

---

# **Documentação de Projeto**

**para o sistema**

## **MobU**

**Versão 1.0**

Projeto de sistema elaborado pelo(s) aluno(s) Joaquim de Moura Thomaz Neto e apresentado ao curso de **Engenharia de Software** da **PUC Minas** como parte do Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo dos professores Danilo de Quadros Maia Filho, Leonardo Vilela Cardoso, Raphael Ramos Dias Costa, orientação acadêmica do professor Cleiton Silva Tavares e orientação de TCC II do professor (a ser definido no próximo semestre).

**16/11/2025**

# Tabela de Conteúdo

<b>Tabela de Conteúdo</b>	<b>ii</b>
<b>Histórico de Revisões</b>	<b>ii</b>
<b>1. Introdução</b>	1
<b>2. Modelo de Requisitos</b>	<b>1</b>
2.1    Descrição de Atores	1
2.2    Modelo de Casos de Uso	4
<b>3. Modelo de Projeto</b>	<b>11</b>
3.1    Diagrama de Classes	11
3.2    Diagramas de Sequência	15
3.3    Diagramas de Comunicação	17
3.4    Arquitetura Lógica: Diagramas de Pacotes	20
3.5    Diagramas de Estados	22
3.6    Diagrama de Componentes	24
<b>4. Projeto de Interface com Usuário</b>	<b>28</b>
4.1    Interfaces Comuns a Todos os Atores	28
4.2    Interfaces Usadas pelo Administrador	29
4.3    Interfaces Usadas pelo Motorista	33
4.4    Interfaces Usadas pelo Passageiro	35
<b>5. Modelo de Dados</b>	<b>37</b>
<b>6. Modelo de Teste</b>	<b>45</b>
6.1    Testes de Aceitação — Necessidade 1: Passageiro solicitar corrida	46
6.1.2    Necessidade 2 — Motorista aceitar e realizar corrida	47
6.1.3    Necessidade 3 — Pagamento via PIX Off-line	48
6.1.4    Necessidade 4 — Administração de Usuários e Monitoramento do Sistema	49
6.2    Testes de Integração	50
<b>7. Cronograma e Processo de Implementação</b>	<b>56</b>
7.1    Cronograma	56

# Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Entrega 3	17/09/2025	Início do documento, definindo seções 2.1, 2.2, 2.3 e 4	1.0
Entrega 4	06/10/2025	Inclusão das seções 2.4, 3.1, 3.2 e 3.3	2.0
Entrega 5	31/10/2025	Inclusão das seções 3.4, 3.5, 3.6 e 5	3.0
Entrega 6	12/11/2025	Inclusão da seção 6, 7 e 7.1	4.0
Entrega 7	25/11/2025	Inclusão da seção 6.1, 6.1.2, 6.1.3, 6.1.4 e 6.2	5.0


## 1. Introdução

Este documento agrega: 1) a elaboração e revisão dos modelos de domínio e 2) os modelos de projeto para o sistema MobU. A referência principal para a descrição geral do problema, domínio e requisitos do sistema é o documento de especificação que descreve a visão de domínio do sistema, que acompanha este documento. Anexo a este documento também se encontra o Glossário.

O sistema MobU tem como proposta o desenvolvimento de uma plataforma de mobilidade urbana que conecta passageiros, motoristas e administradores em um ecossistema digital unificado. Seu objetivo é otimizar a solicitação e realização de corridas, oferecendo uma experiência intuitiva e segura tanto para usuários quanto para motoristas, além de permitir o acompanhamento em tempo real, integração com meios de pagamento (PIX/dinheiro) e recursos administrativos para análise de métricas, gestão de regiões e resolução de disputas.

## 2. Modelos de Usuário e Requisitos

Esta seção tem como propósito caracterizar os usuários e atores envolvidos no sistema, além de especificar os requisitos que deverão ser atendidos. Para isso, são apresentadas descrições resumidas de cada ator, acompanhadas de modelos que representam seu papel como usuário da aplicação. Em seguida, são expostos o diagrama de casos de uso e as histórias de usuário, que servirão como base para orientar o desenvolvimento do sistema. Por fim, são disponibilizados o diagrama de sequência e o contrato de operações.

### 2.1 Descrição de Atores

**Passageiro:** Usuário que utiliza o aplicativo para solicitar corridas. É responsável por informar origem e destino, acompanhar o trajeto em tempo real, realizar o pagamento e avaliar o motorista ao final da viagem. Seu principal objetivo é realizar deslocamentos com praticidade, segurança e previsibilidade de custo.

**Motorista:** Usuário que utiliza o aplicativo para oferecer o serviço de transporte. Recebe solicitações de corridas, navega até o ponto de embarque e conduz o passageiro ao destino informado. Pode consultar seu histórico de corridas e acompanhar seus ganhos. É responsável por manter informações atualizadas no sistema e aceitar ou recusar corridas conforme disponibilidade.

**Administrador:** Responsável por gerenciar o funcionamento da plataforma por meio de um sistema web. Controla o cadastro de passageiros e motoristas, monitora as corridas realizadas, acompanha indicadores de utilização e gera relatórios financeiros. Também pode configurar taxas da plataforma e atuar em situações que demandem suporte ou resolução de conflitos.

## 2.2 Modelos de Usuários

Esta subseção tem como objetivo apresentar os modelos de usuários construídos a partir da definição de personas. Para a elaboração dessas personas, foram consideradas as características dos principais atores do sistema como passageiros, motoristas e administradores, bem como suas necessidades, dores e objetivos dentro do contexto da aplicação. As personas a seguir representam perfis típicos dos usuários previstos para a plataforma e servem como referência para orientar decisões de *design*, desenvolvimento e priorização de funcionalidades do sistema.

A Tabela 1 descreve a persona da usuária Mariana Silva, uma passageira que depende do transporte urbano para deslocamentos diários. É possível identificar que, apesar de estar familiarizada com aplicativos de uso comum como *WhatsApp* e *Instagram*, ela sente dificuldades em confiar nos aplicativos locais de transporte por considerá-los confusos e pouco práticos. Também é perceptível que Mariana valoriza a previsibilidade no preço e segurança durante as viagens. Por fim, após análise, observa-se que ela deseja ter maior clareza no custo antes da corrida e a possibilidade de acompanhar o trajeto em tempo real, o que lhe traria mais confiança no serviço.

Mariana Silva	
Descrição	Mariana possui 27 anos, nasceu e cresceu na cidade onde o aplicativo será implantado. Trabalha como recepcionista em uma clínica e depende de transporte diário para se deslocar ao trabalho e para compromissos pessoais. Costuma usar aplicativos como <i>WhatsApp</i> e <i>Instagram</i> , mas nunca utilizou os aplicativos de transporte já existentes na cidade, pois os considera confusos e pouco confiáveis. Busca uma alternativa prática que lhe permita se deslocar com previsibilidade de preço e segurança.
Dores	<ul style="list-style-type: none"> <li>• Dificuldade em prever o valor de cada corrida com os aplicativos atuais.</li> <li>• Insegurança por não poder acompanhar a rota em tempo real.</li> <li>• Pouca confiança em aplicativos locais, pela falta de avaliação de motoristas.</li> </ul>
Objetivos	<ul style="list-style-type: none"> <li>• Ter acesso a corridas rápidas, seguras e acessíveis.</li> <li>• Saber o valor estimado da corrida antes de embarcar.</li> <li>• Acompanhar em tempo real o trajeto, sentindo-se mais segura.</li> </ul>

Tabela 1. Persona Mariana Silva

A Tabela 2 descreve a persona do usuário Carlos Andrade, um motorista que utiliza o transporte por aplicativo como fonte de renda complementar. Identifica-se que ele já teve experiências anteriores com aplicativos locais, mas considera que eles carecem de recursos de apoio ao motorista e transparência em relação aos ganhos. Nota-se também que Carlos valoriza ferramentas que o ajudam a organizar seu trabalho de forma prática, sem exigir aprendizado complexo de novas tecnologias. Por fim, após análise, percebe-se que ele busca um sistema confiável que facilite o recebimento de corridas e forneça relatórios claros de viagens e rendimentos.

Carlos Andrade	
Descrição	Carlos tem 42 anos e trabalha como motorista de aplicativo em tempo parcial para complementar a renda da família. Já testou os aplicativos locais (LevaAli e MobyGo), mas considera que eles oferecem pouco suporte e não têm transparência nos ganhos. Ele valoriza poder organizar suas corridas e acompanhar de forma simples quanto está ganhando por dia e por semana. Carlos utiliza apenas o necessário em termos de tecnologia, mas aprende rápido quando percebe que o recurso facilita sua rotina.
Dores	<ul style="list-style-type: none"> <li>● Falta de relatórios claros sobre corridas e ganhos nos aplicativos atuais.</li> <li>● Necessidade de depender de chamadas aleatórias, sem controle.</li> <li>● Dificuldade em manter contato com suporte ou resolver problemas por falta de recursos internos no <i>app</i>.</li> </ul>
Objetivos	<ul style="list-style-type: none"> <li>● Receber chamadas de corridas de forma simples e rápida.</li> <li>● Acompanhar histórico de viagens e rendimento financeiro.</li> <li>● Trabalhar em um sistema confiável que ofereça suporte adequado.</li> </ul>

Tabela 2. Persona Carlos Andrade

A Tabela 3 descreve a persona da usuária Fernanda Oliveira, que atua como administradora da plataforma. É possível observar que ela precisa equilibrar a gestão do sistema com suas outras responsabilidades profissionais, o que exige relatórios padronizados e práticos para tomada de decisão. Identifica-se ainda que Fernanda encontra dificuldades em monitorar o desempenho da plataforma com os recursos limitados oferecidos pelos aplicativos concorrentes. Por fim, após análise, verifica-se que sua principal necessidade é dispor de ferramentas que permitam acompanhar a satisfação dos usuários, controlar cadastros e configurar taxas de forma objetiva.

Fernanda Oliveira	
Descrição	Fernanda possui 35 anos, formada em Administração e atua como secretária de gestor da plataforma. Seu papel é garantir que motoristas e passageiros estejam devidamente cadastrados e que o sistema funcione de forma estável. Ela precisa ter acesso a relatórios que mostrem número de corridas realizadas, ganhos, taxas e indicadores de uso do sistema, de modo a tomar decisões sobre melhorias e acompanhar o crescimento do aplicativo.
Dores	<ul style="list-style-type: none"> <li>● Falta de relatórios padronizados nos aplicativos locais.</li> <li>● Dificuldade em monitorar qualidade do serviço (avaliações e <i>feedbacks</i>).</li> <li>● Necessidade de ajustar taxas e políticas de forma manual.</li> </ul>
Objetivos	<ul style="list-style-type: none"> <li>● Ter acesso a relatórios claros e detalhados sobre o funcionamento do sistema.</li> <li>● Acompanhar o desempenho de motoristas e satisfação de passageiros.</li> <li>● Configurar taxas e gerenciar usuários de forma prática.</li> </ul>

Tabela 3. Persona Fernanda Oliveira

## 2.3 Modelo de Casos de Uso e Histórias de Usuários

Esta subseção tem como finalidade apresentar os casos de uso e as histórias de usuário que orientam o desenvolvimento do sistema. Para isso, é exibido o diagrama de casos de uso, no qual estão representadas as principais interações entre os atores e a aplicação. Em complemento, são descritas as histórias de usuário correspondentes, detalhando as funcionalidades esperadas e permitindo uma visão clara dos requisitos que devem ser implementados.

### 2.3.1 Diagrama de Casos de Uso

A Figura 1 apresenta o diagrama de casos de uso do sistema proposto. No diagrama, são representados três atores principais: Passageiro, Motorista e Administrador, que interagem diretamente com o sistema. Além deles, são exibidos dois sistemas externos: o Serviço de Mapas e Geolocalização, utilizado para cálculo de rotas e estimativas de tempo, e o *Gateway* de Pagamentos(Futuramente). O passageiro acessa o sistema para criar conta, solicitar corridas, acompanhar o trajeto, efetuar pagamento e avaliar o motorista. O Motorista utiliza o sistema para receber corridas, aceitar ou recusar solicitações, navegar até o passageiro e ao destino, marcar etapas da viagem e consultar relatórios de ganhos. O Administrador gerencia usuários, motoristas, taxas, regiões de operação, relatórios do sistema e solicitações de suporte. Ela representa visualmente essas interações, bem como as relações entre os casos de uso e os sistemas externos.

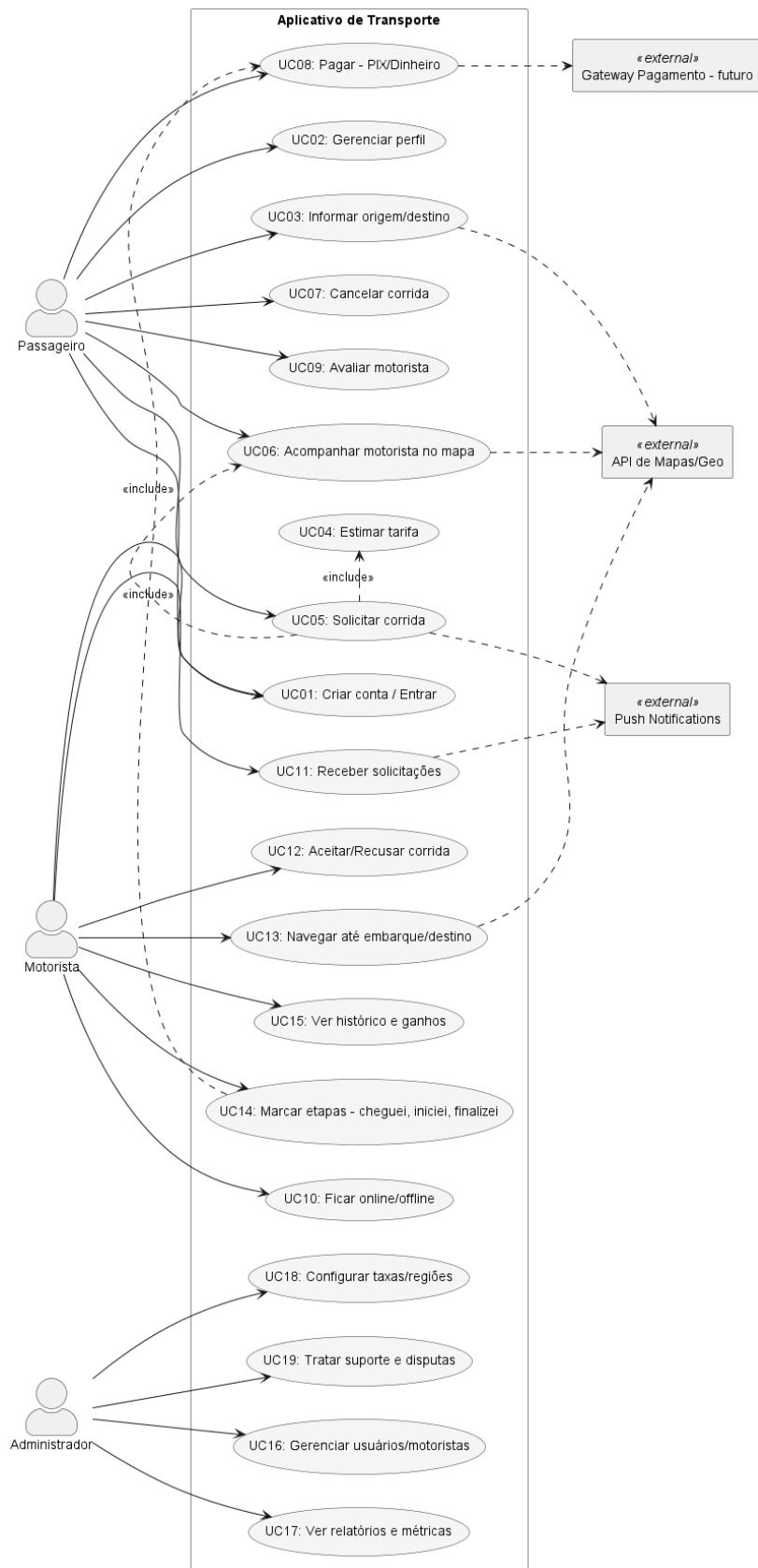


Figura 1 – Diagrama de Casos de Uso

### 2.3.2 Histórias de Usuário

Nesta seção, são listadas as histórias de usuário levantadas para o sistema proposto. Para fins de organização, utiliza-se identificadores no formato *US#ID*, em que *US* se refere a *User Story*.

As histórias de usuário identificadas para o sistema são:

- US1. Como passageiro, eu gostaria de criar uma conta e gerenciar meu perfil para que eu possa acessar o sistema e utilizar seus serviços.
- US2. Como passageiro, eu gostaria de informar origem e destino para que eu possa solicitar uma corrida.
- US3. Como passageiro, eu gostaria de ver a estimativa de tarifa e tempo de chegada do motorista para decidir se quero confirmar a corrida.
- US4. Como passageiro, eu gostaria de acompanhar a rota do motorista em tempo real para saber onde ele está.
- US5. Como passageiro, eu gostaria de efetuar o pagamento via PIX ou em dinheiro para concluir a corrida com segurança.
- US6. Como passageiro, eu gostaria de avaliar o motorista após a corrida para contribuir com a reputação da plataforma.
- US7. Como motorista, eu gostaria de entrar no sistema e definir meu *status* como *online* ou *offline* para indicar quando estou disponível para receber corridas.
- US8. Como motorista, eu gostaria de receber solicitações de corrida com informações de origem, destino e valor estimado para decidir se aceito a viagem.
- US9. Como motorista, eu gostaria de navegar até o passageiro e depois até o destino usando o mapa do *app* para otimizar a rota.
- US10. Como motorista, eu gostaria de marcar as etapas da corrida (cheguei, iniciei, finalizei) para registrar corretamente o andamento da viagem.
- US11. Como motorista, eu gostaria de acessar um painel com meus ganhos diários e semanais para acompanhar meus resultados.
- US12. Como administrador, eu gostaria de gerenciar contas de passageiros e motoristas para manter o sistema organizado e seguro.

US13. Como administrador, eu gostaria de acompanhar relatórios de corridas e métricas do sistema para tomar decisões estratégicas.

US14. Como administrador, eu gostaria de configurar taxas e regiões de operação para ajustar o funcionamento da plataforma conforme a cidade.

US15. Como administrador, eu gostaria de visualizar e responder solicitações de suporte ou disputas para resolver problemas de forma rápida.

## 2.4 Diagrama de Sequência do Sistema e Contrato de Operações

Nesta seção, são apresentados os diagramas de sequência do sistema, assim como seus Contratos de Operações. O objetivo destes diagramas é descrever os fluxos de interação existentes entre os usuários e o sistema, representando a troca de mensagens e as dependências entre módulos da aplicação.

A Figura 2 representa o diagrama de sequência do sistema relacionado ao fluxo de solicitação de corrida. Nesse fluxo, o Passageiro informa sua origem e destino e, ao confirmar o pedido, o sistema consulta o Serviço de Mapas para calcular a rota e o tempo estimado.

