez que ocorra uma mudança, as alterações necessárias para atualizar a *View* estarão no retorno de uma função anônima.

Figura 4.20: Exemplo de View

```
class PassengerProfileViewController: UIViewController {

    var passengerProfileViewModel:
  PassengerProfileViewModel?

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        passengerProfileViewModel =
  PassengerProfileViewControllerModel()
        passengerProfileViewModel?.photoData.bindAndFire({
  [unowned self] (data) in
            if let data = data {
                self.profileImageView.image = UIImage(data:
   data)
            } else {
                self.profileImageView.image = nil
            }
        })
    }
}

```

## 4.6 Servidor

O Parse Server é um BaaS construído em Node e MongoDB que utiliza uma API Restfull para consumir e prover dados. Essa API Restfull funciona sobre o protocolo HTTP e os dados trafegados, em sua maioria, são JSON.

BaaS também são conhecidas como *Serveless architectural*. Segundo Fernandez (2016), o termo significava que a aplicação seria totalmente dependente de aplicações terceiras, e seus serviços seriam utilizados para gerenciar lógicas de servidor e seu estado. Em aplicações BaaS, podemos descrever lógica no servidor, isto é, podemos invocar funções perante ao servidor para desempenhar ações, assim como um serviço qualquer na nuvem.

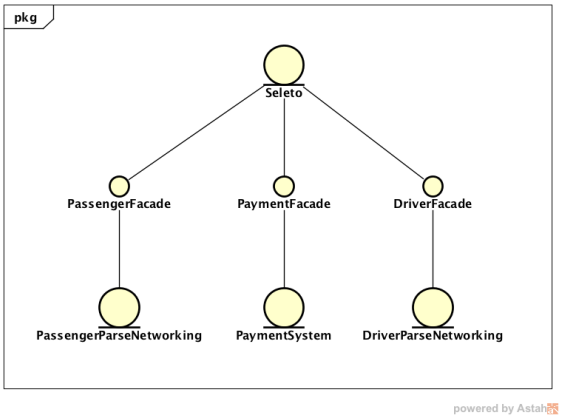
O Parse Server permite a criação de funções através de um serviço chamado CloudCode, o qual permite a criação de lógica no servidor, criando assim, um único ponto em que os clientes podem se comunicar ou invocar ações perante o servidor. Além de trabalhar com os dados já salvos no banco de dados, é possível utilizá-los e criar lógicas baseadas nas invocações dos usuários do sistema.

Figura 4.21: Cloud code - Passenger criando uma corrida

```
var passenger = require("../domain/passenger.js");

Parse.Cloud.define("createRide", function(request, response) {
    var date = new Date();
    if (!_.isUndefined(request.params.date) ) {
        date = new Date(request.params.date);
    }

    passenger.callRide(request.user.id, request.params.source,
                       request.params.destination, date, request.params.price, request.params.driverId ).then(function(results) {
        response.success(results);
    }, function(error) {
        response.error(error);
    });
});

```

No aplicativo, as conexões com o servidor e outros serviços, como o pagamento, são isoladas por *façade*[3], padrão arquitetural encontrado no livro (GAMMA et al., 1994). Caso necessitemos alterar o servidor por outro BaaS, ou até mesmo construir uma nova solução para o servidor, seria somente necessário alterar as classes abaixo da *façade*.

Figura 4.22: Networking Facade



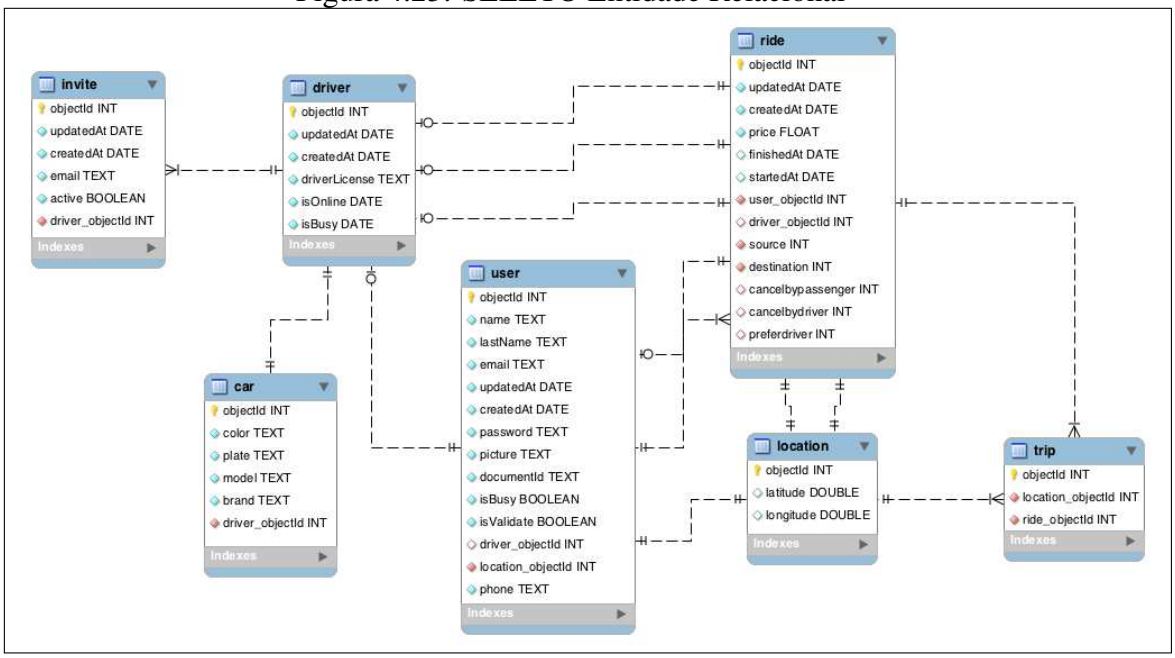

Fonte: Autor

[3]Um Façade é um objeto que provê uma interface simplificada para um corpo de código maior, como por exemplo, uma biblioteca de classes.

### 4.6.1 Banco de Dados

Com a definição dos casos de uso e as funcionalidades do sistema, foi possível construir uma modelagem inicial para o banco de dados. Tradicionalmente, o diagrama entidade relacional de um banco SQL para o projeto seria como o descrito na Figura 4.23.