Após essa etapa, a aplicação envia uma notificação em tempo real ao Motorista disponível mais próximo, que poderá aceitar ou recusar a corrida.

Esse diagrama está diretamente relacionado aos casos de uso UC03 (Informar origem/destino), UC04 (Estimar tarifa), UC05 (Solicitar corrida) e UC11 (Receber solicitação).

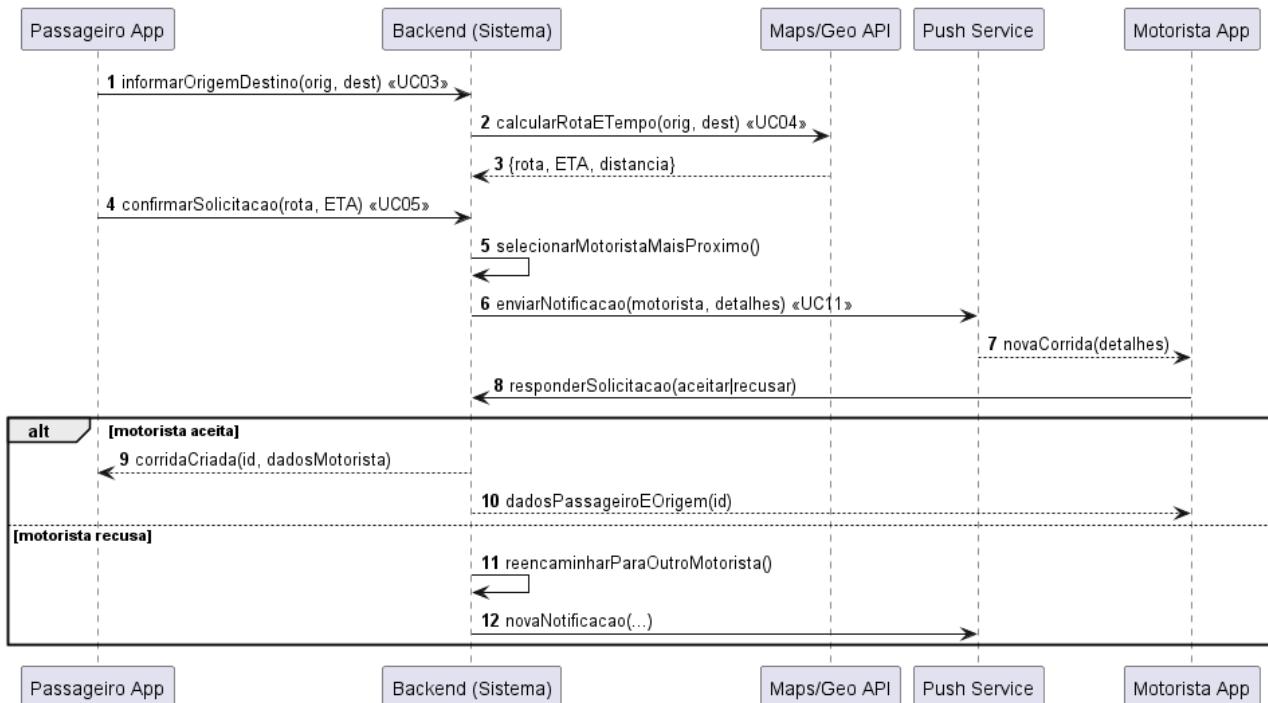


Figura 2 – Diagrama de Sequência Solicitar Corrida e Notificar Motorista

<b>Contrato</b>	Solicitar corrida e notificar motorista
<b>Operação</b>	<i>requestRide(origin: String, destination: String)</i>
<b>Referências cruzadas</b>	UC03 Informar origem/destino, UC04 Estimar tarifa, UC05 Solicitar corrida, UC11 Receber solicitação
<b>Pré-condições</b>	O passageiro deve estar autenticado no sistema e com geolocalização ativa
<b>Pós-condições</b>	O motorista mais próximo é notificado e a corrida é registrada no banco de dados como “pendente”

Tabela 4. Contrato Solicitar Corrida e Notificar Motorista

A Figura 3 mostra o diagrama de sequência referente ao fluxo de pagamento da corrida. Após o término da viagem, o sistema apresenta as opções de PIX ou Dinheiro. No processo de pagamento em dinheiro ou pix, o sistema atualiza o status localmente. Esse fluxo está relacionado aos casos de uso UC08 (Efetuar pagamento) e UC14 (Finalizar corrida).

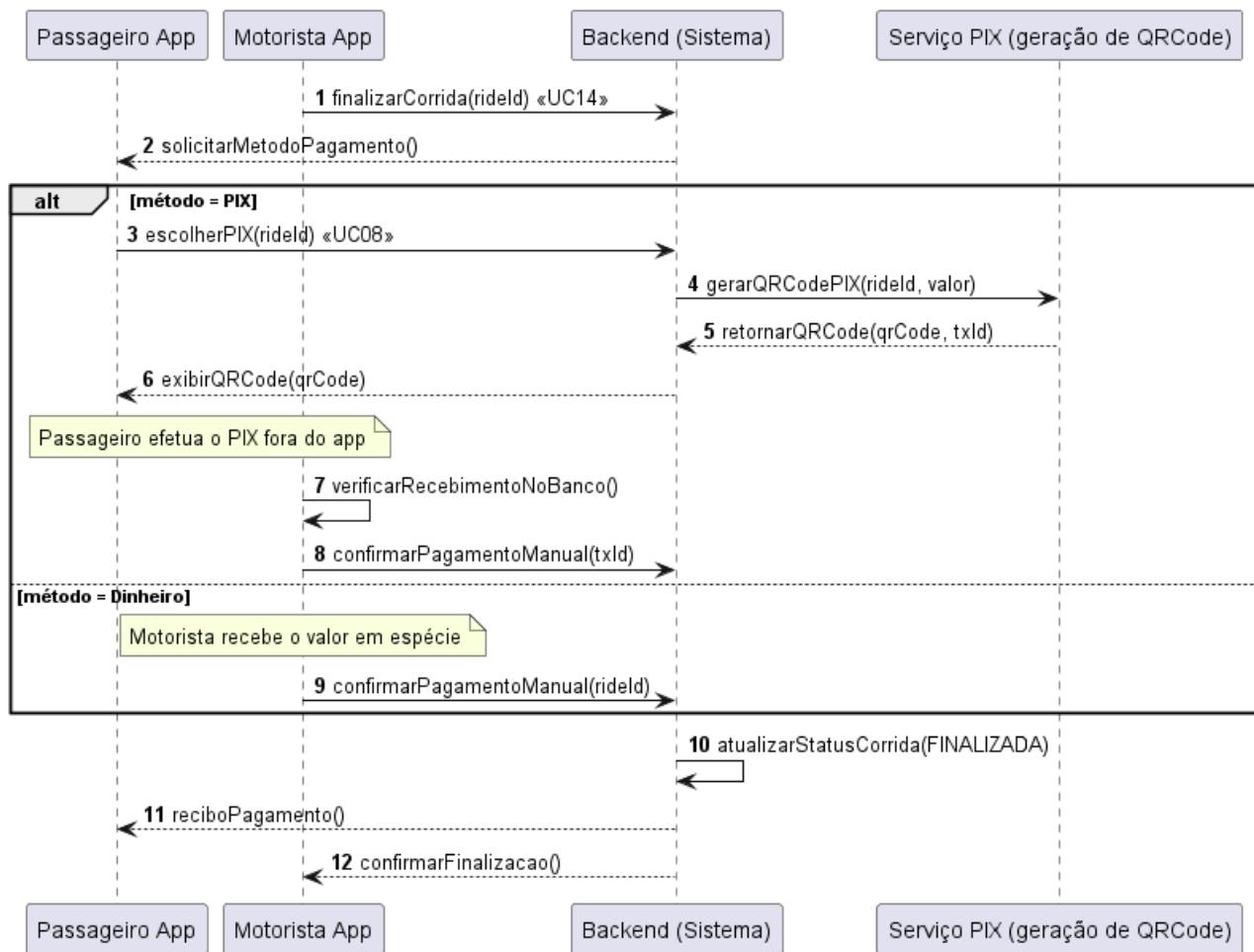


Figura 3 – Diagrama de Sequência Efetuar Pagamento

<b>Contrato</b>	Efetuar pagamento da corrida
<b>Operação</b>	<i>processPayment(paymentMethod: Enum, rideId: Int)</i>

<b>Referências cruzadas</b>	UC08 Efetuar pagamento, UC14 Finalizar corrida
<b>Pré-condições</b>	A corrida deve ter sido concluída e estar aguardando pagamento
<b>Pós-condições</b>	O pagamento é confirmado (PIX) ou marcado como concluído (dinheiro), alterando o status da corrida para “finalizada”

Tabela 5. Contrato Efetuar Pagamento da Corrida

A Figura 4 representa o diagrama de sequência do fluxo de avaliação do motorista. Após o encerramento da corrida e o processamento do pagamento, o Passageiro pode registrar uma avaliação que inclui nota e comentário. O sistema valida os dados e salva a avaliação, atualizando a média do motorista no banco de dados. Esse fluxo está associado ao caso de uso UC09 (Avaliar motorista).

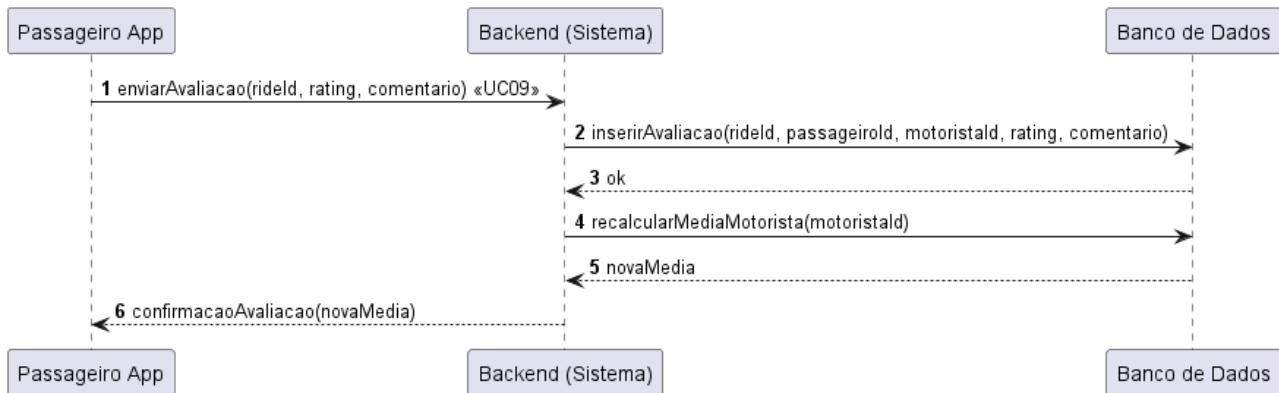


Figura 4 – Diagrama de Sequência Avaliação do Motorista

<b>Contrato</b>	Registrar avaliação do motorista
<b>Operação</b>	<code>submitReview(rideId: Int, rating: Int, comment: String)</code>
<b>Referências cruzadas</b>	UC09 Avaliar motorista
<b>Pré-condições</b>	A corrida deve ter sido finalizada e o pagamento concluído
<b>Pós-condições</b>	A avaliação é registrada e a nota média do motorista é atualizada no sistema

Tabela 6. Contrato Registrar Avaliação do Motorista

A Figura 5 ilustra o diagrama de sequência do painel administrativo, que demonstra como o Administrador acessa o sistema web, solicita relatórios e recebe informações agregadas de corridas, motoristas e usuários. O fluxo inclui a requisição dos dados, o processamento interno e a resposta exibida em forma de gráficos e métricas. Esse diagrama se relaciona aos casos de uso UC16 (Gerenciar usuários/motoristas), UC17 (Ver relatórios e métricas) e UC18 (Configurar taxas/regiones).

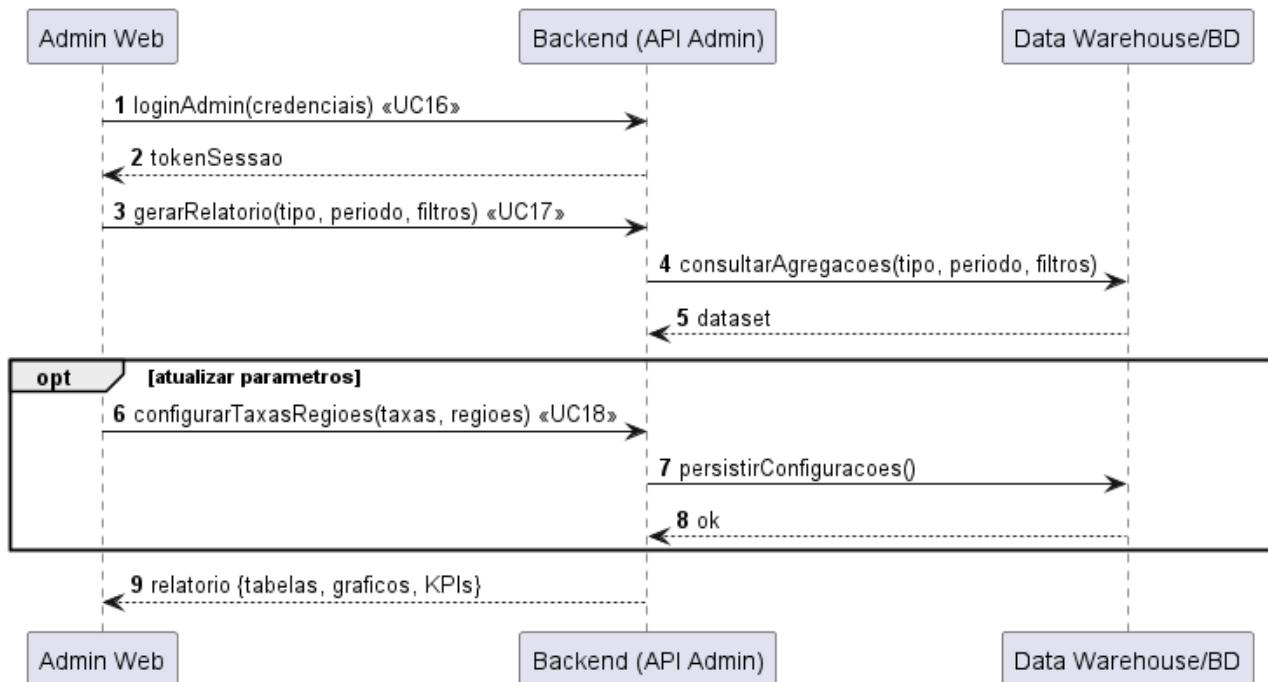


Figura 5 – Diagrama de Sequência Gerar Relatórios Administrativos

<b>Contrato</b>	Gerar relatórios administrativos
<b>Operação</b>	<code>generateReport(type: Enum, dateRange: String)</code>
<b>Referências cruzadas</b>	UC16 Gerenciar usuários, UC17 Ver relatórios, UC18 Configurar taxas
<b>Pré-condições</b>	O administrador deve estar autenticado no painel web
<b>Pós-condições</b>	O relatório solicitado é gerado e apresentado na interface em formato gráfico

Tabela 7. Contrato Gerar Relatórios Administrativos

### 3. Modelos de Projeto

Nesta seção é apresentada a modelagem dos objetos e fluxos que compõem o sistema proposto. Para isso, foram desenvolvidos diagramas de classes que descrevem os principais pacotes do sistema: *controllers*, *services*, *domain/models*, *repositories* e *dtos*. Cada um deles tem como objetivo representar as responsabilidades e inter-relações entre os componentes que compõem o aplicativo de transporte.

Os diagramas a seguir foram projetados para fornecer uma visão clara e modular da arquitetura, evidenciando como os diferentes níveis — interface, lógica de negócios, persistência e dados — interagem para garantir o funcionamento adequado do sistema.

#### 3.1 Diagrama de Classes

A seguir é apresentado o diagrama de classes referente ao pacote de *controllers* do sistema. Essas classes têm como responsabilidade intermediar as requisições oriundas do aplicativo móvel com as camadas de serviço, realizando a comunicação por meio de uma *API REST*.

O pacote contém controladores para autenticação, operações de passageiros, operações de motoristas, gerenciamento de corridas, pagamentos e administração.

Cada controlador expõe métodos que representam os principais casos de uso da aplicação, como solicitar corrida, responder a uma solicitação, acompanhar viagens, registrar pagamentos e gerar relatórios administrativos.

A Figura 6 ilustra o diagrama de classes do pacote *controllers*, evidenciando as dependências diretas entre os controladores e os serviços correspondentes que contêm as regras de negócio.

É possível observar a relação entre *PassageiroController* e *CorridaService*, *MotoristaController* e *RelatorioService*, entre outras.

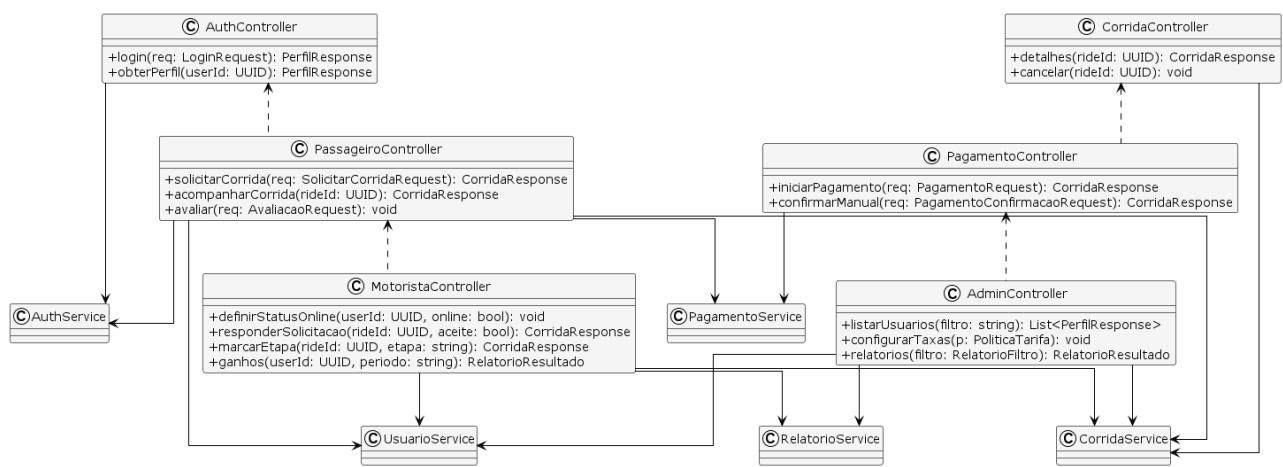


Figura 6 – Diagrama de Classes: Controllers

O diagrama de classes apresentado a seguir representa o pacote de *services*, responsável pela camada de regras de negócio do aplicativo.

Essas classes implementam as operações centrais da aplicação, como o gerenciamento de corridas, cálculo de tarifas, pareamento entre motoristas e passageiros, processamento de pagamentos e geração de relatórios.

O pacote inclui serviços de autenticação (*AuthService*), gestão de usuários (*UsuarioService*), cálculo de preço dinâmico (*PrecificacaoService*), correspondência de motoristas (*MatchingService*), gerenciamento de *status* (*MotoristaStatusService*), controle de corridas (*CorridaService*), pagamentos (*PagamentoService*) e notificações (*NotificacaoService*).

Há ainda o *RelatorioService*, que fornece métricas de desempenho e estatísticas operacionais tanto para motoristas quanto para administradores.

As Figuras 7 e 8 apresentam o diagrama de classes dos *services*, mostrando as dependências com os repositórios de persistência e com integrações externas como *APIs* de mapas e serviços de *push*.

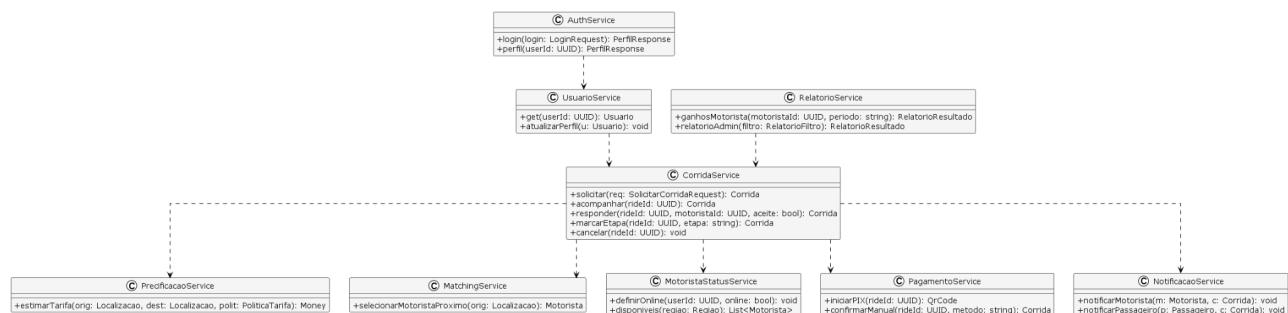


Figura 7 – Diagrama de Classes: Services01



Figura 8 – Diagrama de Classes: Services02

O diagrama de classes a seguir descreve o pacote *domain*, que contém os modelos de dados fundamentais do sistema. Essas classes representam as entidades centrais que sustentam o funcionamento do aplicativo, como *Usuario*, *Passageiro*, *Motorista*, *Veiculo*, *Corrida*, *Pagamento* e *Avaliacao*.

Cada entidade foi estruturada com seus respectivos atributos e relacionamentos, garantindo coerência com os processos do negócio. Por exemplo, a classe *Corrida* mantém vínculos diretos com *Passageiro*, *Motorista* e *Rota*, além de conter informações de status, valores e horários. A classe *Pagamento* está associada à *Corrida*, representando tanto pagamentos via PIX quanto em dinheiro, ambos processados manualmente no aplicativo. Além disso, o modelo inclui entidades auxiliares

como Localizacao, PoliticaTarifa, Regiao e Money, usadas para padronizar endereços, políticas de preço e valores monetários.

A Figura 9 apresenta o diagrama de classes do domínio, permitindo visualizar de forma clara as relações entre as entidades e seus papéis dentro do fluxo geral da aplicação.

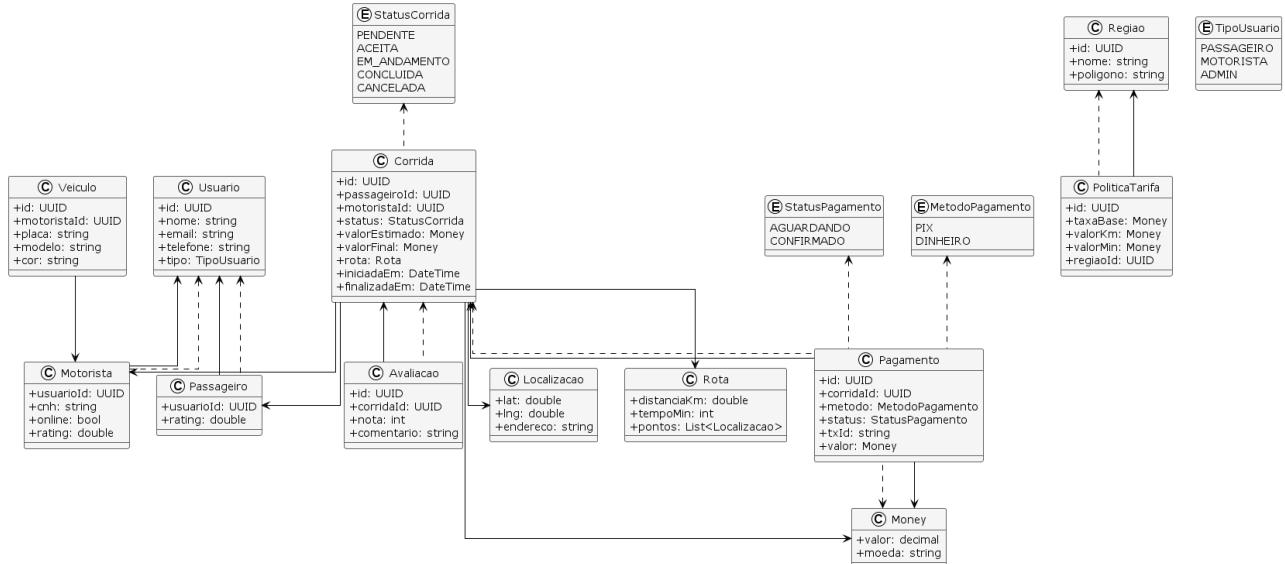


Figura 9 – Diagrama de Classes: Domain / Models

O diagrama de classes a seguir representa o pacote de *repositories*, responsável pela camada de persistência de dados. Essas classes são definidas como interfaces, pois a implementação pode variar de acordo com a tecnologia de banco de dados adotada, mantendo assim o princípio da abstração e separação de responsabilidades.

Cada repositório é responsável por manipular uma entidade específica, garantindo a persistência e recuperação de informações essenciais ao funcionamento do sistema. Entre os repositórios estão UsuarioRepository, CorridaRepository, PagamentoRepository e AvaliacaoRepository, que lidam com as operações de CRUD e consultas filtradas.

A Figura 10 exibe o diagrama de classes de *repositories*, demonstrando a estrutura modular de acesso aos dados e sua conexão com a camada de *services*.

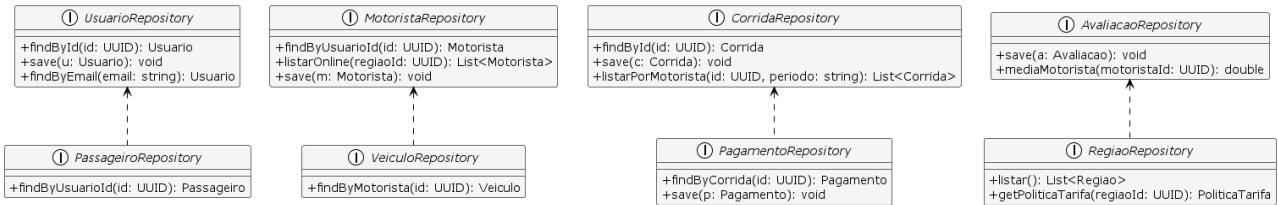


Figura 10 – Diagrama de Classes: Repositories

Por fim, a figura a seguir apresenta o diagrama de classes referente aos *Data Transfer Objects* (*DTOs*), que têm como função padronizar a comunicação entre o cliente (aplicativo móvel) e o servidor (API). Esses objetos simplificam o envio e recebimento de dados, evitando o tráfego de entidades completas e protegendo a estrutura interna do sistema.

Entre os principais *DTOs* estão *LoginRequest*, *PerfilResponse*, *SolicitarCorridaRequest*, *CorridaResponse*, *PagamentoRequest*, *PagamentoConfirmacaoRequest* e *AvaliacaoRequest*. Também foram definidos objetos voltados à geração de relatórios, como *RelatorioFiltro* e *RelatorioResultado*, utilizados por administradores e motoristas para visualizar métricas de desempenho.

A Figura 11 apresenta o diagrama de classes dos *DTOs*, mostrando a utilização de tipos de domínio como *Localizacao*, *Rota* e *Money* para garantir consistência entre os dados de transporte e pagamento.

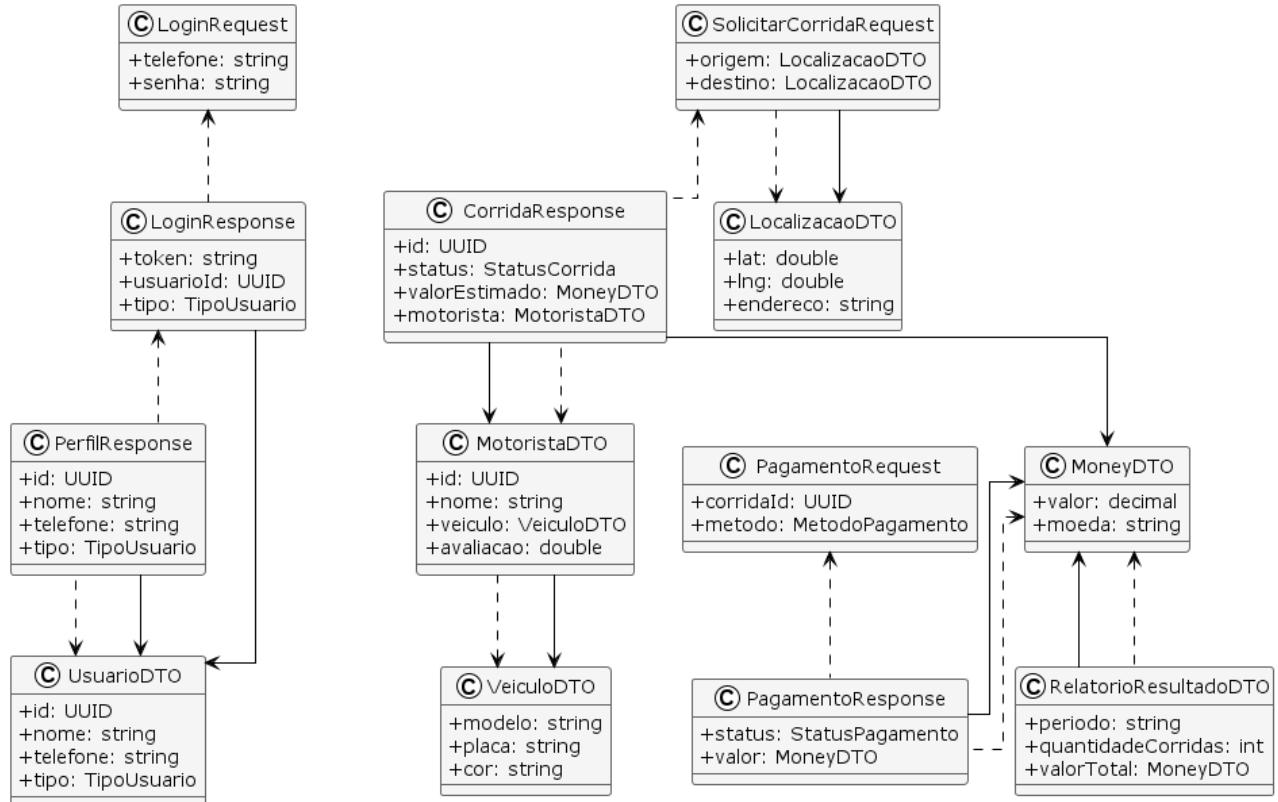


Figura 11 – Diagrama de Classes: DTOs

### 3.2 Diagramas de Sequência

Nesta seção, são apresentados os diagramas de sequência que modelam os fluxos principais do sistema de transporte proposto. Esses diagramas têm como objetivo mapear os caminhos críticos de uso da aplicação, evidenciando quem se comunica com quem, quais mensagens são trocadas e em que ordem.

Foram selecionados quatro fluxos representativos: solicitação de corrida, pagamento (PIX/dinheiro, confirmação manual), avaliação do motorista e relatórios administrativos. Juntos, eles cobrem as interações essenciais entre Passageiro, Motorista e Administrador, além das integrações necessárias com serviços externos.

A Figura 12 representa o fluxo de solicitação de corrida. O Passageiro informa origem e destino; o sistema consulta o serviço de mapas para obter rota, distância e tempo estimado; em seguida, identifica o motorista mais próximo e envia uma notificação de corrida. O motorista pode aceitar ou recusar. Em caso de recusa, a solicitação é reencaminhada a outro motorista. Este fluxo está relacionado aos casos de uso UC03 (Informar origem/destino), UC04 (Estimar tarifa), UC05 (Solicitar corrida) e UC11 (Receber solicitação).

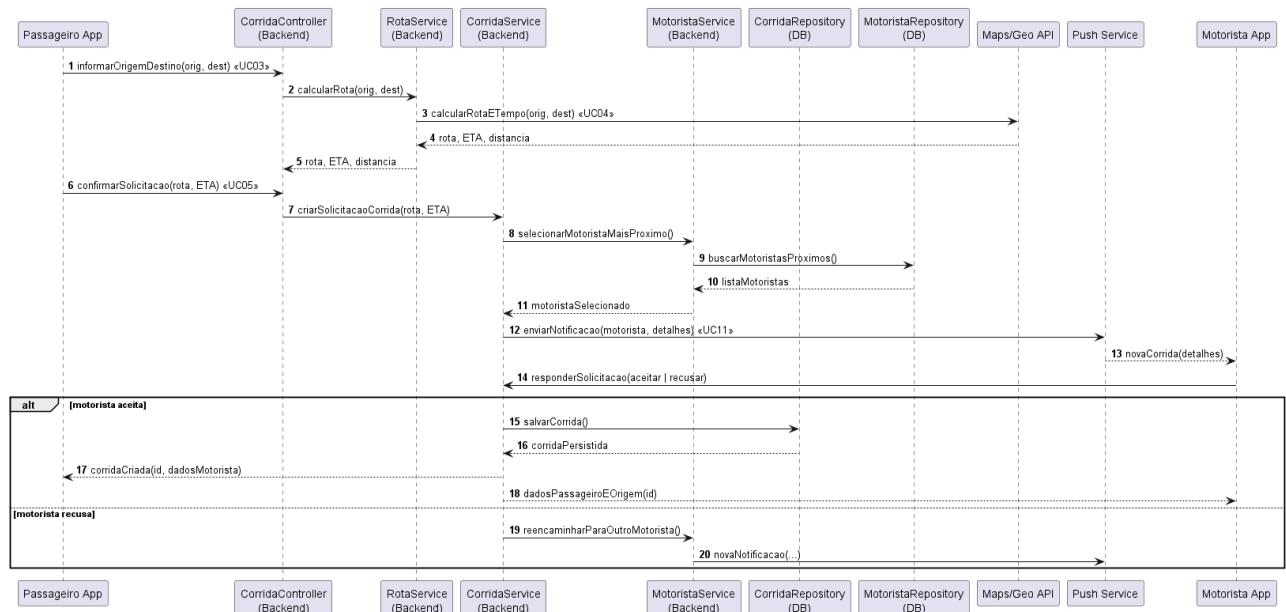


Figura 12 – Diagrama de Sequência Solicitação de Corrida

A Figura 13 descreve o encerramento da corrida e o pagamento. O sistema apresenta os métodos PIX e Dinheiro. Para PIX, o *backend* gera um *QR Code* (sem integração bancária automática); o motorista verifica manualmente no app do banco se o valor entrou e confirma no sistema. Para Dinheiro, o motorista confirma manualmente o recebimento. Em ambos os casos, a corrida passa a Finalizada, e recibos são disponibilizados. Este fluxo cobre UC08 (Efetuar pagamento) e UC14 (Finalizar corrida).

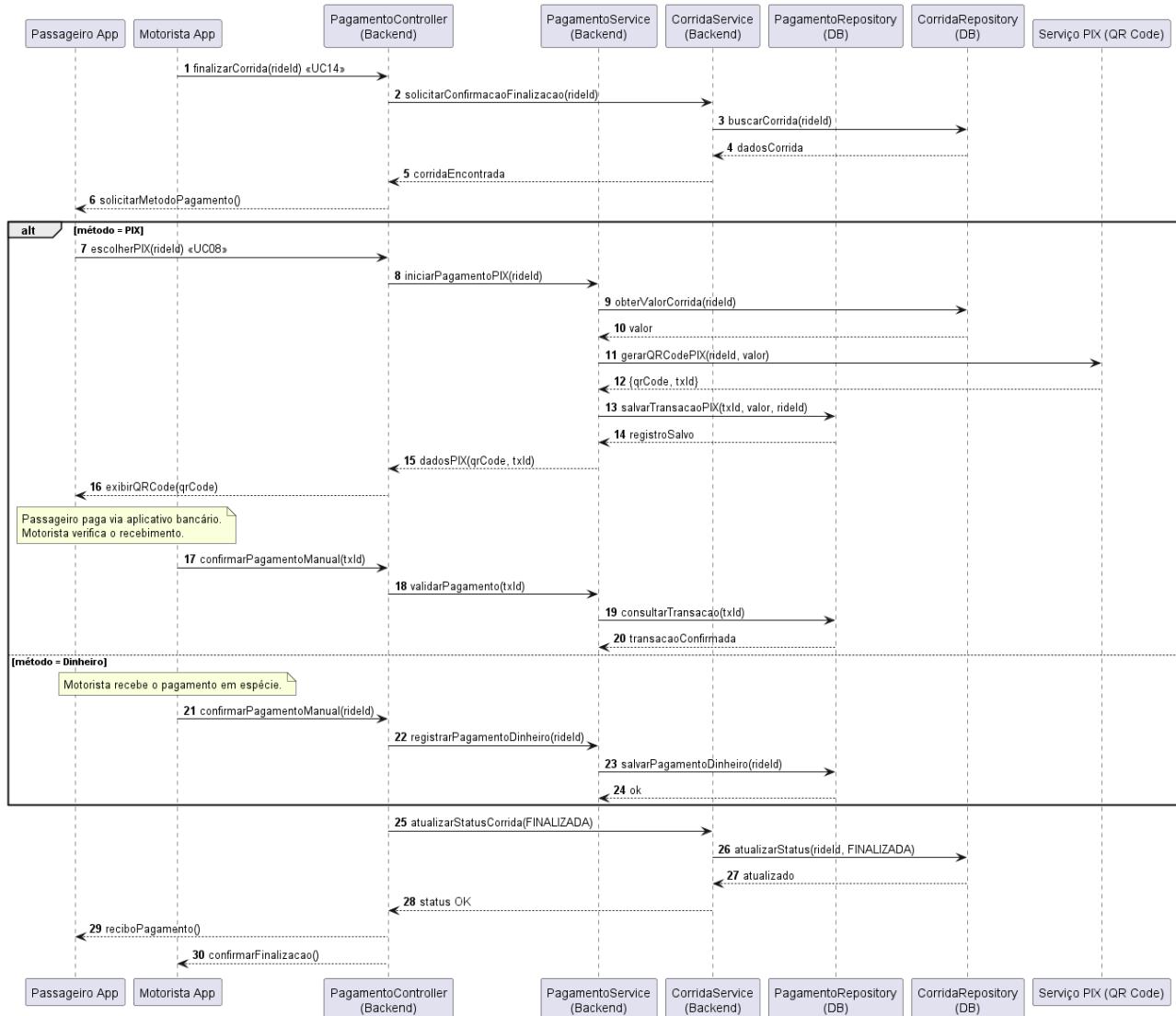


Figura 13 – Diagrama de Sequência Encerramento da Corrida e Pagamento

A Figura 14 mostra o fluxo de avaliação do motorista pelo Passageiro após o pagamento. O cliente informa nota e comentário; o sistema armazena a avaliação e recalcula a média do motorista. Esse processo retroalimenta a qualidade do serviço e influencia futuras seleções de motoristas. Relaciona-se ao caso de uso UC09 (Avaliar motorista).