Figura 4.23: SELETO Entidade Relacional

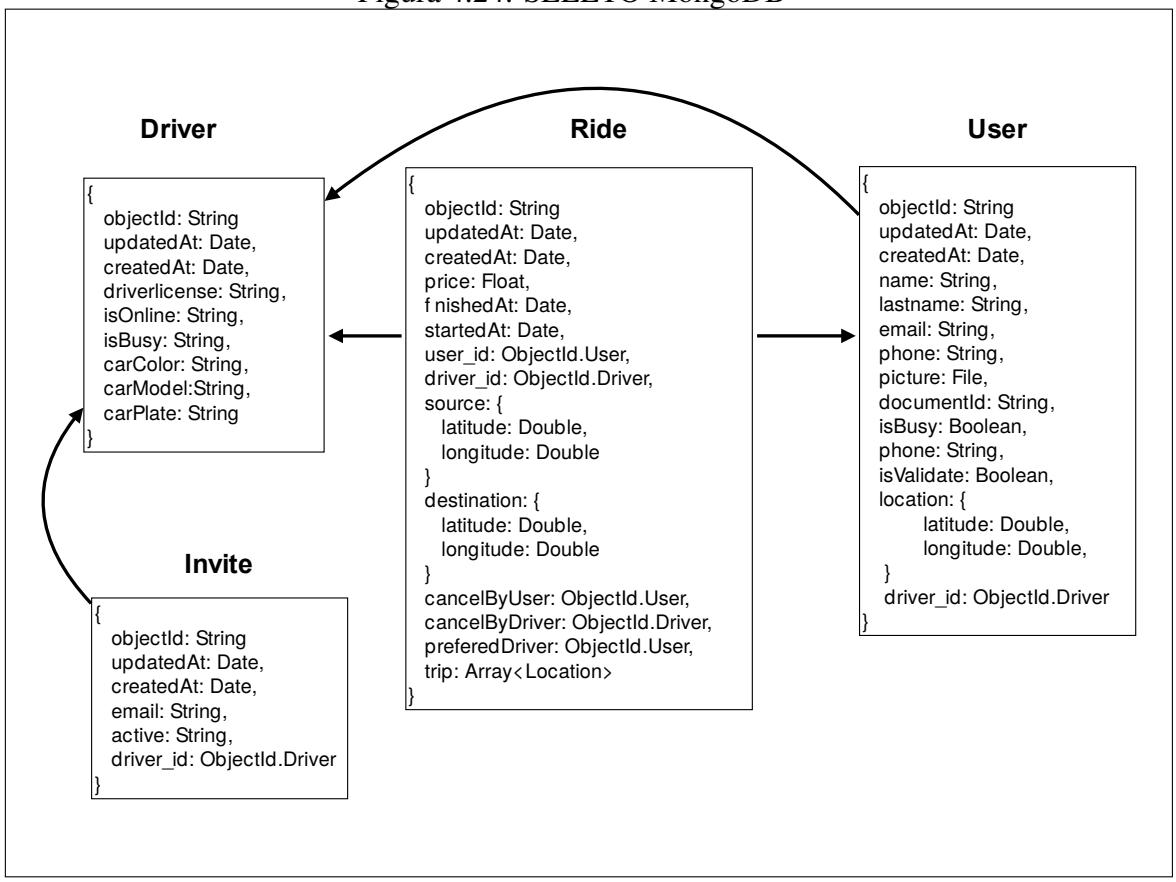



Fonte: Autor

Todavia, devido à utilização da tecnologia MongoDB, que possui um esquema flexível, no futuro é possível que apresente modificações para adaptar as novas funcionalidades do sistema. Esta seção aborda as coleções criadas para o desenvolvimento da aplicação e, também, as coleções existentes para o controle e segurança do sistema.

Como o MongoDB guarda documentos do tipo JSON, as tabelas anteriores foram desnormalizadas, a fim de conseguir todos os dados necessários em um único *request*. Por exemplo, a tabela Localização foi adiciona a todas as classes que as utilizavam, deste modo evitando joins ou outras consultas. Estas regras são recomendadas na documentação do (MONGODBDOCS, 2017).

Figura 4.24: SELETO MongoDB




Fonte: Autor

### 4.6.2 *User*

Esta coleção é utilizada para armazenar as informações dos usuários. Dessa forma, possui as propriedades necessárias para armazenar as informações do nome, e-mail, senha, foto, número de documento e localização - latitude e longitude. Caso o usuário seja um motorista, existe um ponteiro para a coleção *Driver* em que o adicionamos. Essas informações são preenchidas no momento do cadastro de um novo usuário e, também, na tela de Perfil do aplicativo. Essa coleção é muito importante porque está relacionada com várias outras existentes no sistema.

### 4.6.3 *Driver*

Esta coleção é utilizada para armazenar as informações dos motoristas. Um usuário pode ser um motorista, desde que exista um registro na coleção usuário que remeta a coleção *Driver*. A coleção *Driver* contém informações do carro do motorista como cor,

modelo e placa. O motorista precisa informar para o sistema se está online, pois somente assim o servidor poderá informá-lo que existe uma corrida disponível.

### 4.6.4 *Ride*

Esta coleção é utilizada para armazenar as informações quando uma corrida é inicializada por um usuário. Portanto, todos os estados durante a corrida são armazenados neste documento. Durante a criação da corrida, informamos ao usuário que a criou, o preço da corrida e o motorista que aceitou a corrida. Armazenamos também quando começa a corrida e quando ela é encerrada. Além disso, armazenamos a localização de onde o passageiro se encontra, assim como o ponto de destino da corrida. Nesta coleção foi adicionado também um *array* de localizações quando um motorista inicia uma corrida, ou seja, o aplicativo irá enviar ao servidor, de tempos em tempos, as localizações que o motorista percorreu até completar a corrida.

### 4.6.5 *Invite*

Esta coleção é utilizada para armazenar as informações quando são convidados novos passageiros para o sistema. Quando é adicionado um novo invite, com o email do novo passageiro, o servidor envia um email com um código e um link ao passageiro, o que permitirá a ele fazer o *signup* no sistema. Uma vez que o passageiro faz *signup* no sistema, a variável *active* informa que o código já foi utilizado.

### 4.6.6 Mapeamento *Backend* e *Frontend*

A fim de demostrar o mapeamento entre o *backend e o frontend*, o parse framework facilita o mapeamento direto entre os dados do Parse Server e do aplicativo, neste caso, a aplicativo móvel. Logo após o login, por exemplo, a classe descrita na Figura 4.25 gera um usuário, como definido no *backend*.

Figura 4.25: User

```
import Foundation
import Parse

class User: PFUser {

    @NSManaged var name: String
    @NSManaged var lastName: String
    @NSManaged var phone: String
    @NSManaged var picture: PFFile
    @NSManaged var documentId: String
    @NSManaged var driver: Driver?
    @NSManaged var location: PFGeoPoint?
    @NSManaged var busy: Bool

}

```

### 4.6.7 Aplicações em Tempo Real

Em aplicações convencionais, temos a tradicional arquitetura cliente servidor em que o servidor é passivo e somente responde após alguma requisição do cliente. Em aplicações de tempo real, a cada modificação de estado é preciso notificar o cliente sobre esta mudança. Aplicações deste tipo são bem comuns, como games, chats, colaborações de código, entre outras. Existem várias técnicas e protocolos para proporcionar aplicações em tempo real como *pooling*, *long-polling* e *comet*, todavia um protocolo que provê baixa latência e baixo consumo de tráfego na rede é o WebSocket, definido na RFC (WEBSOCKETRFC, 2011), o qual pode ser analisado em (LIU; SUN, 2012).

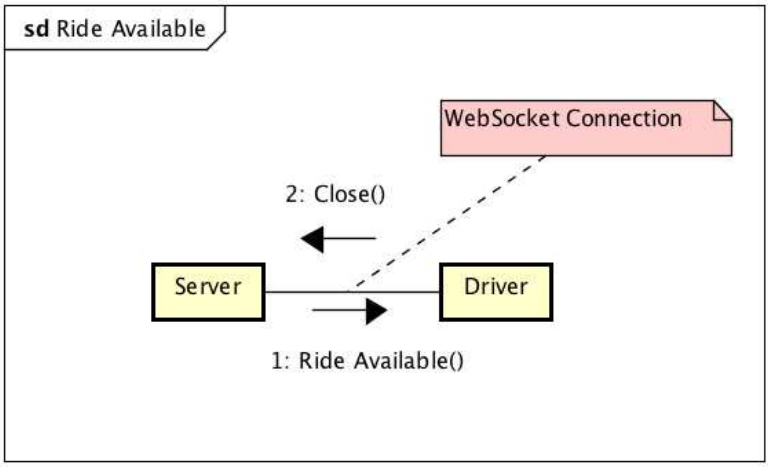
O projeto Seleto tem necessidade do servidor ser reativo, por exemplo, quando um passageiro requisita uma corrida, é necessário notificar todos os motoristas sobre esta corrida. Um outro exemplo de aplicação em tempo real no projeto Seleto, acontece quando o motorista aceita a corrida. O motorista envia sua localização para o servidor, o qual notifica o passageiro, informando-o sobre a localização do motorista.

O protocolo WebSocket funciona abrindo uma conexão entre o servidor e o cliente e/ou servidor e motorista. Nesta conexão é possível a comunicação bilateral entre as entidades envolvidas. Deste modo, é possível mapear as conexões necessárias para que o aplicativo desempenhe suas funcionalidades.

Na Figura 4.26, o motorista sempre abrirá uma conexão por websocket com o servidor para receber notificações quando uma corrida estiver disponível. Nesta mensagem,

enviada pelo servidor, está contido um objeto *Ride*. Uma vez que esta mensagem chegue ao motorista, ele tem duas opções: aceitar ou rejeitar a corrida. Caso venha a aceitar, o aplicativo enviará ao servidor a informação de que a aceita ou a rejeita.