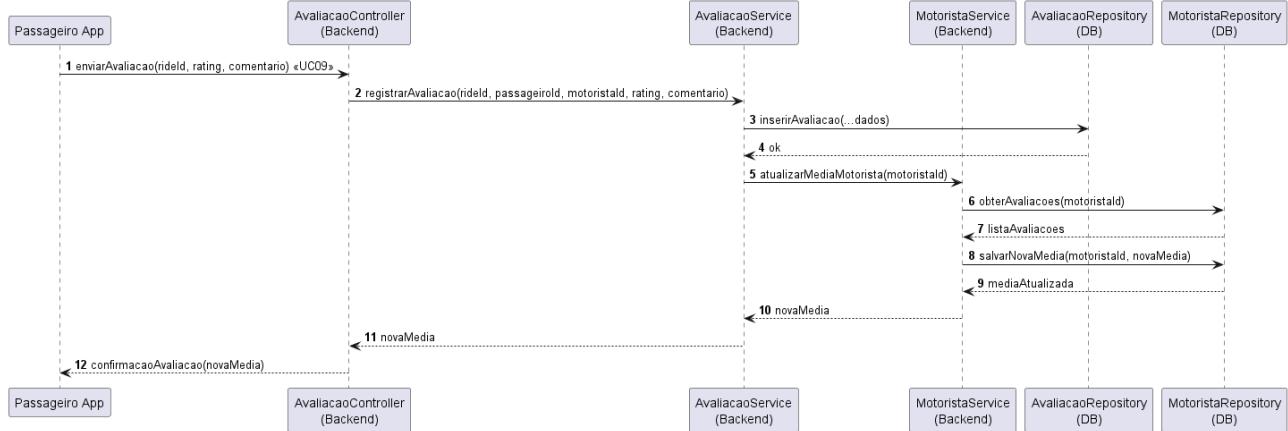


Figura 14 – Diagrama de Sequência Avaliação do Motorista

A Figura 15 ilustra o painel administrativo. O Administrador realiza login, solicita relatórios (por período, região, motorista etc.), e o *backend* consulta o repositório/warehouse para agregar dados de corridas e pagamentos. Opcionalmente, o Administrador pode ajustar configurações (taxas, regiões), que são persistidas para uso nos cálculos. Relaciona-se aos casos de uso UC16 (Gerenciar usuários/motoristas), UC17 (Ver relatórios e métricas) e UC18 (Configurar taxas/regiões).

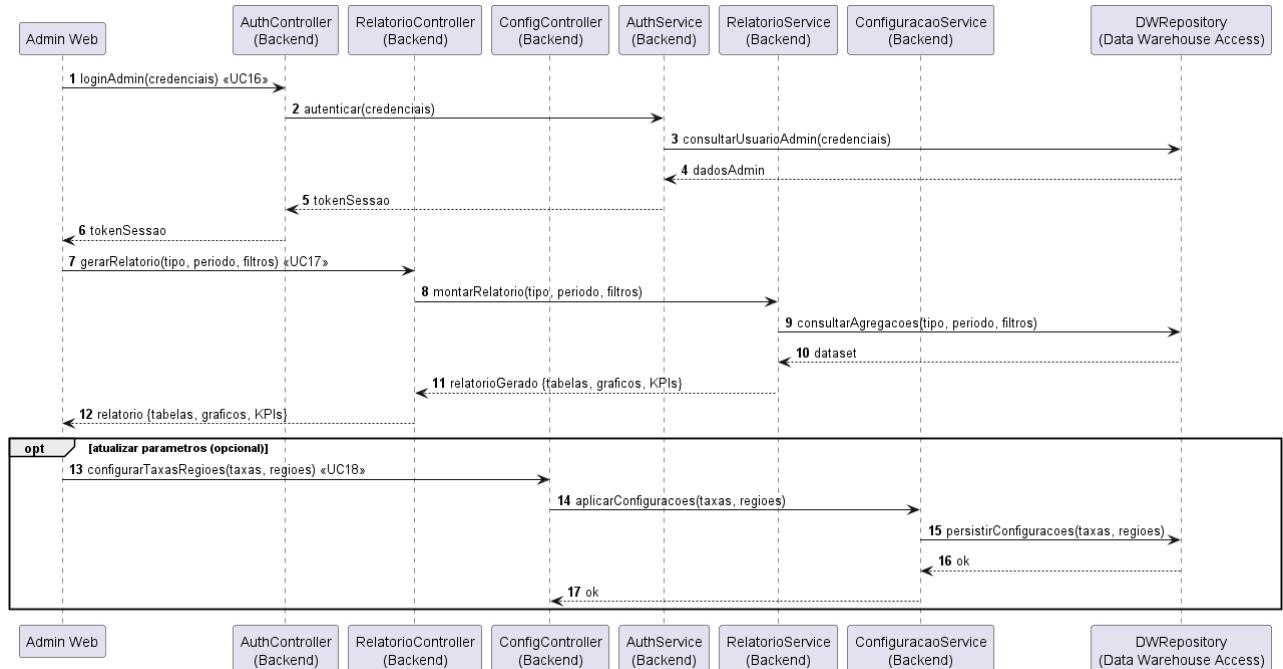


Figura 15 – Diagrama de Sequência Painel Administrativo

### 3.3 Diagramas de Comunicação

Nesta seção, são apresentados os diagramas de comunicação que modelam as trocas de mensagens entre atores, camadas e serviços do sistema de transporte. O objetivo é documentar quem se

comunica com quem e em que ordem, cobrindo os principais fluxos: autenticação, solicitação e despacho de corrida, atualização de *status*, pagamento e relatórios administrativos. Os diagramas a seguir complementam os diagramas de sequência, servindo como registro das interfaces e contratos de mensagem utilizados na aplicação.

A Figura 16 apresenta as trocas de mensagens para autenticação de usuários (passageiros e motoristas). O aplicativo envia as credenciais para a *API*, que delega a validação ao *AuthService*. Em seguida, os dados do usuário são obtidos no *UsuarioRepository* e um *token* de sessão é retornado ao cliente. Esse fluxo garante a segurança de acesso e inicializa o contexto de sessão para as demais operações do sistema.

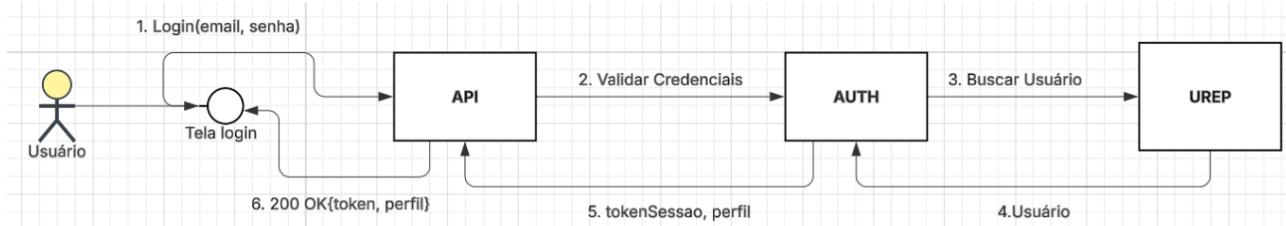


Figura 16 – Diagrama de Comunicação Autenticar Usuário

A Figura 17 descreve o fluxo de comunicação para solicitação de corrida. Após informar origem/destino, a *API* consulta a *Maps/Geo API* para estimativa de rota/tempo, chama o *MatchingService* para selecionar o motorista disponível mais próximo e utiliza o *PushService* para notificar o motorista. A resposta do motorista (aceite/recusa) retorna à *API*, que confirma ao passageiro e publica o estado inicial da corrida.

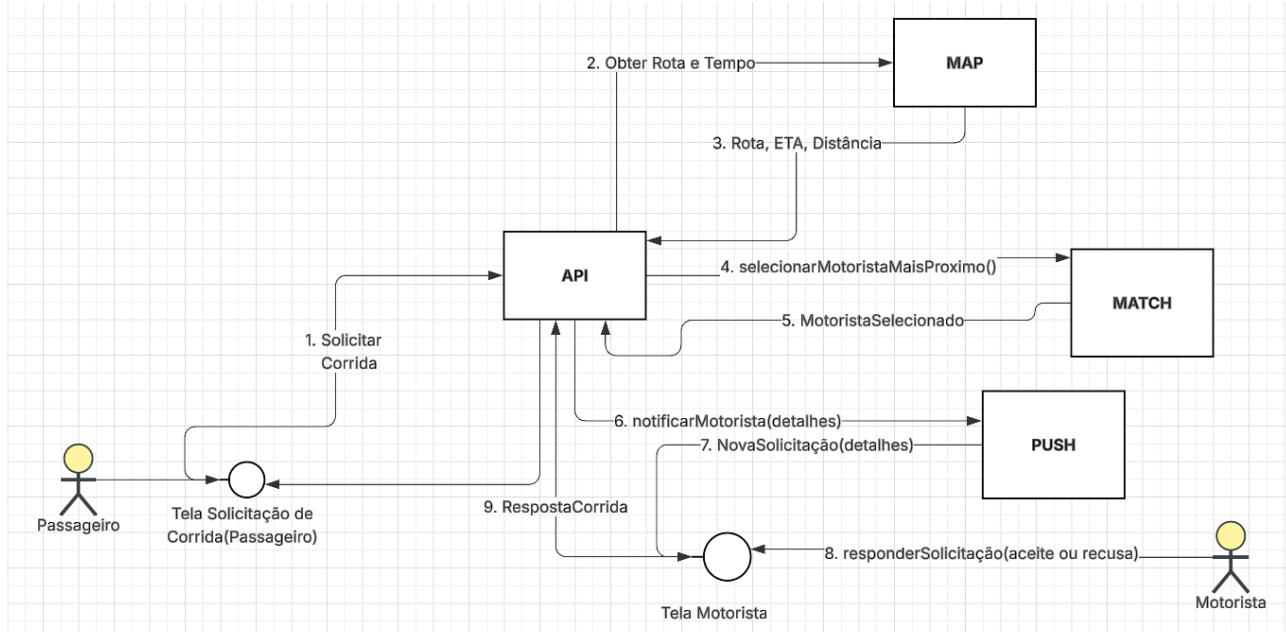


Figura 17 – Diagrama de Comunicação Solicitar Corrida e Despachar Motorista

A Figura 18 mostra as mensagens trocadas durante a execução da corrida. O motorista envia eventos de *status* (a caminhou, chegou, iniciou, finalizou) à *API*. A *API* persiste no *CorridaRepository*, recalcula *ETA* quando necessário via *Maps/Geo API* e notifica passageiro/motorista com atualizações relevantes. Esse fluxo mantém ambas as partes informadas e cria o histórico operacional da corrida.

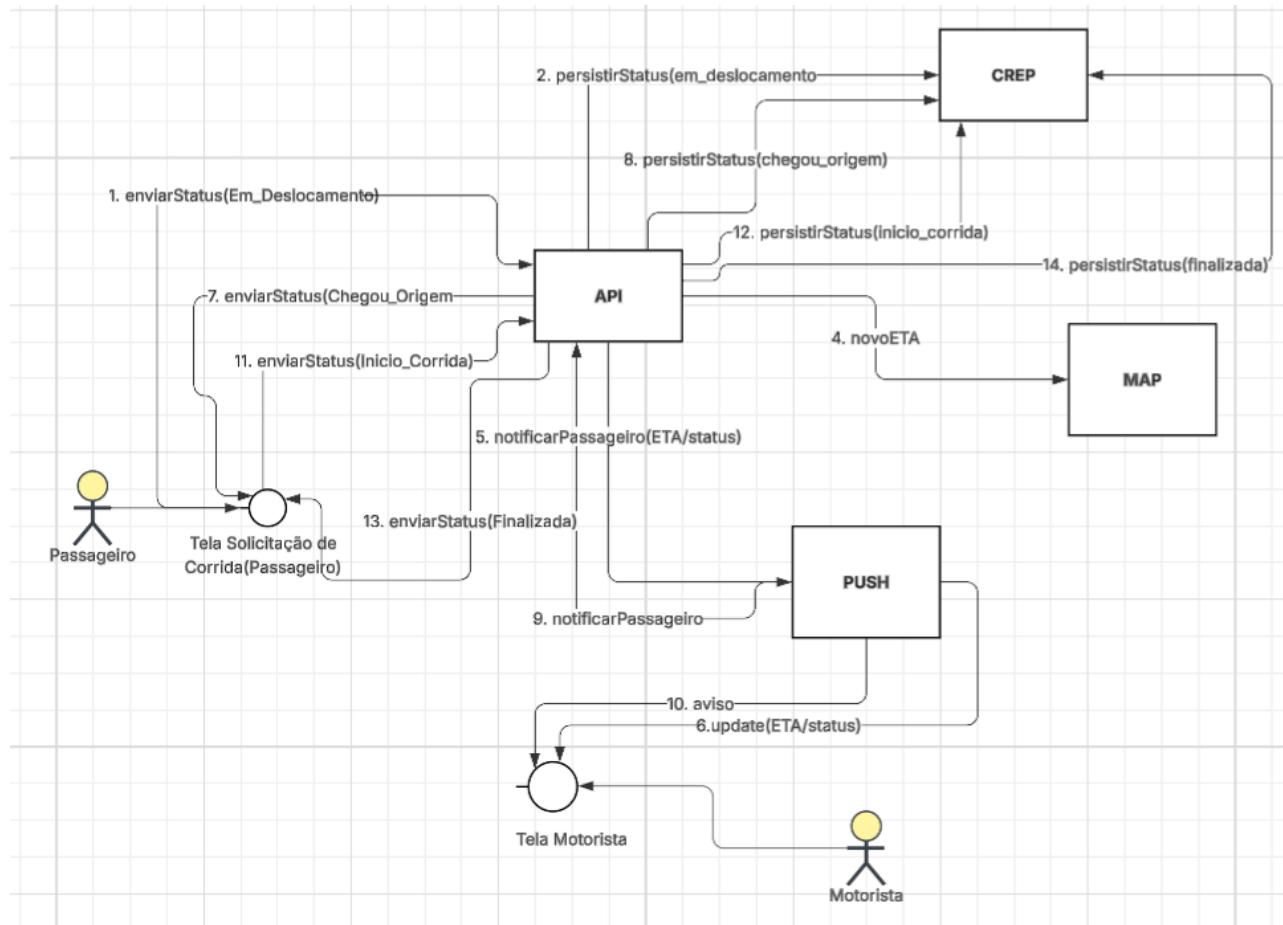


Figura 18 – Diagrama de Comunicação Atualizar Status da Corrida e Notificar Partes

A Figura 19 detalha as comunicações no pagamento. Ao finalizar a corrida, o passageiro escolhe PIX ou dinheiro. Para PIX, a API solicita ao *PixQrService* a geração do *QR Code* e o exibe no app; o motorista confere manualmente no seu banco e confirma na *API*. Para dinheiro, o motorista apenas confirma o recebimento. Em ambos os casos, a *API* atualiza a Corrida para “Finalizada” e emite recibos.

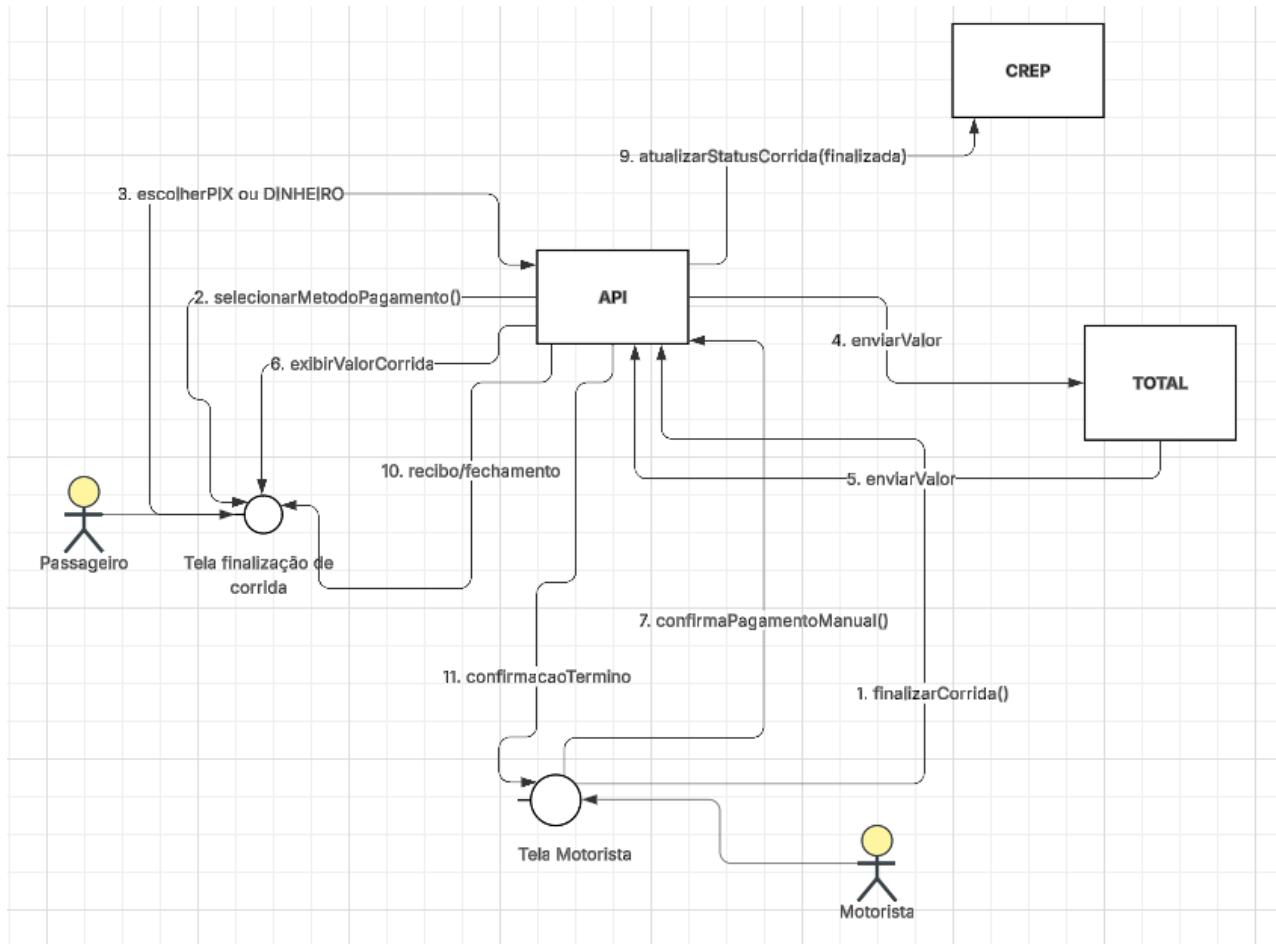


Figura 19 – Diagrama de Comunicação Efetuar Pagamento

### 3.4 Arquitetura

A Figura 21 representa o diagrama de pacotes, também conhecido como diagrama de arquitetura, do sistema MobU. Nele é possível observar a organização estrutural dos principais módulos da aplicação, representando a separação de responsabilidades entre as camadas que compõem o sistema.

O primeiro pacote representa a aplicação mobile, que contém a Interface de Usuário (UI, do inglês *User Interface*), responsável pela interação direta com os diferentes perfis de usuários — passageiro, motorista e administrador. Essa camada é responsável por exibir as informações e capturar as ações realizadas pelos usuários durante o uso do aplicativo.

A aplicação mobile se comunica com a MobU API, que corresponde à camada intermediária do sistema e é responsável por processar as regras de negócio e gerenciar as comunicações com os serviços externos.

Dentro desta camada, estão definidos os pacotes Controllers, Services, Jobs e Utils, os quais implementam as funcionalidades principais da aplicação.

- O pacote Controllers contém os pontos de entrada da API e é responsável por intermediar as requisições entre o cliente e o backend.
- O pacote Services contém a lógica de negócio central do sistema, como o gerenciamento de corridas, pagamentos e autenticações.
- O pacote Jobs trata de tarefas assíncronas e agendadas, como envio de notificações ou processamento de relatórios.
- O pacote Utils contém funções auxiliares reutilizáveis em diferentes partes da aplicação.
- O pacote Libs reúne bibliotecas e adaptadores compartilhados entre serviços.

Por fim, a camada Data é responsável pelo acesso e persistência dos dados, representando a comunicação com o banco de dados e os modelos de entidades da aplicação. Ela contém o pacote Models, que define as estruturas de dados principais do sistema, como *Usuário*, *Corrida*, *Pagamento* e *Avaliação*.

Essa arquitetura segue o padrão camada UI → API (business) → Data, proporcionando alta coesão e baixo acoplamento entre os módulos, o que facilita a manutenção, a escalabilidade e futuras expansões da aplicação.

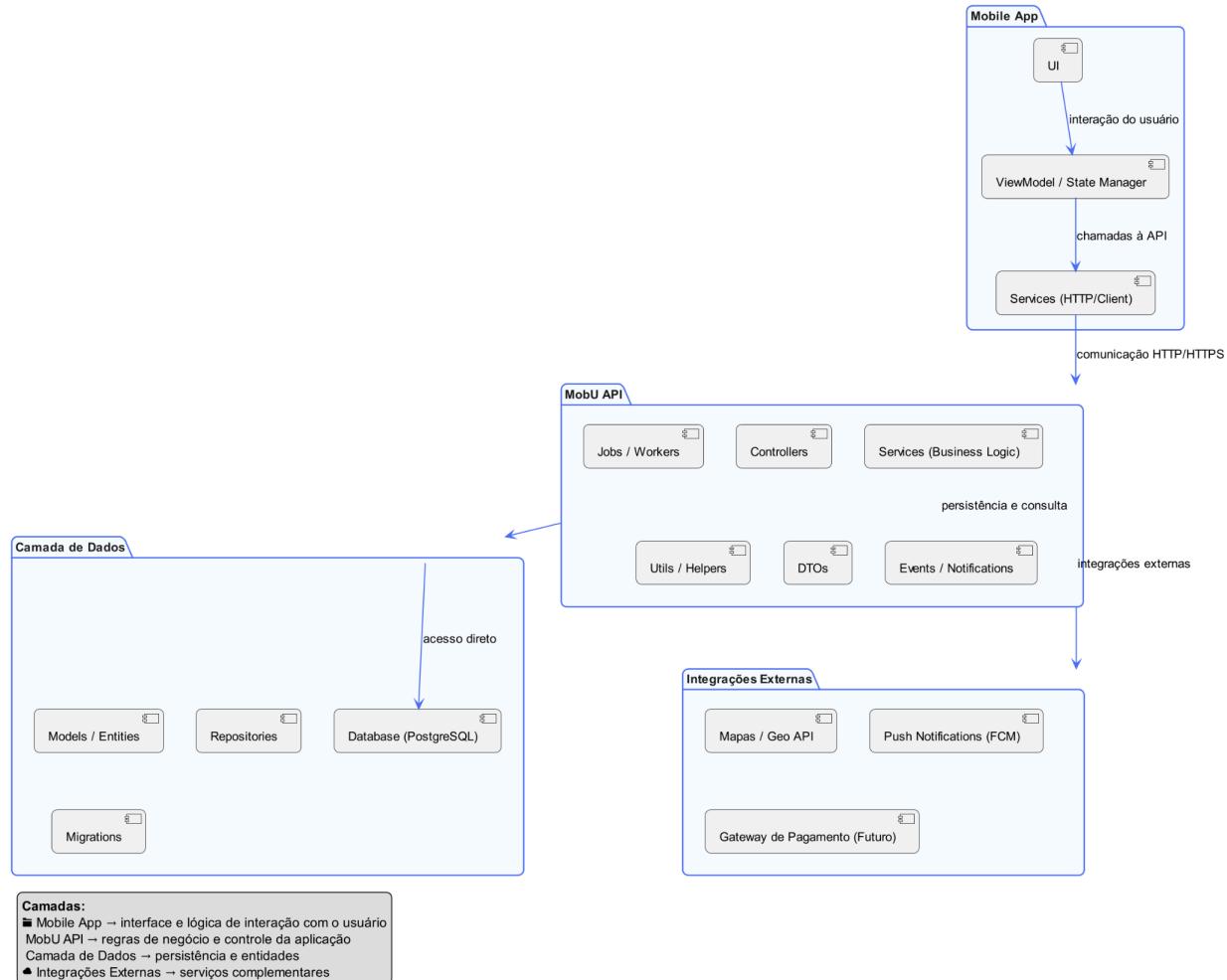


Figura 20 – Diagrama de Pacotes

### 3.5 Diagramas de Estados

O diagrama representado pela Figura 22 contém a lógica de mudança de estados pelos quais uma corrida passa no sistema MobU. É possível observar que a corrida percorre uma sequência de estados que refletem as diferentes fases do processo de transporte, desde sua solicitação até a finalização ou cancelamento.

Inicialmente, o estado Solicitada representa a criação da corrida pelo passageiro. Em seguida, a corrida pode ser Despachada, momento em que o sistema tenta associar um motorista disponível à solicitação. A partir desse ponto, o motorista pode aceitar ou recusar a corrida, resultando respectivamente nos estados Aceita ou Recusada — neste último caso, o sistema pode reencaminhar a solicitação para outro motorista disponível.

Quando aceita, a corrida avança para o estado A Caminho da Origem, no qual o motorista se dirige até o ponto de embarque do passageiro. Ao chegar, a corrida transita para o estado Chegou à Origem, e, após o embarque, é iniciada, entrando no estado Em Andamento.

Por fim, o estado Finalizada indica a conclusão da corrida, com o registro das informações de trajeto e valor.

O diagrama também contempla transições de cancelamento, que podem ocorrer em diferentes fases do processo, seja por parte do passageiro, do motorista ou por algum motivo forçado pelo sistema. Essas transições levam a corrida ao estado Cancelada, que, assim como o estado final de Finalizada, encerra o ciclo de vida da corrida dentro do sistema.

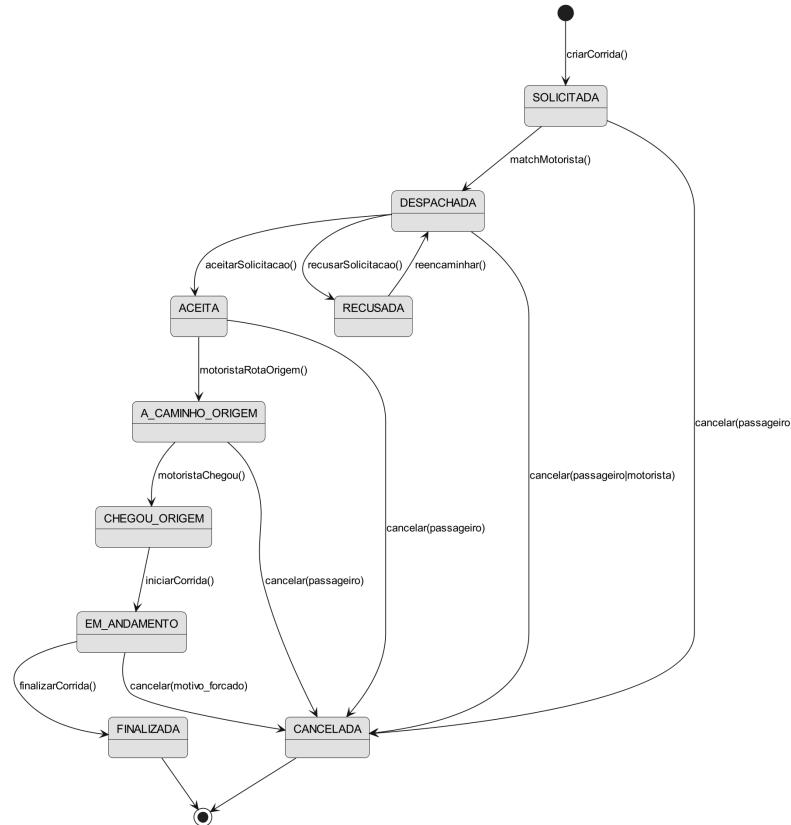


Figura 21 – Diagrama de Estados da Corrida

A Figura 23 representa o Diagrama de Estados do Pagamento no sistema MobU. Esse diagrama descreve o ciclo de vida de um pagamento associado a uma corrida, desde o momento em que a corrida é finalizada até a confirmação ou cancelamento da transação.

O processo se inicia no estado Aguardando, quando a corrida é concluída e o usuário deve escolher o método de pagamento. A partir desse ponto, existem dois fluxos possíveis:

1. Caso o usuário escolha PIX, o sistema gera um QR Code, e o pagamento entra no estado PIX Gerado.
2. Caso o usuário opte por dinheiro, o pagamento segue para o estado Aguardando Confirmação do Motorista, onde o motorista deve confirmar o recebimento.

A partir do estado PIX Gerado, o pagamento pode ser Confirmado após a validação do txId ou pode Expirar caso o QR Code não seja utilizado dentro do prazo estabelecido.

Além disso, o pagamento pode ser Cancelado manualmente em qualquer um desses estados intermediários.

Por fim, os estados Confirmado, Expirado e Cancelado representam os estados finais do processo de pagamento, encerrando o ciclo de transações para aquela corrida específica.

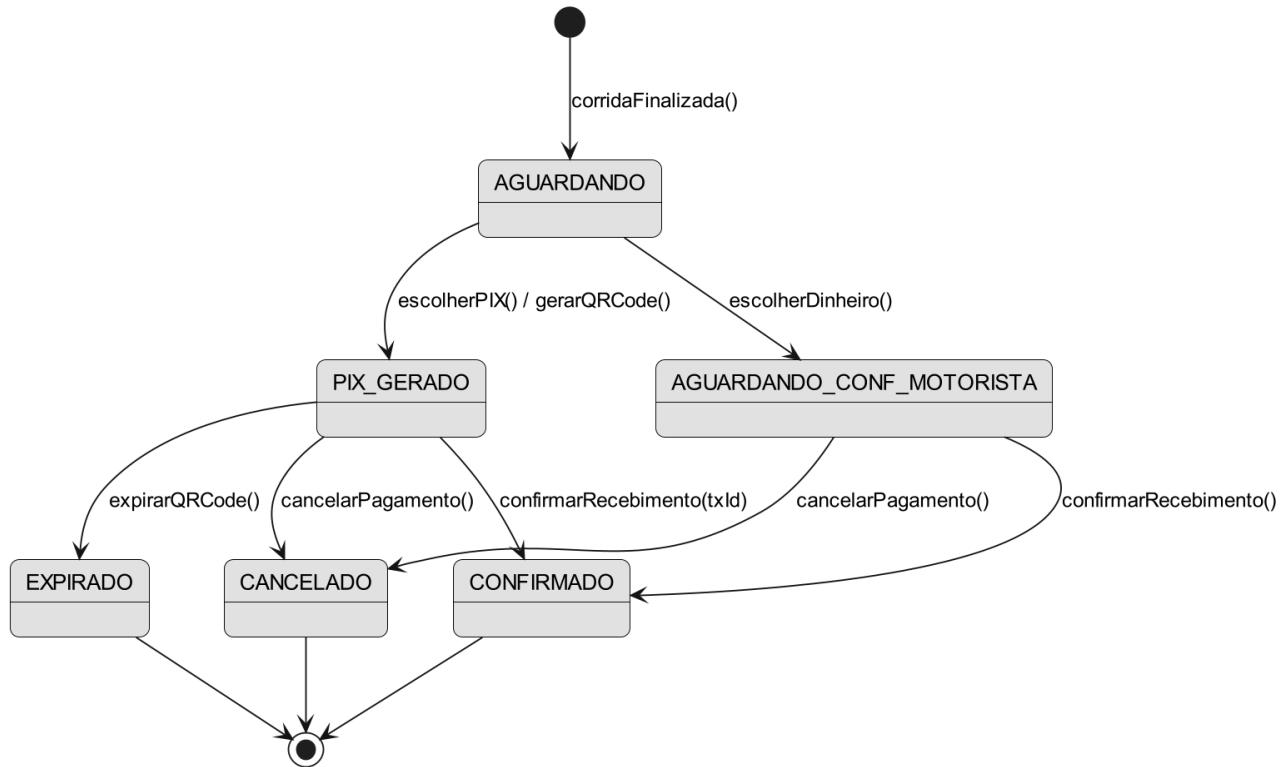


Figura 22 – Diagrama de Estados do Pagamento

### 3.6 Diagrama de Componentes e Implantação.

O diagrama representado pela Figura 24 mostra a organização dos principais componentes que compõem a arquitetura lógica do sistema *MobU*. Nele é possível observar dois grandes módulos: o Mobile App e a *MobU API*, além de seus elementos de integração com serviços externos e infraestrutura de persistência de dados.

O Mobile App representa a aplicação móvel executada em dispositivos Android e iOS, composta pela camada de Interface de Usuário (UI), responsável pelas páginas, componentes visuais e estados da aplicação. Essa camada se comunica com a *MobU API* por meio de requisições HTTP/HTTPS, realizadas através do módulo Services, que encapsula as funções de acesso à API e controle de fluxo de dados entre o cliente e o servidor.

A *MobU API* representa o núcleo de processamento do sistema, responsável por tratar as regras de negócio e integrar as diferentes partes da aplicação.

Dentro dessa camada, encontram-se pacotes especializados:

- Controllers, que atuam como pontos de entrada das requisições;
- Services, que implementam a lógica de negócio principal;
- Jobs, responsáveis por tarefas assíncronas e agendadas;
- Utils e Libs, que contêm funções e bibliotecas auxiliares;
- Models, que representam as entidades e dados do sistema;
- APIs, dedicadas às integrações externas, e Events, que tratam notificações e mensagens internas.

O Message Queue representa o serviço de mensageria responsável por filas de eventos, enquanto o SGBD (PostgreSQL) é responsável pelo armazenamento persistente de informações. Além disso, o sistema se integra a APIs externas, como Mapas/Geo, Push Notifications (FCM) e o Gateway de Pagamento (futuro), garantindo comunicação com serviços complementares e expansibilidade futura.

Essa estrutura modular facilita a manutenção, o escalonamento e a reutilização de componentes, promovendo um acoplamento reduzido e alta coesão entre as camadas.

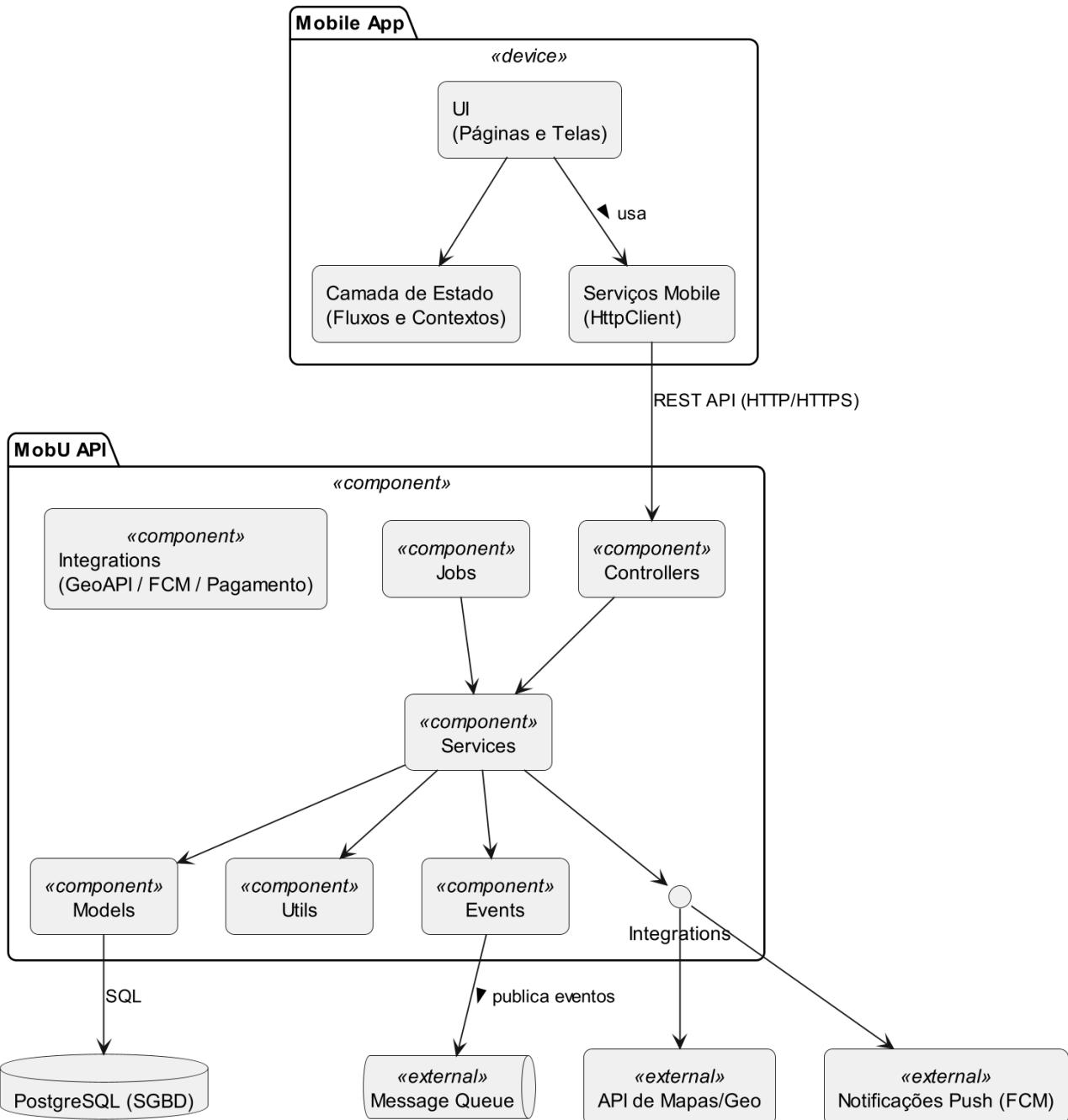


Figura 23 – Diagrama de Componentes da Aplicação

A Figura 25 representa o diagrama de implantação da aplicação *MobU*, evidenciando a distribuição dos principais módulos do sistema em diferentes nós de execução. O diagrama demonstra como o sistema é hospedado, os componentes que o compõem e as comunicações realizadas entre eles.