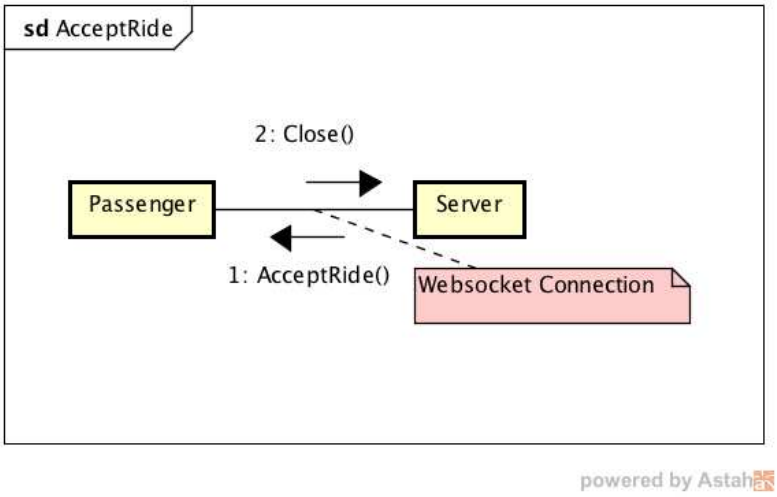
Figura 4.26: *Available Rides - Websocket Connection*




Fonte: Autor

O passageiro abre uma conexão com o servidor logo que requisita uma corrida. Quando uma corrida é aceita por algum motorista, o servidor informa imediatamente o *status* da corrida, a qual agora contém um motorista alocado para atendê-la.

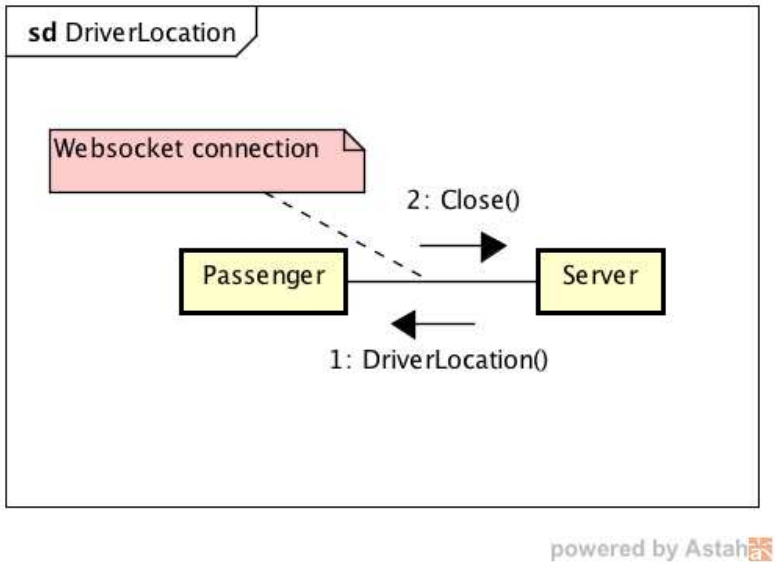
Figura 4.27: *Accept Ride Websocket Connection*




Fonte: Autor

O passageiro tem de abrir uma conexão com o servidor para saber a localização que o motorista se encontra, à medida que se dirige para encontrar o passageiro.

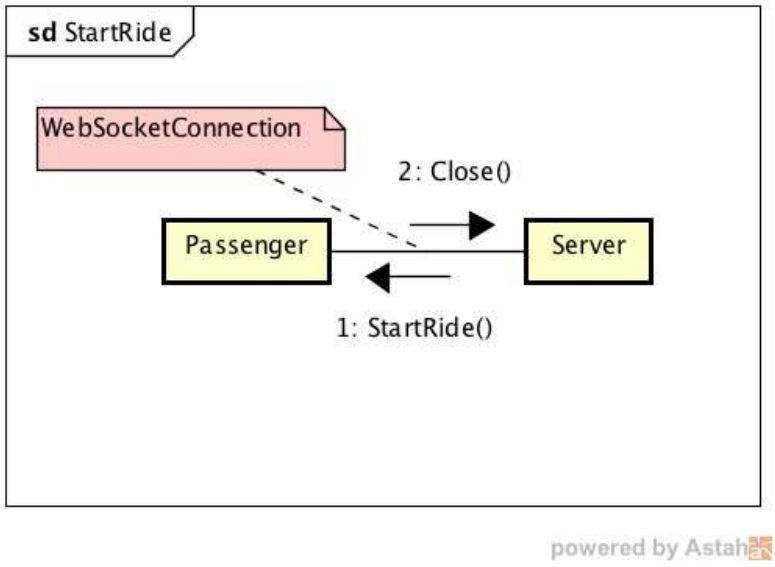
Figura 4.28: *Driver Location Websocket Connection*




Fonte: Autor

O passageiro necessita ser informado quando o motorista inicia a corrida, logo o servidor precisa notificar o passageiro quando esta ação ocorre.

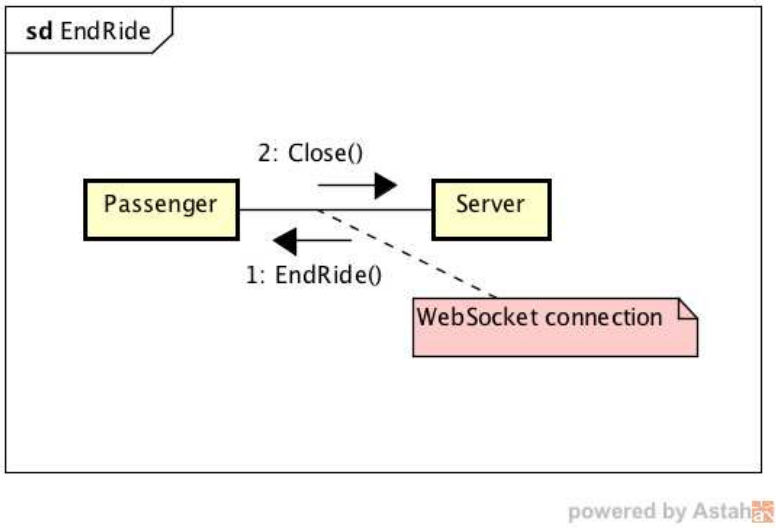
Figura 4.29: *Start Ride Websocket Connection*




Fonte: Autor

Do mesmo modo que o passageiro necessita saber que a corrida foi iniciada pelo motorista, o passageiro precisa saber que o motorista finalizou a corrida.

Figura 4.30: *End Ride Websocket Connection*




Fonte: Autor

## 4.7 *Business Model*

Um *business model canvas* é composto de nove blocos, os quais são: *Customer Segments*, *Value proposition*, *Channel*, *Customer Relationships*, *Revenue Streams*, *Key Resources* e *Key Activities*, *Key Partners* e *Cost structure*. Com essas nove categorias, é possível mapear, discutir e analisar o negócio, como apresentado no livro de (OSTERWALDER; PIGNEUR; CLARK, 2010)

*Custumer Relationships* (Segmento de clientes): agrupam todos os tipos de clientes, desde os mais simples até organizações mais complexas. No caso do aplicativo Seleto, foram mapeados passageiros que necessitam se locomover, não possuem um carro e que gostariam de um tratamento diferenciado. Além disso, o aplicativo possui outros clientes, os motoristas, que seriam considerados parceiros da plataforma.

*Value Preposition* (Proposta de valor): agrupam todos os valores e serviços que o produto entregará ao cliente. Na solução proposta foram analisados diferenciais que outras plataformas não oferecem ou não executam de forma excelente. Entre estes valores temos a segurança, ou seja, tanto o motorista quanto o passageiro devem possuir uma análise de perfil. Os motoristas terão um treinamento para prover um melhor atendimento aos passageiros. A fim de tornar mais atrativos para os motoristas a diferença entre o valor da corrida, o repasse sobre o valor cobrado na corrida será maior.

*Channel* (Canais): é como nós entregamos nosso valor ao nosso cliente. Neste caso, por se tratar de um aplicativo mobile, utilizaremos a *AppStore* para distribuir nossa solução aos nossos clientes.

*Customer Relationship* (Relacionamento com os clientes): é como nós nos relacionamos com o nosso cliente. A fim de manter contato, tanto com os motoristas quanto com os passageiros, utilizaremos de redes sociais, e um sistema de *feedback*, através do qual eles poderão sugerir melhorias para a plataforma.

*Revenue Streams* (Métodos de Receita): é como o serviço gera receita a partir de cada cliente. Neste caso, sempre que o usuário aceita uma corrida, é possível gerar receita.

*Key Resources* (Recursos Principais): nesta seção, adicionamos quais são os recursos necessários para entregar a proposta. Neste momento, precisamos somente de recursos técnicos para gerar o aplicativo.