O nó Dispositivo Android/iOS representa o ambiente de execução do aplicativo móvel, que contém a UI e os serviços de interação do usuário. Ele realiza comunicação com o Servidor de

Comunicação, responsável pelo balanceamento de carga, controle de requisições e redirecionamento de tráfego para o servidor principal da aplicação.

O Servidor de Aplicação hospeda o container que executa a MobU API, onde residem os módulos de controle, serviços, e regras de negócio.

Essa camada central realiza integrações com APIs externas, incluindo Mapas/Geo, Push Notifications (FCM) e o PSP de Pagamentos (futuro), permitindo expansão modular da solução.

O Servidor de Comunicação atua como intermediário entre o aplicativo e os demais serviços da infraestrutura, garantindo desempenho e disponibilidade. O nó Mensageria (como *AWS SQS*, *RabbitMQ* ou *Kafka*) é responsável pelo tratamento de eventos e notificações assíncronas entre os módulos, assegurando confiabilidade na entrega de mensagens.

Por fim, o SGBD (PostgreSQL) representa o banco de dados responsável pela persistência e integridade dos dados do sistema, armazenando informações de usuários, corridas, pagamentos e métricas.

A comunicação entre os nós ocorre de forma segura, utilizando protocolos HTTPS, AMQP/SQS e SQL/TLS, garantindo confidencialidade e consistência.

Essa topologia de implantação possibilita à aplicação *MobU* alcançar alta disponibilidade, escalabilidade horizontal e resiliência, sendo adequada tanto para ambientes de desenvolvimento quanto de produção em nuvem (AWS, GCP ou Azure).

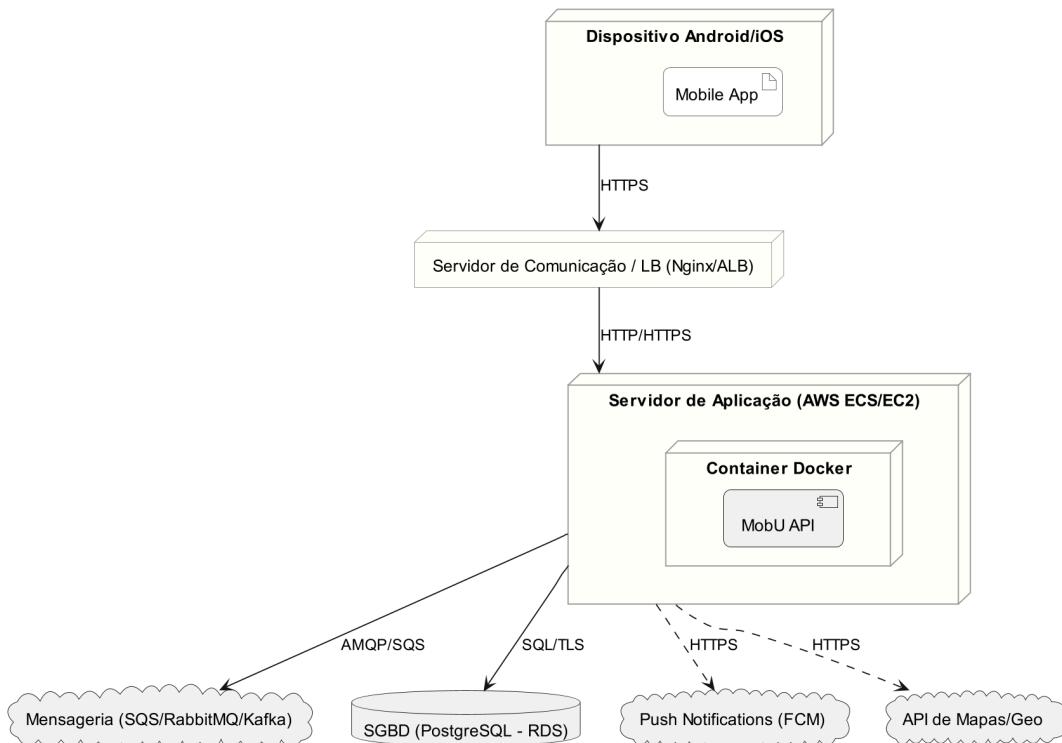


Figura 24 – Diagrama de Implantação da Aplicação

## 4. Projeto de Interface com Usuário

Esta seção tem como objetivo apresentar e descrever as interfaces de interação com o usuário que compõem o sistema MobU. Para isso, foi elaborado um protótipo de baixa fidelidade, desenvolvido com o intuito de representar a estrutura visual e o fluxo principal de navegação da aplicação. As telas foram construídas de forma simplificada, com foco na organização dos elementos e na disposição das funcionalidades, possibilitando a validação inicial da proposta de interface antes da etapa de design detalhado. Dessa forma, as interfaces aqui apresentadas estão relacionadas aos casos de uso descritos na Seção 2.3.1, servindo como base para o mapeamento das funcionalidades essenciais que atenderão aos requisitos funcionais e não funcionais do sistema.

O objetivo deste protótipo é demonstrar a interação prevista entre os usuários e o sistema, permitindo visualizar o comportamento esperado das principais telas e apoiar o planejamento da fase de implementação.

### 4.1 Esboço das Interfaces Comuns a Todos os Atores

Wireframe/mockup/storyboard das interfaces que são comuns a todos os atores do sistema.

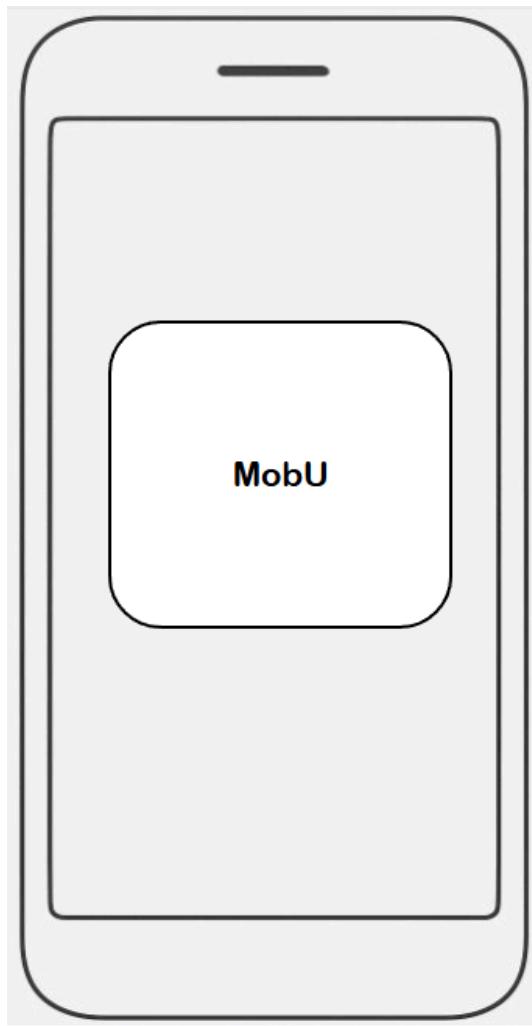


Figura 25. Página Splash Screen.



Figura 26. Página de login/cadastro.

A Figura 21 representa a tela de Splash Screen do aplicativo do passageiro. Esta página é exibida brevemente ao abrir o app e enquanto o sistema carrega dados de sessão ou verifica se o usuário já está autenticado. Caso a autenticação seja válida, a aplicação redireciona automaticamente o usuário para a página principal de solicitação de corrida (Figura 33). A Figura 22 mostra a tela de login/cadastro do usuário.

#### 4.2 Esboço das Interfaces Usadas pelo Administrador

Wireframe/mockup/storyboard das interfaces exclusivas do Administrador.

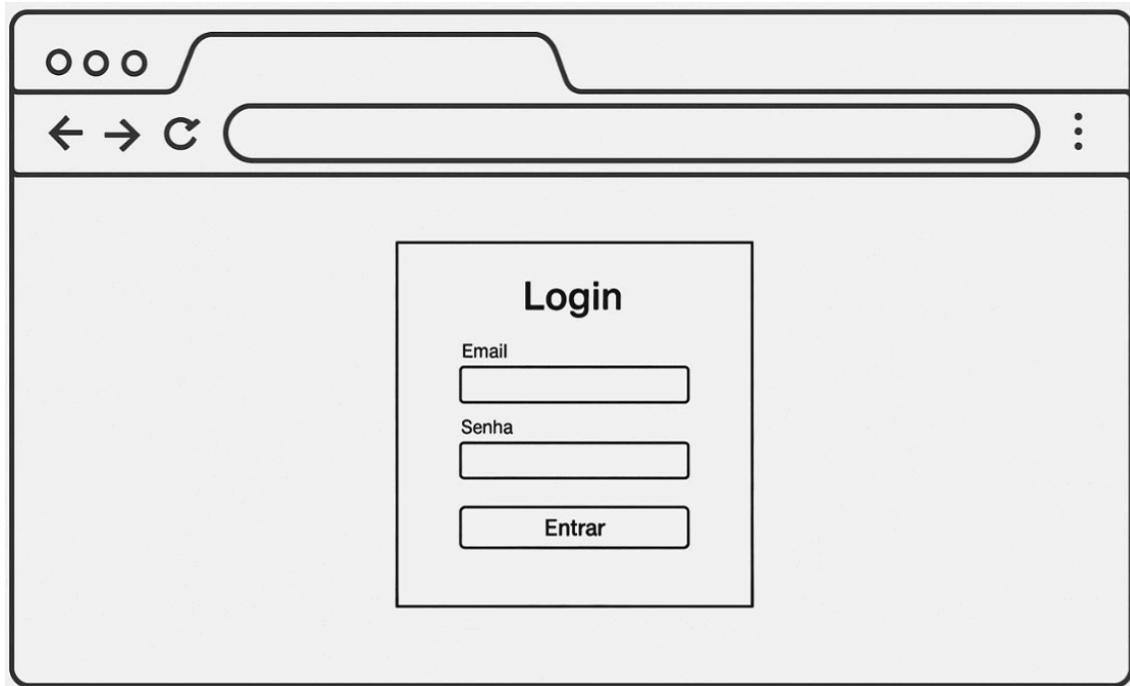


Figura 27. Página de login do administrador.

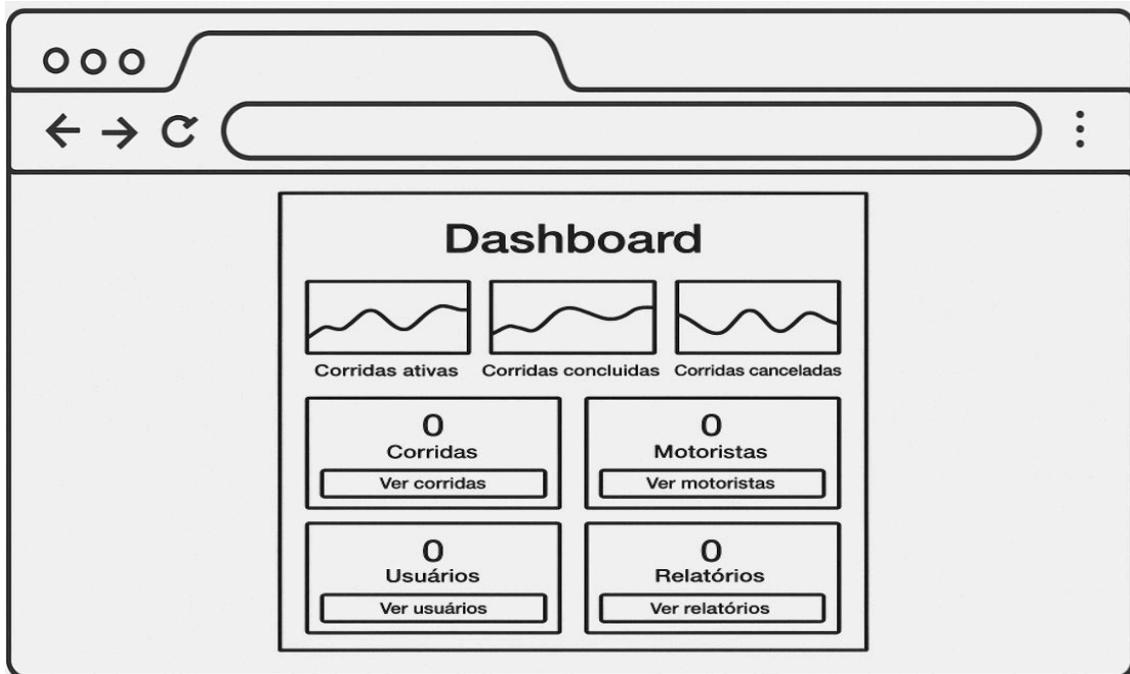


Figura 28. Página do dashboard principal.

A Figura 23 representa a tela de login do administrador, que dá acesso à tela de dashboard principal (Figura 24). O dashboard contém gráficos de corridas ativas, concluídas e canceladas, além de atalhos para funcionalidades administrativas.

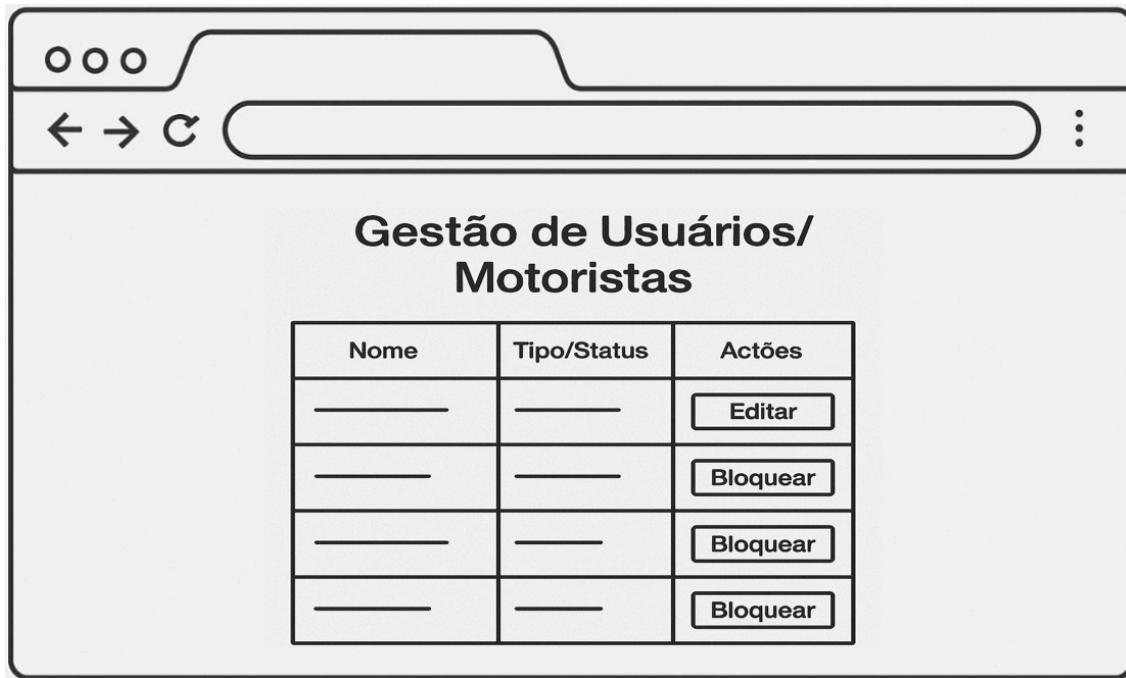


Figura 29. Página da gestão de usuários.

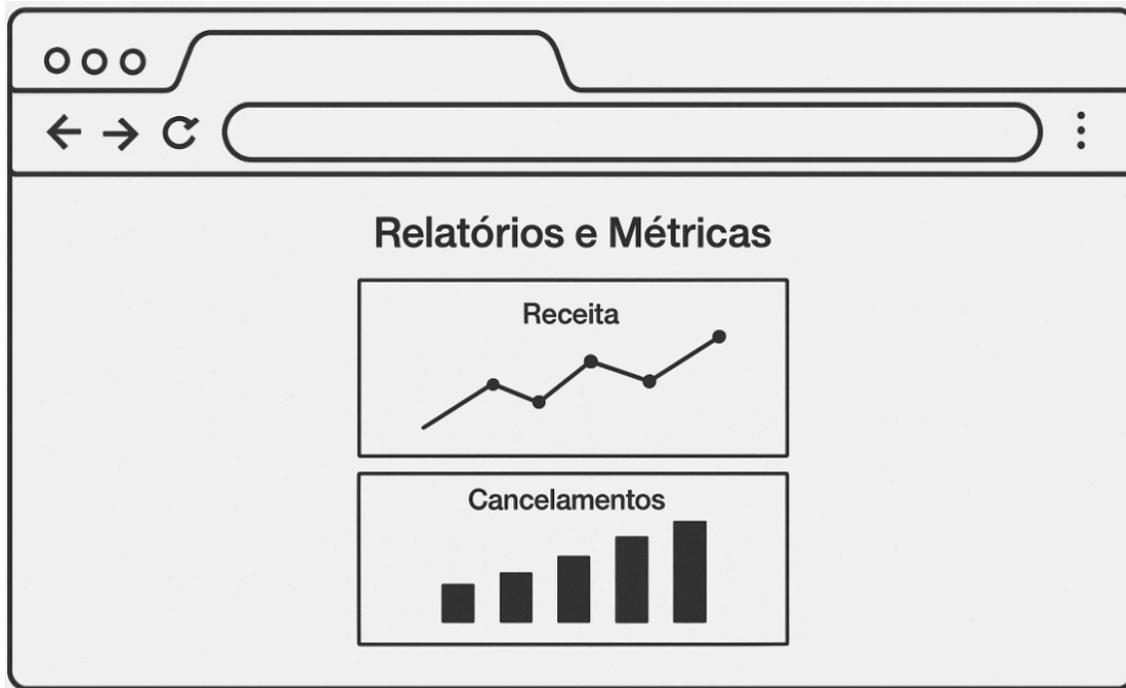


Figura 30. Página de relatórios e métricas.

A Figura 25 apresenta a tela de gestão de usuários e motoristas, com tabela listando os registros, status e botões para editar ou bloquear perfis. A Figura 26 mostra a tela de relatórios e métricas, que contém gráficos sobre receitas, taxas de cancelamento e desempenho do sistema.

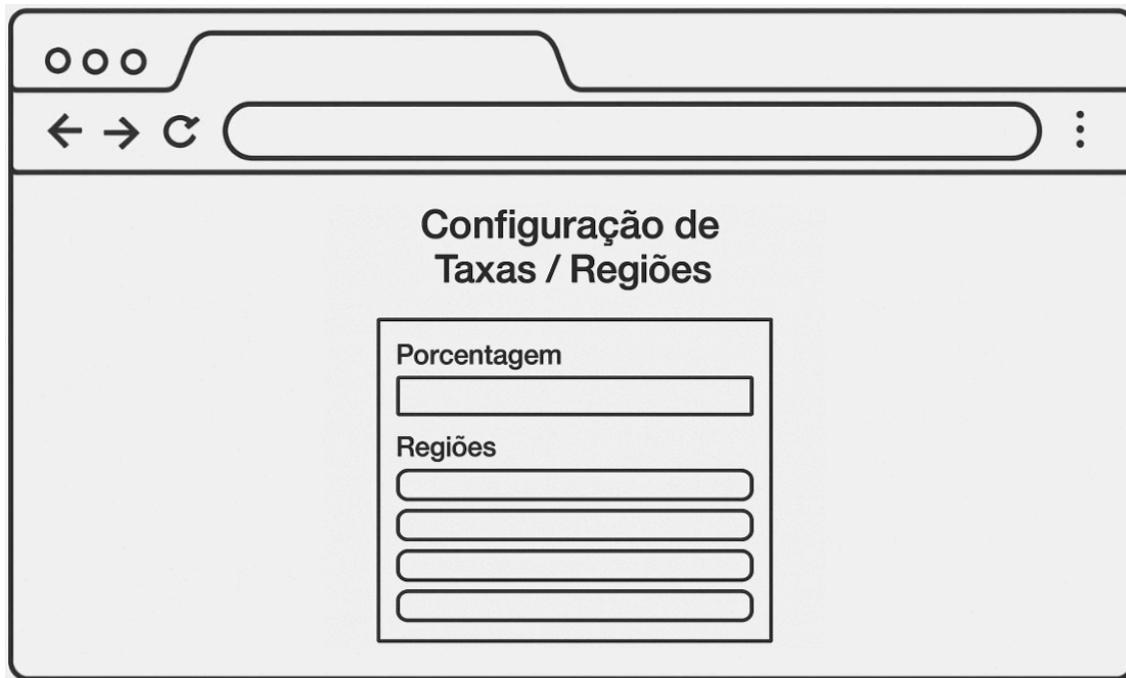


Figura 31. Página de configuração de taxas e regiões.

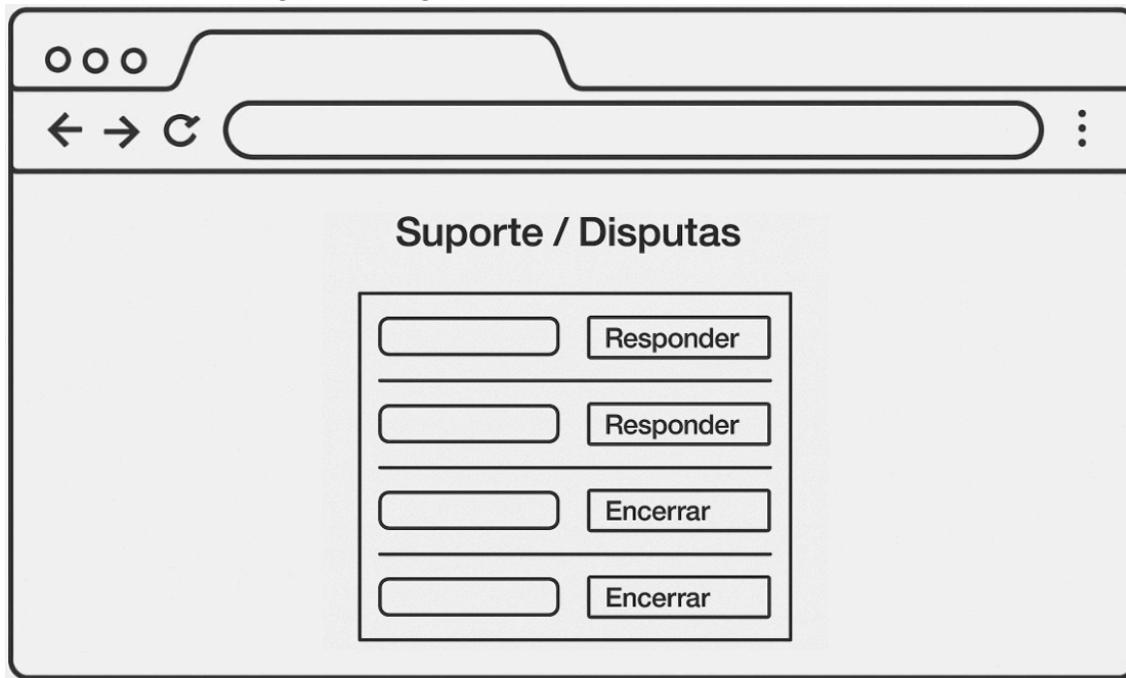


Figura 32. Página de suporte e disputas.

A Figura 27 apresenta a tela de configuração de taxas e regiões, na qual o administrador define percentuais de comissão e gerencia as áreas atendidas pela plataforma. Por fim, a Figura 28 exibe a tela de suporte e disputas, que lista chamados abertos por usuários ou motoristas e permite ao administrador registrar respostas ou encerrar cada caso.

#### 4.3 Esboço das Interfaces Usadas pelo Motorista

Wireframe/mockup/storyboard das interfaces exclusivas do Motorista.



Figura 33. Página de status do motorista.

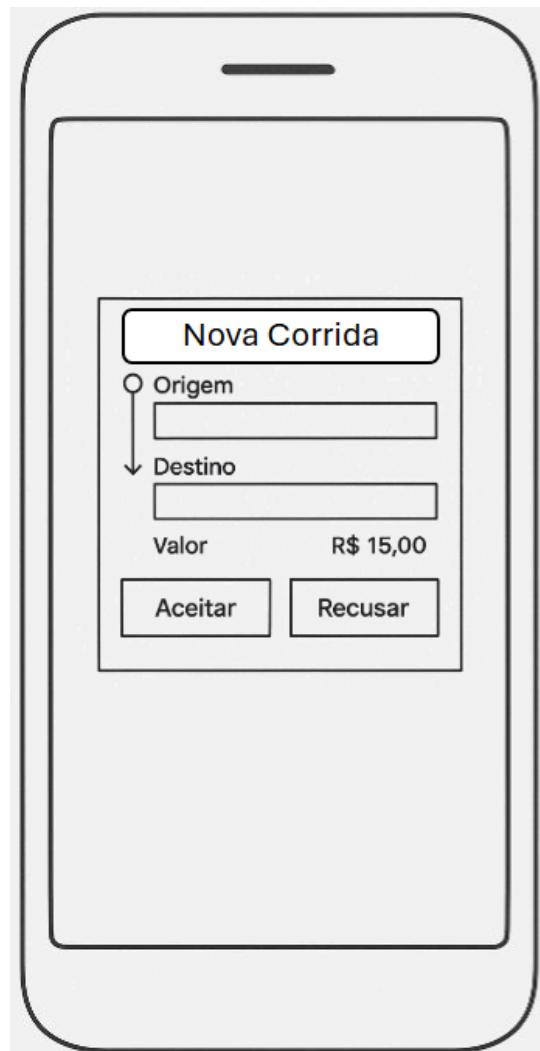


Figura 34. Página de nova corrida.

Após acessar a conta, o motorista é direcionado para a tela de status online/offline (Figura 29), que possui um botão central destacando o estado atual (verde para disponível, vermelho para indisponível). Quando o motorista está online e recebe uma nova corrida, é exibida a tela de solicitação (Figura 30), que contém os dados do passageiro, origem, destino e valor estimado, além dos botões “Aceitar” e “Recusar”.



Figura 35. Página de navegação do motorista.



Figura 36. Página relatório de ganhos.

Uma vez aceita a corrida, o app abre a tela de navegação (Figura 31), que mostra no mapa a rota até o local de embarque e, posteriormente, até o destino do passageiro. Nesta tela, o motorista também tem acesso aos botões “Cheguei”, “Iniciar viagem” e “Finalizar viagem” (Figura 31), cada um alterando o estado da corrida. Concluída a viagem, o motorista é redirecionado à tela de relatórios de ganhos (Figura 32), que apresenta dados de corridas finalizadas, ganhos do dia e gráficos semanais.

#### 4.4 Esboço das Interfaces Usadas pelo Passageiro

Wireframe/mockup/storyboard das interfaces exclusivas do Passageiro.



Figura 37. Página inicial do passageiro.



Figura 38. Página estimativa de corrida.

A Figura 33 representa a tela principal de solicitação de corrida. Nela o passageiro vê um mapa com a posição atual, campo para inserir o destino e botão para solicitar corrida. Após preencher o destino, o usuário é redirecionado para a tela de estimativa de tarifa (Figura 34), que mostra o valor previsto da viagem e tempo de chegada do motorista. A partir desta tela, o passageiro pode confirmar ou cancelar a solicitação.



Figura 39. Página de acompanhamento.

A Figura 35 apresenta a tela de acompanhamento da corrida, que exibe no mapa a posição do motorista em tempo real, dados do motorista e do veículo, além de um botão para cancelamento da corrida antes do início.

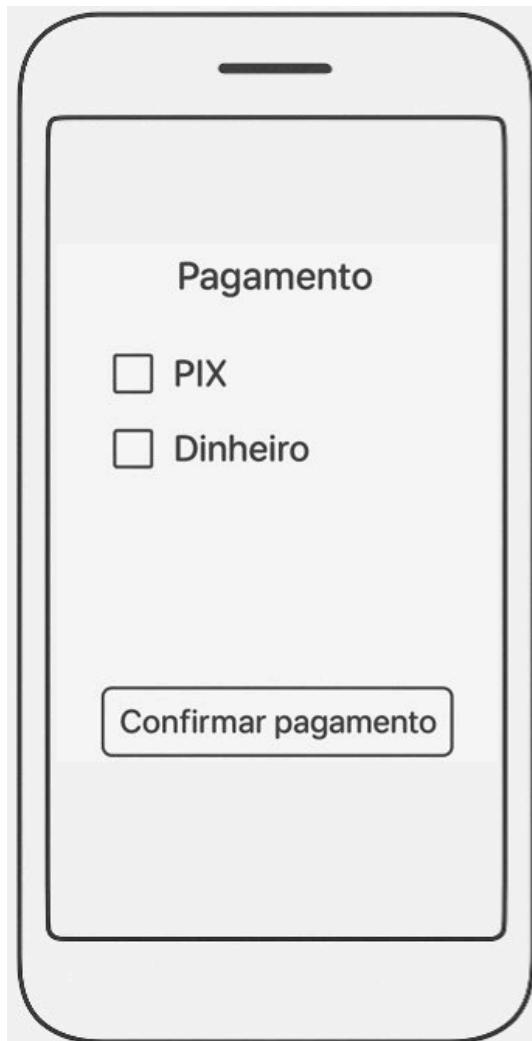


Figura 40. Página forma de pagamento.



Figura 41. Página perfil do usuário.

Após a finalização da viagem, o passageiro é redirecionado à tela de pagamento (Figura 36), onde escolhe entre PIX ou dinheiro. A Figura 37 mostra a tela de perfil do usuário, na qual é possível editar informações pessoais, acessar o histórico de corridas (Figura 37) e encerrar a sessão. O histórico apresenta uma lista de viagens anteriores com informações resumidas como data, valor e status.

## 5. Glossário e Modelos de Dados

Esta Seção tem como objetivo descrever os modelos de dados que compõem a aplicação MobU, assim como definir um glossário que permite interpretar os principais conceitos e atributos utilizados no sistema.

Tendo isso em vista, as Tabelas 4, 5, 6, 7, 8, 9, 10 e 11 tem como objetivo apresentar os diversos atributos que servem de entrada ou saída para a aplicação e que são específicos deste projeto, contemplando as informações manipuladas por usuários, motoristas e administradores.

Da mesma forma, a Figura 43 tem como objetivo representar a camada de banco de dados da aplicação por meio de um Diagrama Entidade-Relacionamento (DER). Nesse diagrama, é possível identificar todas as entidades implementadas, tais como Usuário, Motorista, Veículo, Corrida, Localização, Pagamento e Avaliação, bem como a maneira pela qual elas se relacionam.

O modelo de dados do MobU foi estruturado para garantir consistência, integridade e rastreabilidade das informações. Cada corrida é associada a um passageiro e motorista, contendo dados de origem, destino, valor e status operacional. Os pagamentos armazenam o método utilizado (PIX ou dinheiro), o status da transação e os dados de confirmação. Já as avaliações permitem o registro da nota e do comentário referente à experiência de corrida.

Por fim, o veículo é vinculado diretamente ao motorista e contém as informações de identificação e uso dentro do sistema.

Autenticação e Acesso		
Atributo	Formato	Descrição
Telefone	Texto	Número de telefone do usuário (passageiro ou motorista) utilizado para login.
Código de verificação	Número (6 dígitos)	Código enviado por SMS/WhatsApp para validar a sessão do usuário.
Token de sessão	Texto	Identificador temporário de autenticação do usuário no app.
Perfil	Enum (PASSAGEIRO, MOTORISTA, ADMIN)	Tipo de usuário na plataforma.

Autenticação e Acesso		
Status de conta	Enum (ATIVA, SUSPENSA)	Situação de uso da conta.

*Tabela 8. Autenticação e Acesso*

Usuário e Motorista		
Atributo	Formato	Descrição
Nome	Texto	Nome completo do usuário.
CPF (opcional)	Texto	Documento usado para verificação, quando aplicável.
CNH	Texto	Número da carteira de motorista (apenas motoristas).
Validade CNH	Data	Data de validade da CNH (motoristas).
Rating	Número (0–5, 1 casa)	Média de avaliações recebidas.
Total de corridas	Número inteiro	Quantidade total de corridas concluídas.

*Tabela 9. Usuário e Motorista*

Veículo (motorista)		
Atributo	Formato	Descrição
Placa	Texto	Placa do veículo.
Modelo	Texto	Modelo do veículo.
Cor	Texto	Cor do veículo.
Ano	Número inteiro	Ano de fabricação.
Capacidade	Número inteiro	Quantidade máxima de passageiros.

Tabela 10. Veículo (motorista)

Corrida		
Atributo	Formato	Descrição
ID da corrida	UUID	Identificador único da corrida.
Origem	Objeto Localização	Local de embarque (lat, lon, endereço).
Destino	Objeto Localização	Local de desembarque (lat, lon, endereço).

Corrida		
Distância estimada	Número (km)	Distância prevista do trajeto.
Tempo estimado	Número (min)	Tempo previsto do trajeto.
Tarifa base	Money	Valor base aplicado à corrida.
Preço dinâmico	Número (fator)	Multiplicador por demanda/região.
Valor previsto	Money	Preço esperado antes do embarque.
Status da corrida	Enum (SOLICITADA, DESPACHADA, ACEITA, A_CAMINHO_ORIGEM, CHEGOU_ORIGEM, EM_ANDAMENTO, FINALIZADA, CANCELADA)	Estado operacional da corrida.
Início	Data/Hora	Momento em que a corrida é iniciada.
Fim	Data/Hora	Momento em que a corrida é finalizada.

Tabela 11. Corrida

Localização e Rota		
Atributo	Formato	Descrição
Latitude	Número (double)	Coordenada geográfica.
Longitude	Número (double)	Coordenada geográfica.
Precisão	Número (metros)	Precisão do ponto (opcional).
Rota (waypoints)	Lista de Localização	Pontos intermediários do trajeto.

Tabela 12. Localização e Rota

Pagamento		
Atributo	Formato	Descrição
Método	Enum (PIX, DINHEIRO)	Método escolhido pelo passageiro.
Status do pagamento	Enum (AGUARDANDO, PIX_GERADO, AGUARDANDO_CONF_MOTORISTA, CONFIRMADO, EXPIRADO, CANCELADO)	Estado da transação.
Valor cobrado	Money	Valor efetivo a ser pago.

Pagamento		
TXID	Texto (até 25)	Identificador único do pagamento PIX (quando aplicável).
Payload BR Code	Texto	Conteúdo “copia e cola”/QR do PIX.
Confirmado por	Enum (MOTORISTA, SISTEMA)	Origem da confirmação (manual/automática).
Confirmado em	Data/Hora	Momento de confirmação.

Tabela 13. Pagamento

Avaliação e Suporte		
Atributo	Formato	Descrição
Nota	Número (0–5)	Avaliação atribuída ao motorista/pass.
Comentário	Texto	Observação opcional do avaliador.
Tipo de problema	Enum	Classificação de chamados (cancelamento, cobrança, segurança, etc.).

Avaliação e Suporte		
Observação	Texto	Detalhes adicionais do chamado.

*Tabela 14. Avaliação e Suporte*

Parâmetros Administrativos		
Atributo	Formato	Descrição
Tarifa por km	Money	Valor por quilômetro rodado.
Tarifa por min	Money	Valor por minuto de trajeto.
Taxa da plataforma	Percentual	Comissão aplicada sobre o valor da corrida.
Regiões e taxas	Objeto (Geo/JSON)	Regras por região da cidade.