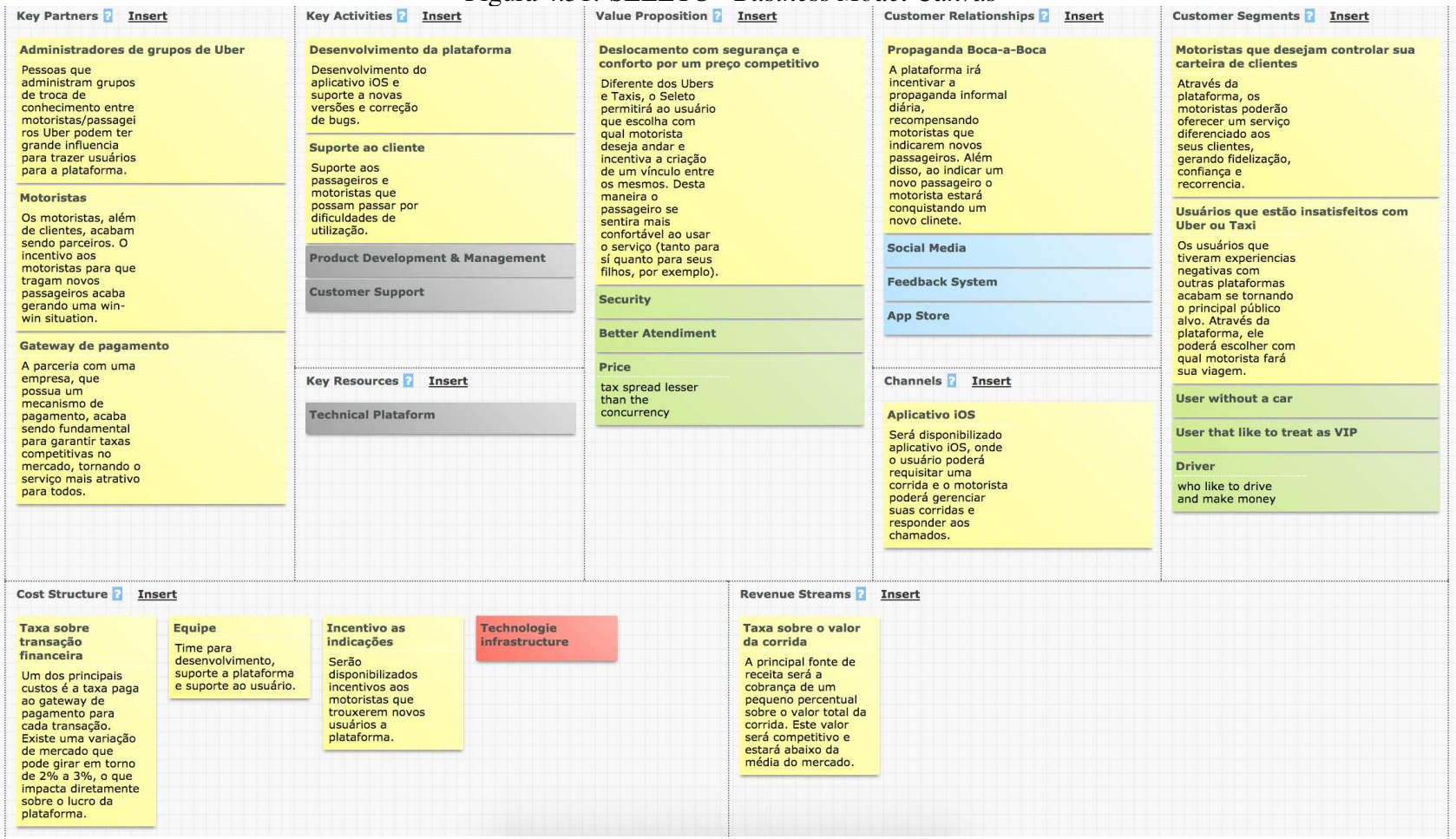
*Key Activities* (Atividades Chaves): são as atividades principais para entrega do produto. A fim de prover a entrega, é necessário desenvolvermos o produto e o suporte ao cliente quando o produto for lançado.

*Key Parteners* (Parceiros Principais): depois que as atividades principais forem completadas, é necessário ter os parceiros (motoristas) para performar bem o aplicativo. Neste caso, além dos motoristas, é necessário um sistema de pagamento para ajudar na cobrança do passageiro e pagamento do motorista.

*Cost Structure* (Estruturas de Custo): nesta seção os custos do sistema são apresentados. Atualmente, temos todo o custo do servidor e de desenvolvimento.

O canvas do Modelo de Negócio é uma ferramenta que ajuda a descrever, desenhar, desafiar, inventar e pivotar o modelo de negócio do produto. O canvas permite sintetizar o modelo de negócio de forma visual e de fácil compreensão.

Figura 4.31: SELETO - *Business Model Canvas*




Fonte: Autor

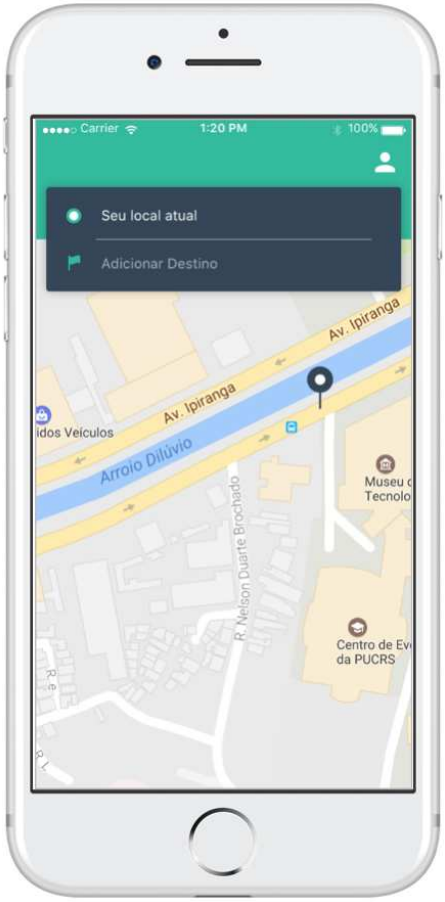
## 4.8 Telas do Aplicativo

Nesta seção serão mostradas as principais telas do aplicativo desenvolvido, assim como seu comportamento baseado nos casos de uso citados anteriormente.

### 4.8.1 Tela inicial

A tela inicial do aplicativo contém a posição atual do usuário e dois campos, os quais serão preenchidos pelo usuário para informar sua localização atual, em que ele deseja ser encontrado, e seu destino.

Figura 4.32: Tela inicial

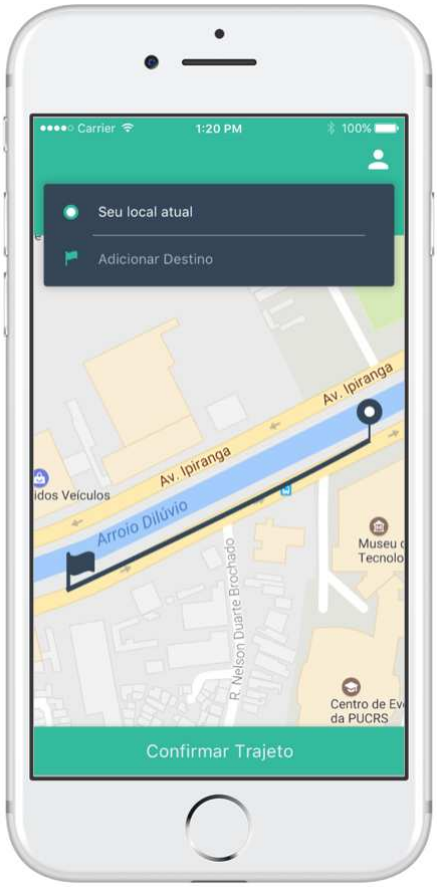



Fonte: Autor

### 4.8.2 Seleção de locais origem e destino

Uma vez que o passageiro tenha preenchido os campos de localização atual e destino. O aplicativo mostrará uma linha sobre o mapa indicando a posição atual e o destino nas extremidades, assim como o trajeto ideal entre a origem e o destino.

Figura 4.33: Seleção de locais de origem e destino

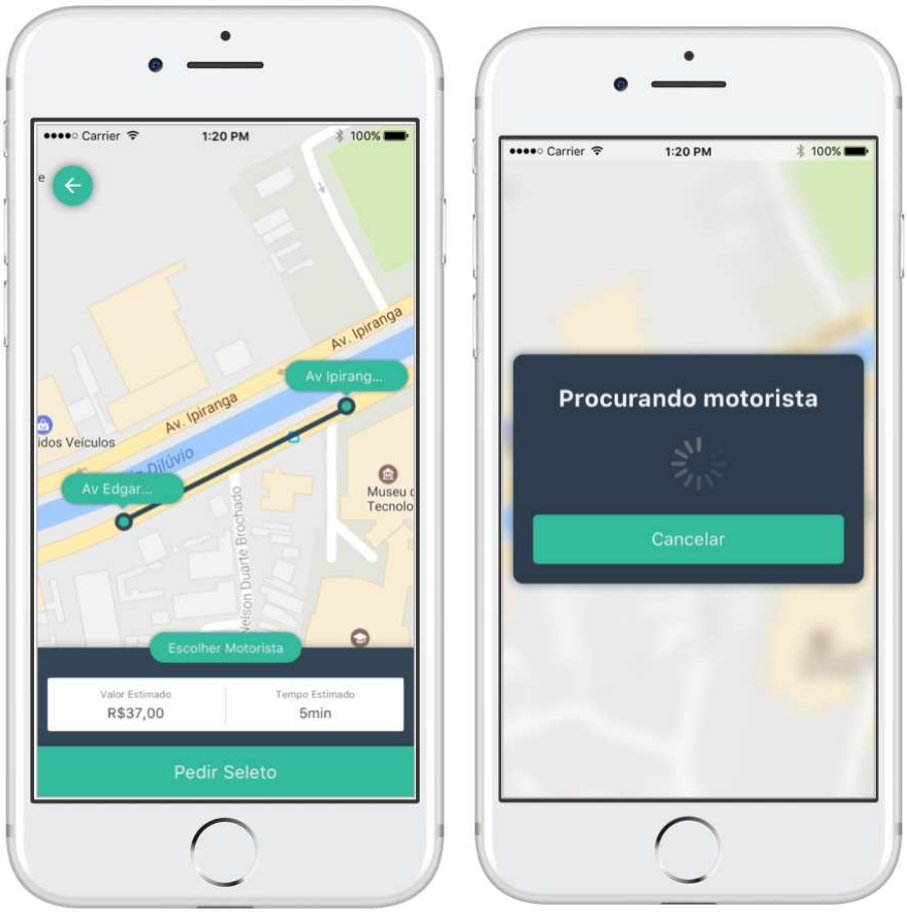



Fonte: Autor

### 4.8.3 Informações da Corrida

O passageiro, ao continuar o fluxo, após informar sua localização atual e destino. O aplicativo informará todas as informações antes de continuar o processo de chamar o motorista. Nesta tela o passageiro ainda pode escolher um motorista de sua preferência, caso este esteja disponível para atendê-lo, este terá preferência para fazê-lo. O usuário, após fiscalizar os dados, confirmará e o aplicativo informará que está a procura de motoristas para atendê-lo.

Figura 4.34: Dados da corrida

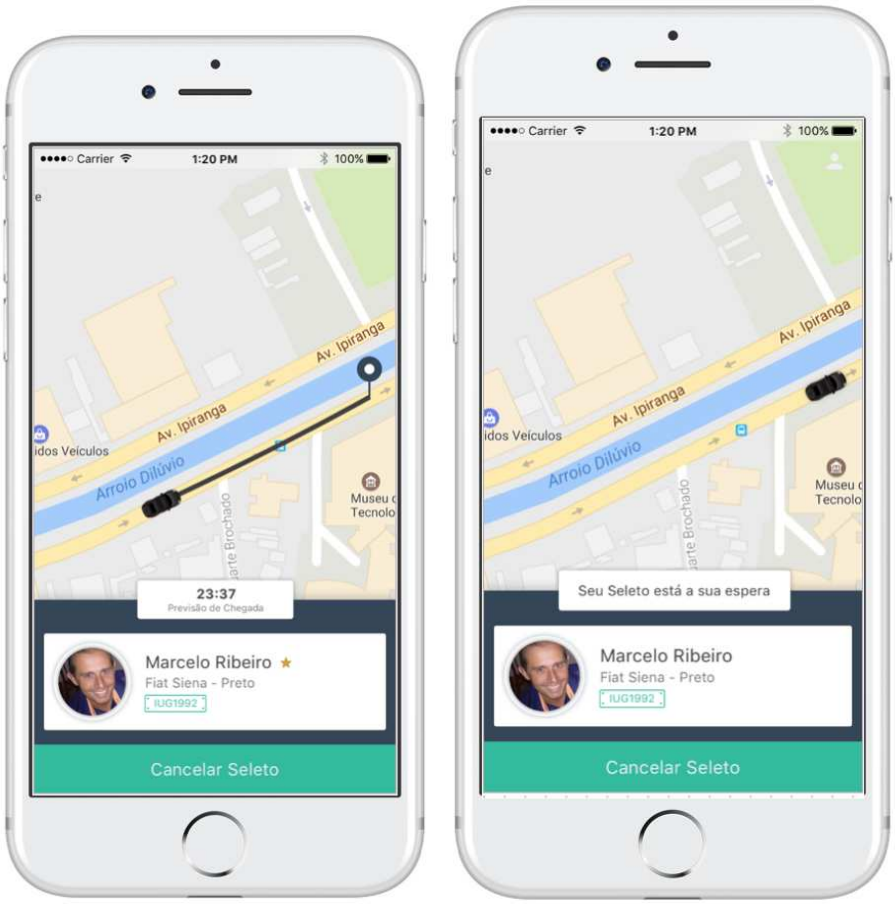



Fonte: Autor

### 4.8.4 Corrida Aceita

O sistema, quando encontra um motorista que quer atender o passageiro, notifica o aplicativo do passageiro, o qual muda a interface do passageiro indicando onde o motorista se encontra e quanto tempo levará para chegar ao encontro do passageiro. Uma vez que o motorista chegue ao ponto de encontro. O aplicativo do passageiro informará que o motorista está a sua espera.

Figura 4.35: Motorista a caminho e motorista




Fonte: Autor

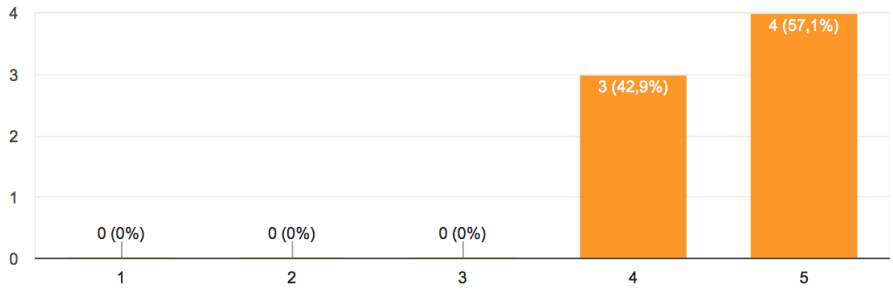
### 4.8.5 Análise do Sistema

Uma pesquisa, no *google forms*, foi criada para validar as funcionalidades desenvolvidas até o momento no aplicativo Seleto. Nesta pesquisa foi desenvolvido um questionário que pode ser encontrado no apêndice 5.3. As funcionalidades foram testadas com 7 usuários dos quais 57,1% são homens e 42,9% são mulheres e entre eles 85,8% estão cursando ou já acabaram o curso superior. As funções testadas pelos usuários, até o

momento, foram o *Signup* e *Request Ride*. Os usuários eram instruídos a criar uma conta, executando assim o caso de uso *Signup*, em seguida, eles logavam no sistema e criavam uma corrida, executando o caso de uso *Request Ride*.

Quando perguntados sobre *Foi fácil adicionar suas informações para se cadastrar no aplicativo?* 57,1% avaliaram com nota 5, ou seja, acharam muito fácil adicionar suas informação, enquanto 42,9% avaliaram com nota 4 durante o processo de *Signup*

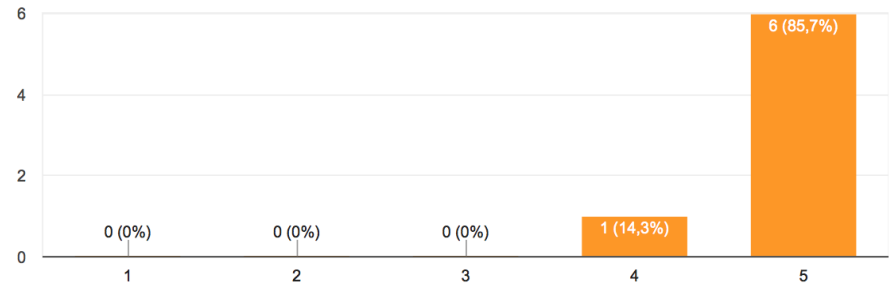
Figura 4.36: Pesquisa - Adicionar informações durante cadastro




Fonte: Autor

Quando perguntados sobre *Foi fácil incluir os endereços do percurso desejado no aplicativo?* 85,7% avaliaram 5, ou seja, acharam muito fácil adicionar suas informação, enquanto 14,3% avaliaram com nota 4 durante o processo de incluir origem e destino da viagem.

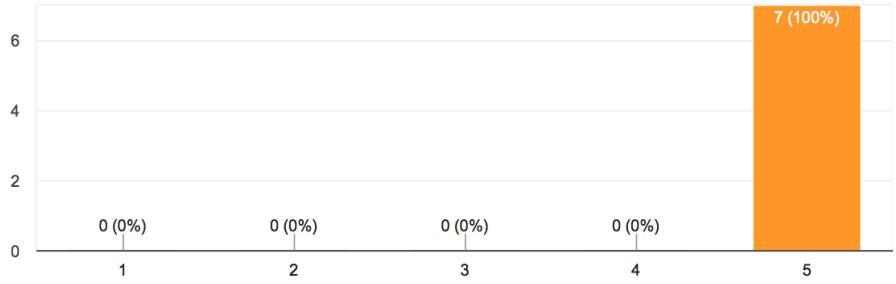
Figura 4.37: Pesquisa - Adicionar incluir endereço




Fonte: Autor

Quando perguntados sobre *Foi fácil selecionar um motorista no aplicativo?* todos os usuários avaliaram com 5.

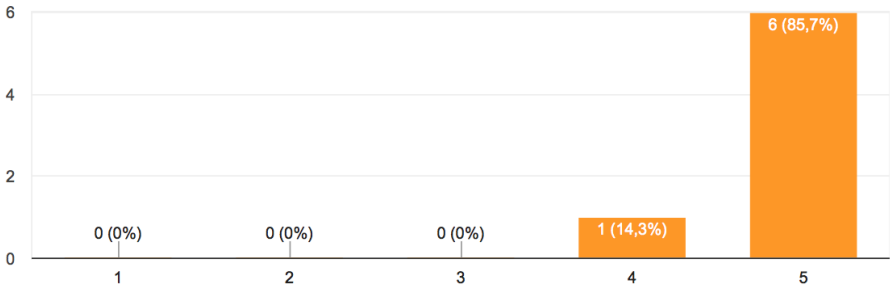
Figura 4.38: Pesquisa - Selecionar Motorista




Fonte: Autor

Por fim, a última pergunta do questionário *Foi fácil chamar uma corrida no aplicativo?* 85,7% avaliaram com nota 5, enquanto 14,3% avaliaram com nota 4.

Figura 4.39: Pesquisa - Chamar corrida




Fonte: Autor

Os usuários ainda puderam dar sugestões e *feedbacks* sobre melhorias ou novas funcionalidades. Abaixo seguem as sugestões propostas dos usuários:

- Dar a opção de editar o endereço de origem (número da residência) sem apagar a informação que o aplicativo inseriu automaticamente ao reconhecer a rua onde eu estava. Pois ele deletou o nome da rua e tive que digitar tudo novamente.

- Acredito que o aplicativo atende bem a necessidade e um próximo passo seria salvar os endereços favoritos para tornar ainda mais simples a funcionalidade de chamar um carro.

Adicionar as principais localidades em que o passageiro tem como destino diariamente é uma funcionalidade importante, pois evita que o passageiro tenha que digitar novamente o mesmo destino ao executar o aplicativo. As sugestões expostas foram anotadas e adicionadas ao *Backlog* do aplicativo para que sejam implementadas futuramente.

# 5 CONCLUSÃO

O trabalho teve como principal objetivo descrever o desenvolvimento da construção inicial de um aplicativo de mobilidade urbana com funcionalidades que deem mais segurança tanto para o passageiro quanto para o motorista. Embora o aplicativo ainda esteja em desenvolvimento, as principais funcionalidades foram apresentadas, além das tecnologias e técnicas em que o aplicativo foi construído. O trabalho não explorou o módulo de pagamento, pois este será implementado por terceiros como MercadoPago ou Stripe.

O Seleto tem como objetivo proporcionar um sistema que provê mais segurança tanto para o passageiro quanto para o motorista como analisado nas funcionalidades 3.5. Portanto a validação das informações providas pelo passageiro durante o cadastramento, no caso de uso 4.14, e somente aceitar o pagamento pelo cartão de crédito são funcionalidades essenciais para o funcionamento do sistema que forneça segurança para os seus usuários.

## 5.1 Trabalhos Futuros

A versão desenvolvida até o momento foi adaptada para contemplar as funcionalidades básicas do MVP em relação ao escopo e ao tempo disponível. O sistema apresentado neste Trabalho de Conclusão possui o objetivo de expandir e aprimorar o produto. Dessa forma, essa seção apresenta os principais itens que serão desenvolvidos no projeto.

### 5.1.1 Android

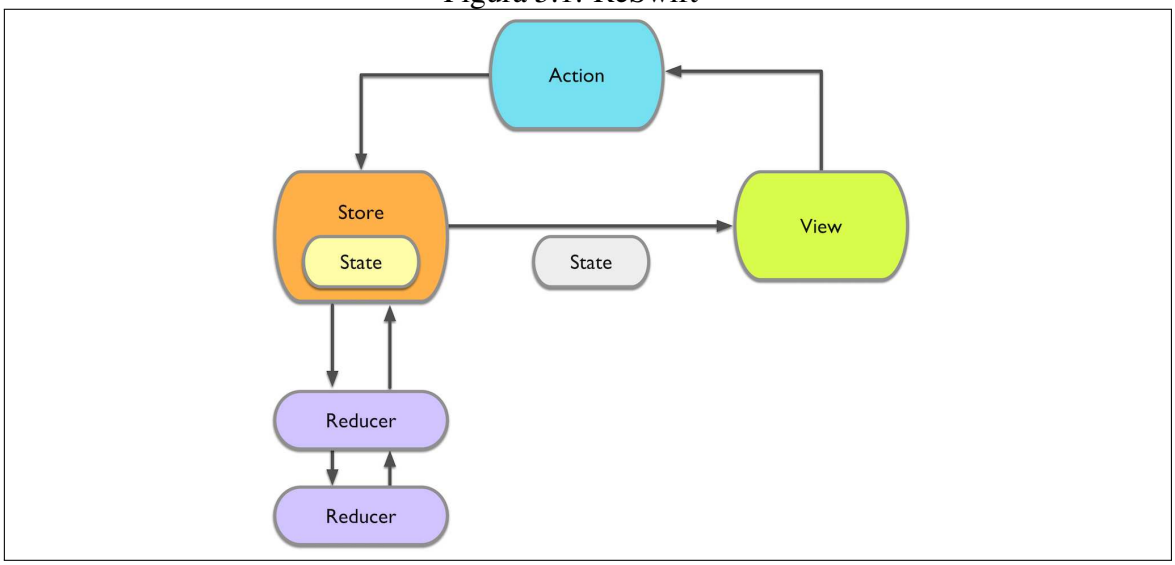
A opção pela escolha do desenvolvimento de uma versão inicial para iOS está relacionada com o conhecimento técnico da equipe e, também, como forma de validação do projeto. Segundo (KANTARWORDPANEL, 2017), o número de dispositivos iOS existentes no Brasil correspondem a 4,3% enquanto os smartphones que utilizam o sistema operacional Android possuem 92,1% do mercado em 2017. Dessa forma, é indispensável o desenvolvimento de uma versão do sistema para a plataforma Android. O objetivo é lançar a versão iOS e receber feedbacks para o desenvolvimento de novas versões e ajustes. Após essa análise, será desenvolvido o sistema Android contemplando essas alterações.

A estrutura do servidor foi desenvolvida considerando múltiplas plataformas e, assim, a integração com essa nova versão deverá ser mais fácil considerando o conhecimento do negócio adquirido no desenvolvimento da versão iOS e da facilidade desenvolvida no servidor.

### 5.1.2 Arquitetura

Como foi apresentado no decorrer do trabalho, o aplicativo contém inúmeros estados os quais impactam a tela do aplicativo. Por exemplo, quando o passageiro está selecionando os campos de origem e destino, quando passageiro requisita a corrida ou quando o motorista aceita uma corrida, a interface do aplicativo muda dependendo do estado em um determinado momento. Uma arquitetura que gerência melhor o estado é Flux, apresentada e utilizada pelo Facebook, a qual tem somente um ciclo unidirecional, ou seja, uma vez que o estado da aplicação mude, é preciso passar pelo ciclo, logo a interface mantêm o estado consistente. Entre as várias implementações desta arquitetura, frameworks como (REDUX, 2017) e (RESWIFT, 2017) baseiam-se nesta arquitetura.

Figura 5.1: ReSwift




Fonte: Autor

ReSwift implementam uma camada a mais durante o ciclo unidirecional, a qual é chamada de *Reducer* a qual gerencia o estado das entidades antes de apresentá-las a view. Portanto, esta arquitetura demostra um melhor gerenciamento de estados perante à aplicação.

# REFERÊNCIAS

AMAA. **Associação dos motoristas autônomos por aplicativo**. 2017. Disponível em: <https://noticias.uol.com.br/ultimas-noticias/bbc/2016/09/19/com-pagamento-em-dinheiro-motorista-do-uber-ve-crescer-risco-de-assaltos-em-sao-paulo.htm>. Acessado em: 07/06/2017.

APPLE. **Aplle**. 2017. Disponível em: <https://www.apple.com/br/swift/>. Acessado em: 07/06/2017.

CABIFY. **Cabify**. 2017. Disponível em: <https://cabify.com>. Acessado em: 01/07/2017.

CAREEM. **Careem**. 2017. Disponível em: <https://www.careem.com/dubai/node>. Acessado em: 01/07/2017.

CLOUDCODE. **CloudCode**. 2017. Disponível em: <http://docs.parseplatform.org/cloudcode/guide/>. Acessado em: 01/07/2017.

FABRIC. **Fabric**. 2017. Disponível em: <https://fabric.io/kits?utm_campaign=fabric-marketing&utm_medium=natural>. Acessado em: 01/07/2017.

FERNANDEZ, M. F. O. Serverless architectures. 2016.

FLURRY. **Flurry year-over-year**. 2017. Disponível em: <http://flurrymobile.tumblr.com/post/155761509355/on-their-tenth-anniversary-mobile-apps-start>. Acessado em: 07/06/2017.

FLUX. **Flux Arquitecture**. 2017. Disponível em: <https://facebook.github.io/flux/>. Acessado em: 07/06/2017.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1. ed. Addison-Wesley Professional, 1994. ISBN 0201633612. Available from Internet: <http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1>.

IPEA. **Atlas da violência**. 2017. Disponível em: <http://ipea.gov.br/atlasviolencia/>. Acessado em: 07/06/2017.

KANTARTNS. **Le Bipe - World Mobility Observatory**. 2017. Disponível em: <http://go.tnsglobal.com/emerging-trends-in-urban-mobility?utm_source=global-website&utm_campaign=mobility>. Acessado em: 07/06/2017.

KANTARWORDPANEL. **Análise da utilização de Smartphones**. 2017. Disponível em: <https://www.kantarworldpanel.com/global/smartphone-os-market-share/>. Acessado em: 07/06/2017.

LIU, Q.; SUN, X. Research of web real-time communication based on web socket. **International Journal of Communications, Network and System Sciences**, Scientific Research Publishing, Inc,, v. 05, n. 12, p. 797–801, 2012. Available from Internet: <https://doi.org/10.4236/ijcns.2012.512083>.

LYFT. **Lyft**. 2017. Disponível em: <https://www.lyft.com>. Acessado em: 01/07/2017.

MAROTTO. **ParseServer**. 2016. Disponível em: <http://blog.parse.com/announcements/introducing-parse-server-and-the-database-migration-tool>. Acessado em: 01/07/2017.

MERCADOPAGO. **MercadoPago**. 2017. Disponível em: <https://www.mercadopago.com.br/developers/>. Acessado em: 07/06/2017.

MIXPANEL. **Mixpanel**. 2017. Disponível em: <https://mixpanel.com/trends/#report/ios_10>. Acessado em: 01/07/2017.

MONGODB. **MongoDB**. 2016. Disponível em: <http://mongodb.com>. Acessado em: 01/07/2017.

MONGODBDOCS. **MongoDB Documentação**. 2017. Disponível em: <https://docs.mongodb.com>. Acessado em: 07/06/2017.

OSTERWALDER, A.; PIGNEUR, Y.; CLARK, T. **Business model generation: a handbook for visionaries, game changers, and challengers**. Wiley, 2010. Available from Internet: <https://www.bookdepository.com/Business-Model-Generation-Alexander-Osterwalder/9780470876411>.

PARSE. **Parse**. 2017. Disponível em: <http://parseplatform.org/>. Acessado em: 01/07/2017.

PNAD (Ed.). **Pesquisa nacional por amostra de domicílios - PNAD**. Rio de Janeiro, RJ, Brasil: [s.n.], 2015.

REDUX. **Redux**. 2017. Disponível em : <http://redux.js.org>. Acessado em: 07/06/2017.

RESWIFT. **ReSwift**. 2017. Disponível em : <https://github.com/ReSwift/ReSwift#reswift>. Acessado em: 07/06/2017.

RUBIN, K. **Essential Scrum: A Practical Guide to the Most Popular Agile Process**. Addison-Wesley, 2012. (Addison-Wesley signature series). ISBN 9780137043293. Available from Internet: <https://books.google.com.br/books?id=HkXX65VCZU4C>.

SORENSEN, E.; MIHAILESC. Model-view-viewmodel (mvvm) design pattern using windows presentation foundation (wpf) technology. **MegaByte Journal**, p. 1–3, 2010. Available from Internet: <http://megabyte.utm.ro/articole/2010/info/sem1/InfoStraini_Pdf/1.pdf>.

STRIPE. **Stripe**. 2017. Disponível em: <https://stripe.com>. Acessado em: 07/06/2017.

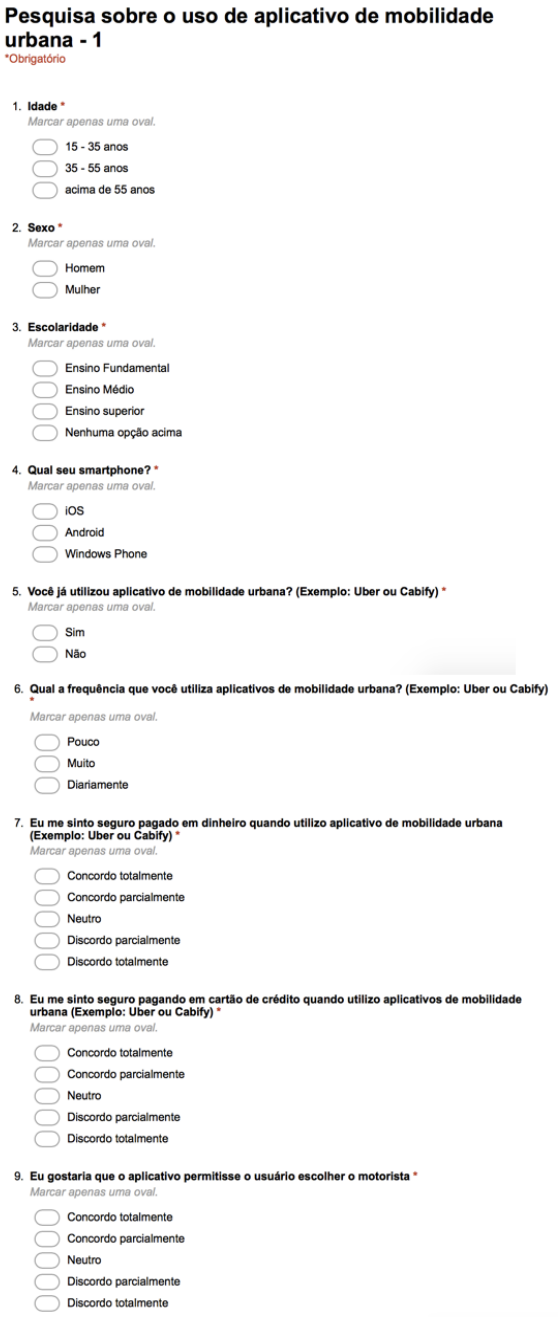
SWIFTGITHUB. **SwiftGithub**. 2017. Disponível em: <https://github.com/apple/swift>. Acessado em: 07/06/2017.

UBER. **Uber**. 2017. Disponível em: <https://www.uber.com/en-BR/>. Acessado em: 01/07/2017.

WEBSOCKETRFC. **WebSocketRFC**. 2011. Disponível em: <https://tools.ietf.org/html/rfc6455>. Acessado em: 01/07/2017.

# APÊNDICE A - PESQUISA SOBRE MOBILIDADE URBANA PARA PASSAGEIROS

Figura 5.2: Pesquisa sobre mobilidade para passageiro



**Pesquisa sobre o uso de aplicativo de mobilidade urbana - 1**

*Obrigatório

1. **Idade** *
   *Marcar apenas uma oval.*
   - 15 - 35 anos
   - 35 - 55 anos
   - acima de 55 anos

2. **Sexo** *
   *Marcar apenas uma oval.*
   - Homem
   - Mulher

3. **Escolaridade** *
   *Marcar apenas uma oval.*
   - Ensino Fundamental
   - Ensino Médio
   - Ensino superior
   - Nenhuma opção acima

4. **Qual seu smartphone?** *
   *Marcar apenas uma oval.*
   - iOS
   - Android
   - Windows Phone

5. **Você já utilizou aplicativo de mobilidade urbana? (Exemplo: Uber ou Cabify)** *
   *Marcar apenas uma oval.*
   - Sim
   - Não

6. **Qual a frequência que você utiliza aplicativos de mobilidade urbana? (Exemplo: Uber ou Cabify)** *
   *Marcar apenas uma oval.*
   - Pouco
   - Muito
   - Diariamente

7. **Eu me sinto seguro pagado em dinheiro quando utilizo aplicativo de mobilidade urbana (Exemplo: Uber ou Cabify)** *
   *Marcar apenas uma oval.*
   - Concordo totalmente
   - Concordo parcialmente
   - Neutro
   - Discordo parcialmente
   - Discordo totalmente

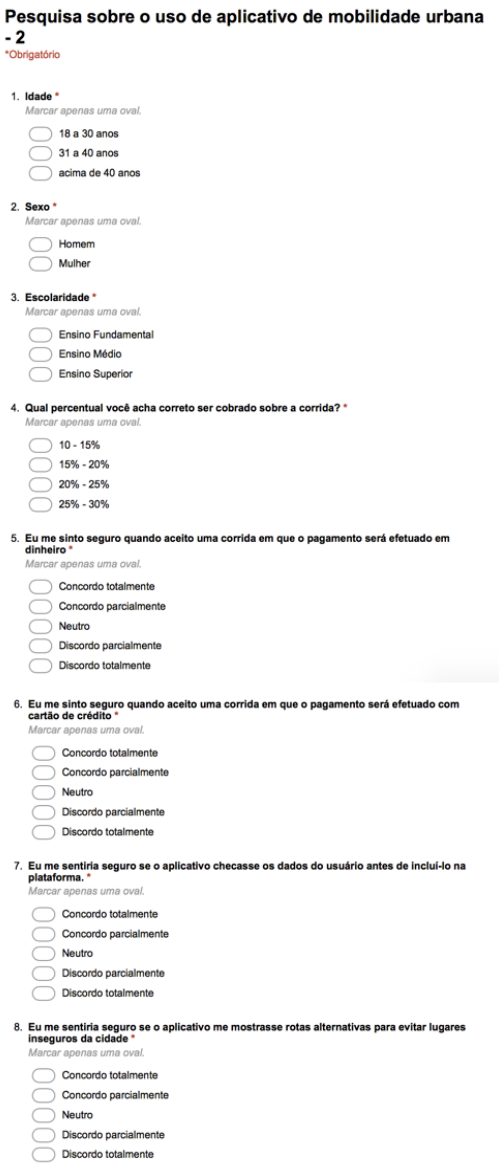
8. **Eu me sinto seguro pagando em cartão de crédito quando utilizo aplicativos de mobilidade urbana (Exemplo: Uber ou Cabify)** *
   *Marcar apenas uma oval.*
   - Concordo totalmente
   - Concordo parcialmente
   - Neutro
   - Discordo parcialmente
   - Discordo totalmente

9. **Eu gostaria que o aplicativo permitisse o usuário escolher o motorista** *
   *Marcar apenas uma oval.*
   - Concordo totalmente
   - Concordo parcialmente
   - Neutro
   - Discordo parcialmente
   - Discordo totalmente

Fonte: Autor

# APÊNDICE B - PESQUISA SOBRE MOBILIDADE URBANA PARA MOTORISTAS

Figura 5.3: Pesquisa sobre mobilidade para Motorista



**Pesquisa sobre o uso de aplicativo de mobilidade urbana - 2**

*Obrigatório

1. **Idade** *
   *Marcar apenas uma oval.*
   - 18 a 30 anos
   - 31 a 40 anos
   - acima de 40 anos

2. **Sexo** *
   *Marcar apenas uma oval.*
   - Homem
   - Mulher

3. **Escolaridade** *
   *Marcar apenas uma oval.*
   - Ensino Fundamental
   - Ensino Médio
   - Ensino Superior

4. **Qual percentual você acha correto ser cobrado sobre a corrida?** *
   *Marcar apenas uma oval.*
   - 10 - 15%
   - 15% - 20%
   - 20% - 25%
   - 25% - 30%

5. **Eu me sinto seguro quando aceito uma corrida em que o pagamento será efetuado em dinheiro** *
   *Marcar apenas uma oval.*
   - Concordo totalmente
   - Concordo parcialmente
   - Neutro
   - Discordo parcialmente
   - Discordo totalmente

6. **Eu me sinto seguro quando aceito uma corrida em que o pagamento será efetuado com cartão de crédito** *
   *Marcar apenas uma oval.*
   - Concordo totalmente
   - Concordo parcialmente
   - Neutro
   - Discordo parcialmente
   - Discordo totalmente

7. **Eu me sentiria seguro se o aplicativo checasse os dados do usuário antes de incluí-lo na plataforma.** *
   *Marcar apenas uma oval.*
   - Concordo totalmente
   - Concordo parcialmente
   - Neutro
   - Discordo parcialmente
   - Discordo totalmente

8. **Eu me sentiria seguro se o aplicativo me mostrasse rotas alternativas para evitar lugares inseguros da cidade** *
   *Marcar apenas uma oval.*
   - Concordo totalmente
   - Concordo parcialmente
   - Neutro
   - Discordo parcialmente
   - Discordo totalmente

Fonte: Autor

# APÊNDICE C - PESQUISA SOBRE TESTE DE FUNCIONALIDADES PARA USUÁRIOS

Figura 5.4: Pesquisa sobre teste de funcionalidades

**Teste de funcionalidades app de mobilidade Urbana**

Esse questionário visa avaliar a utilização das funcionalidades do aplicativo de mobilidade urbana Seleta. Para responder as perguntas deve ser considerada a escala de 1 a 5, sendo 1 NADA FACIL e 5 MUITO FÁCIL

*Obrigatório

1. **Sexo** *
   *Marcar apenas uma oval.*
   - ◯ Feminino
   - ◯ Masculino

2. **Escolaridade** *
   *Marcar apenas uma oval.*
   - ◯ Ensino Fundamental
   - ◯ Ensino Médio
   - ◯ Ensino Superior Incompleto
   - ◯ Ensino Superior Completo
   - ◯ Pós-Graduação

3. **Foi fácil adicionar suas informações para se cadastrar no aplicativo?** *
   *Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Nada Fácil | ◯ | ◯ | ◯ | ◯ | ◯ | Muito Fácil |

4. **Foi fácil incluir os endereços do percurso desejado no aplicativo?** *
   *Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Nada Fácil | ◯ | ◯ | ◯ | ◯ | ◯ | Muito Fácil |

5. **Foi fácil selecionar um motorista no aplicativo?** *
   *Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Nada Fácil | ◯ | ◯ | ◯ | ◯ | ◯ | Muiot Fácil |

6. **Foi fácil chamar uma corrida no aplicativo?** *
   *Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Nada Fácil | ◯ | ◯ | ◯ | ◯ | ◯ | Muiot Fácil |

Fonte: Autor