*Tabela 15. Parâmetros Administrativos*

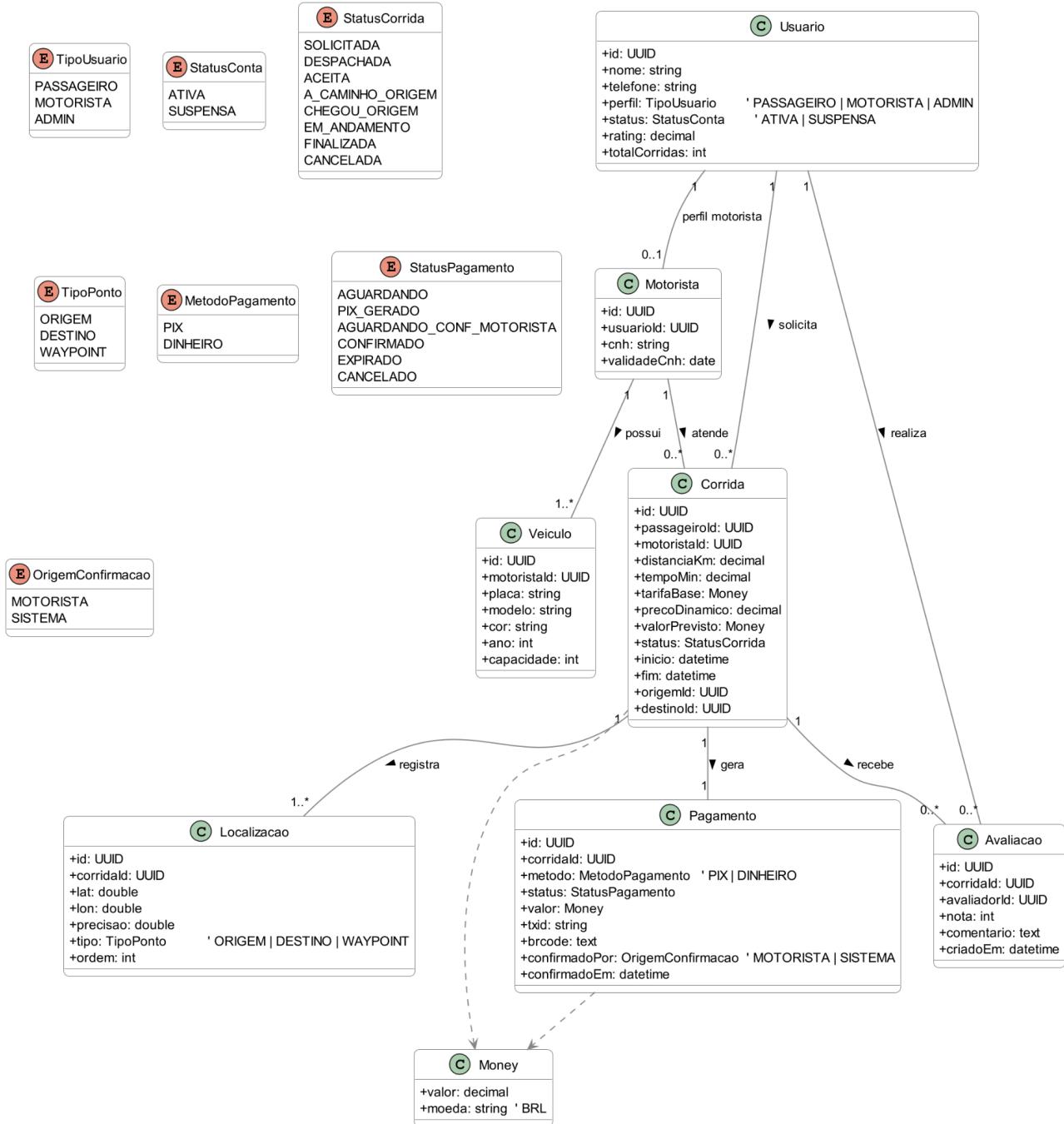


Figura 42. Diagrama de Entidade e Relacionamento.

## 6. Casos de Teste

Esta seção tem como objetivo apresentar os testes de aceitação planejados para o sistema MobU, garantindo que as funcionalidades previstas no Documento de Visão sejam devidamente verificadas. Os testes aqui definidos serão implementados futuramente e têm como propósito assegurar que o

sistema atenda às necessidades essenciais de seus usuários — passageiros e motoristas — permitindo o uso correto e seguro dos principais fluxos da aplicação.

Cada teste de aceitação é descrito por meio de uma tabela contendo quatro campos principais: identificador, pré-condição, ações e resultados esperados. O identificador permite referenciar o teste de maneira única. A pré-condição define o estado necessário para que o teste possa ser iniciado. As ações descrevem passo a passo as interações do usuário na aplicação. Por fim, os resultados descrevem o comportamento esperado do sistema após a execução do teste.

Esses testes serão utilizados futuramente para validar a experiência do usuário, confirmar o correto funcionamento dos fluxos críticos e demonstrar que o sistema opera conforme os requisitos estabelecidos para a primeira entrega funcional do MobU. Todos os testes apresentados foram definidos com base nas necessidades identificadas durante o processo de levantamento de requisitos e serão executados conforme o progresso do desenvolvimento.

## 6.1 Testes de Aceitação — Necessidade 1: Passageiro solicitar corrida

A primeira necessidade prevê que o passageiro seja capaz de solicitar uma corrida informando origem e destino. Para essa necessidade, foram definidos três testes de aceitação, relacionados à criação da conta, à solicitação da corrida e à visualização do motorista no mapa.

<b>Caso de Teste de Aceitação 1: Passageiro deve conseguir criar conta no MobU</b>	
<b>Identificador</b>	TA1
<b>Pré-condição</b>	O passageiro não deve possuir cadastro prévio no MobU.
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Inserir número de telefone.</li> <li>2. Inserir código de validação enviado por SMS.</li> <li>3. Informar nome, e-mail e aceitar os termos de uso.</li> </ol>
<b>Resultados</b>	A conta deve ser criada e o usuário deve ser redirecionado ao dashboard inicial.

Tabela 16. Caso de teste de aceitação 1

<b>Caso de Teste de Aceitação 2: Passageiro deve conseguir solicitar uma corrida</b>	
<b>Identificador</b>	TA2

<b>Pré-condição</b>	O passageiro deve estar autenticado na aplicação.
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Inserir origem e destino.</li> <li>2. Confirmar os dados da corrida.</li> <li>3. Selecionar forma de pagamento (dinheiro ou Pix offline).</li> <li>4. Solicitar corrida.</li> </ol>
<b>Resultados</b>	O sistema deve registrar a corrida e exibir o status “Aguardando motorista”.

Tabela 17. Caso de teste de aceitação 2

<b>Caso de Teste de Aceitação 3:</b> Passageiro deve conseguir acompanhar o motorista	
<b>Identificador</b>	TA3
<b>Pré-condição</b>	O motorista deve ter aceitado a corrida.
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Abrir tela de acompanhamento.</li> <li>2. Visualizar a posição do motorista no mapa em tempo real.</li> </ol>
<b>Resultados</b>	O passageiro deve visualizar o motorista se movendo no mapa até o local de embarque.

Tabela 18. Caso de teste de aceitação 3

### 6.1.2 Necessidade 2 — Motorista aceitar e realizar corrida

A segunda necessidade garante que o motorista seja capaz de receber, aceitar e concluir corridas, validando o funcionamento da API, os eventos e a comunicação entre os módulos.

<b>Caso de Teste de Aceitação 4:</b> Motorista aceitar corrida	
<b>Identificador</b>	TA4
<b>Pré-condição</b>	O motorista deve estar autenticado e marcado como “online”.
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Receber notificação de corrida.</li> <li>2. Visualizar origem e destino.</li> <li>3. Aceitar corrida.</li> </ol>
<b>Resultados</b>	A corrida passa para o estado “Aceita” e o

	motorista passa para o modo de navegação.
--	---

Tabela 19. Caso de teste de aceitação 4

<b>Caso de Teste de Aceitação 5:</b> Motorista finalizar corrida	
<b>Identificador</b>	TA5
<b>Pré-condição</b>	A corrida deve estar em andamento.
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Marcar chegada ao destino.</li> <li>2. Finalizar corrida.</li> <li>3. Confirmar valor final com o passageiro.</li> </ol>
<b>Resultados</b>	A corrida deve ser finalizada e registrada no histórico do motorista e do passageiro.

Tabela 20. Caso de teste de aceitação 5

<b>Caso de Teste de Aceitação 6:</b> O Motorista deve ser capaz de gerenciar seu status (online/offline)	
<b>Identificador</b>	TA6
<b>Pré-condição</b>	O motorista deve estar autenticado na aplicação.
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Acessar o painel inicial.</li> <li>2. Selecionar o botão “Ficar Online”.</li> </ol>
<b>Resultados</b>	O motorista passa a receber solicitações de corrida e seu status é atualizado no sistema.

Tabela 21. Caso de teste de aceitação 6

### 6.1.3 Necessidade 3 — Pagamento via PIX Off-line

Esta necessidade refere-se ao processo de geração de QR Code estático utilizando chave Pix previamente cadastrada no banco de dados, sem uso de API de instituição financeira.

<b>Caso de Teste de Aceitação 7:</b> Gerar QR Code Pix estático	
<b>Identificador</b>	TA7
<b>Pré-condição</b>	O motorista deve possuir chave Pix cadastrada no sistema
<b>Ações</b>	<ol style="list-style-type: none"> <li>1. Finalizar corrida.</li> </ol>

	2. Selecionar pagamento via Pix. 3. Gerar QR Code com valor.
<b>Resultados</b>	O QR Code deve ser exibido corretamente e pode ser pago pelo aplicativo bancário do passageiro.

Tabela 22. Caso de teste de aceitação 7

<b>Caso de Teste de Aceitação 8:</b> Registrar pagamento não confirmado	
<b>Identificador</b>	TA8
<b>Pré-condição</b>	A corrida deve estar finalizada e aguardando validação de pagamento.
<b>Ações</b>	1. Selecionar corrida finalizada. 2. Informar “Pagamento não recebido”.
<b>Resultados</b>	O sistema marca o pagamento como “Pendente”, registrando data e motorista responsável.

Tabela 23. Caso de teste de aceitação 8

<b>Caso de Teste de Aceitação 9:</b> Visualizar comprovante do PIX	
<b>Identificador</b>	TA9
<b>Pré-condição</b>	O pagamento da corrida deve estar marcado como “Pago”.
<b>Ações</b>	1. Abrir detalhes da corrida. 2. Acessar a área de pagamentos.
<b>Resultados</b>	O comprovante (txid + dados da cobrança) é exibido ao motorista e ao passageiro.

Tabela 24. Caso de teste de aceitação 9

#### 6.1.4 Necessidade 4 — Administração de Usuários e Monitoramento do Sistema

Esta necessidade refere-se às funcionalidades destinadas ao administrador da plataforma MobU, permitindo a gestão centralizada de usuários e o acompanhamento operacional do sistema. O administrador deve ser capaz de visualizar, monitorar e realizar ações de controle sobre contas de passageiros e motoristas, bem como acessar relatórios consolidados de uso da aplicação.

<b>Caso de Teste de Aceitação 10:</b> O Administrador deve gerenciar motoristas e passageiros	
<b>Identificador</b>	TA10
<b>Pré-condição</b>	O administrador deve estar autenticado no painel web.
<b>Ações</b>	1. Abrir a aba “Usuários”.
<b>Resultados</b>	O sistema exibe lista com passageiros e motoristas cadastrados.

Tabela 25. Caso de teste de aceitação 10

<b>Caso de Teste de Aceitação 11:</b> Bloquear usuário	
<b>Identificador</b>	TA11
<b>Pré-condição</b>	Usuário selecionado deve estar ativo no sistema.
<b>Ações</b>	1. Selecionar usuário. 2. Clicar em “Bloquear usuário”
<b>Resultados</b>	O usuário fica impedido de acessar o sistema e seu status é marcado como “Bloqueado”.

Tabela 26. Caso de teste de aceitação 11

<b>Caso de Teste de Aceitação 12:</b> Visualizar relatórios de corridas	
<b>Identificador</b>	TA12
<b>Pré-condição</b>	O sistema deve conter corridas concluídas.
<b>Ações</b>	1. Acessar a aba “Relatórios”. 2. Selecionar período desejado.
<b>Resultados</b>	Um relatório com número de corridas, ganhos e estatísticas é exibido ao administrador.

Tabela 27. Caso de teste de aceitação 12

## 6.2 Testes de Integração

Nesta seção serão descritos os testes de integração previstos para o sistema MobU. Os testes de integração serão responsáveis por verificar se os módulos principais da solução — aplicativo móvel, API MobU e serviços externos (API de Mapas/Geo, serviço de notificações push e futuro gateway

de pagamento) — se comunicam corretamente, trocando dados de forma consistente e mantendo o fluxo das corridas do início ao fim.

Cada caso de teste de integração será documentado por meio de uma tabela contendo: identificador, sistemas envolvidos, interface utilizada, pré-condições, entradas e resultados esperados. A partir desses casos, será possível validar cenários críticos como autenticação de usuário, solicitação de corrida com cálculo de rota, envio de notificações para motoristas e registro de pagamentos.

Como estratégia geral, pretende-se adotar uma abordagem top-down. Inicialmente, serão implementadas e testadas as integrações entre a MobU API e os serviços externos, utilizando mocks para simular respostas da API de Mapas/Geo, do serviço de notificações push (FCM). Em um segundo momento, o aplicativo móvel será conectado à API real, permitindo a execução de testes ponta a ponta. A automatização desses testes deverá ser feita por meio de suítes de testes da API e, quando possível, por testes de integração no aplicativo móvel, garantindo que regressões possam ser detectadas de forma rápida ao longo do desenvolvimento.

A Tabela 28 apresenta o caso de teste de integração que valida o processo de autenticação de usuários no MobU. Esse teste verifica a comunicação entre o aplicativo móvel e a API, garantindo que credenciais válidas resultem em geração de token e acesso ao perfil correspondente.

<b>Identificador</b>	TI1 – Autenticação de usuário
<b>Sistemas Envolvidos</b>	Mobile App (Passageiro/Motorista) e MobU API
<b>Interface</b>	Requisição HTTP REST ( <a href="#">POST /auth/login</a> )
<b>Pré-condição</b>	Usuário cadastrado no banco de dados e servidor da API em execução.
<b>Entradas</b>	1. Enviar telefone/e-mail e senha válidos a partir da tela de login.
<b>Resultados</b>	API deve retornar código <b>200</b> , token JWT e dados básicos do usuário; o aplicativo deve armazenar o token e redirecionar para a tela inicial do perfil correspondente.

*Tabela 28. Caso de teste de integração 1*

A Tabela 29 descreve o teste de integração relacionado à solicitação de corrida com cálculo de rota. O teste avalia a interação entre o Mobile App, a MobU API e a API de Mapas, assegurando que origem e destino sejam processados corretamente e que o passageiro visualize estimativas de percurso e valor.

<b>Identificador</b>	TI2 – Solicitação de corrida com rota
----------------------	---------------------------------------

<b>Sistemas Envolvidos</b>	Mobile App (Passageiro), MobU API e API de Mapas/Geo
<b>Interface</b>	HTTP REST entre App e MobU API; HTTP/HTTPS entre MobU API e API de Mapas
<b>Pré-condição</b>	Passageiro autenticado; localização de origem habilitada no dispositivo.
<b>Entradas</b>	1. Informar origem e destino no aplicativo. 2. Enviar requisição para criação de corrida ( <code>POST /rides/estimate</code> ou similar).
<b>Resultados</b>	API de Mapas deve retornar rota e distância; MobU API deve calcular valor estimado e tempo de viagem e responder ao app com os dados consolidados; o app deve exibir o resumo para confirmação do passageiro.

*Tabela 29. Caso de teste de integração 2*

A Tabela 30 apresenta o caso de teste que verifica o envio de notificações push ao motorista quando uma nova corrida é confirmada. Esse teste garante o funcionamento integrado entre a MobU API, o serviço FCM e o aplicativo do motorista.

<b>Identificador</b>	TI3 – Notificação de nova corrida
<b>Sistemas Envolvidos</b>	MobU API, serviço de notificações push (FCM) e Mobile App (Motorista)
<b>Interface</b>	HTTP/HTTPS entre MobU API e FCM; canal de Push Notification entre FCM e App
<b>Pré-condição</b>	Motorista autenticado e marcado como <b>online</b> ; token de notificação registrado na API.
<b>Entradas</b>	1. Passageiro confirma solicitação de corrida. 2. MobU API envia mensagem de nova corrida ao FCM com dados resumidos (origem, destino, valor estimado).
<b>Resultados</b>	Motorista deve receber uma notificação push no dispositivo; ao tocar na notificação, o aplicativo deve abrir a tela de detalhes da corrida com as informações recebidas da API.

*Tabela 30. Caso de teste de integração 3*

A Tabela 31 descreve o teste de integração destinado a validar o processo de registro de pagamento PIX offline, sem API bancária. O teste garante que a confirmação enviada pelo motorista seja corretamente recebida e registrada pela API.

<b>Identificador</b>	TI4 – Registro de pagamento PIX off-line
<b>Sistemas Envolvidos</b>	Mobile App (Passageiro/Motorista) e MobU API
<b>Interface</b>	HTTP REST ( <a href="#">POST /payments/confirm</a> )
<b>Pré-condição</b>	Corrida finalizada e QR Code PIX já gerado pela API; passageiro realizou o pagamento utilizando o aplicativo bancário externo.
<b>Entradas</b>	1. Motorista informa, no app, que o pagamento foi recebido. 2. App envia confirmação de pagamento à API com o identificador da corrida.
<b>Resultados</b>	MobU API deve atualizar o status do pagamento para <b>CONFIRMADO</b> e da corrida para <b>PAGA</b> ; o aplicativo deve refletir o novo status no histórico de corridas do passageiro e do motorista.

Tabela 31. Caso de teste de integração 4

A Tabela 32 apresenta o teste de integração sobre a atualização do status do motorista entre online e offline. O teste assegura que o aplicativo acione a API corretamente e que o estado atualizado seja refletido no sistema.

<b>Identificador</b>	TI5 – Alternar status online/off-line
<b>Sistemas Envolvidos</b>	Mobile App (Motorista) e MobU API
<b>Interface</b>	Requisição HTTP REST ( <a href="#">PATCH /drivers/status</a> )
<b>Pré-condição</b>	Motorista autenticado no aplicativo.
<b>Entradas</b>	1. Motorista aciona o botão “Ficar online/off-line” na tela principal do app.
<b>Resultados</b>	A API deve atualizar o status do motorista no banco de dados; o aplicativo deve receber a confirmação e exibir o novo estado (online ou off-line). Motoristas <b>online</b> passam a ser considerados pelo serviço de <i>matching</i> de corridas.

Tabela 32. Caso de teste de integração 5

A Tabela 33 descreve o teste que valida o fluxo de aceitação de corrida pelo motorista. O caso assegura que a API atualize o estado da corrida e que a API de Mapas forneça o trajeto adequado até o local de embarque.

<b>Identificador</b>	TI6 – Aceitar corrida e iniciar rota
<b>Sistemas Envolvidos</b>	Mobile App (Motorista), MobU API e API de Mapas/Geo
<b>Interface</b>	HTTP REST entre App e MobU API; HTTP/HTTPS entre MobU API e API de Mapas
<b>Pré-condição</b>	Corrida em estado <b>DESPACHADA</b> para o motorista; motorista em modo <b>online</b> .
<b>Entradas</b>	1. Motorista clica em “Aceitar corrida”. 2. Aplicativo solicita rota até o ponto de embarque.
<b>Resultados</b>	MobU API atualiza a corrida para estado <b>ACEITA</b> ; API de Mapas retorna a rota até a origem; o aplicativo exibe o trajeto no mapa, pronto para navegação.

Tabela 33. Caso de teste de integração 6

A Tabela 34 apresenta o teste de integração que avalia o processo de finalização de corrida com pagamento em dinheiro. O teste confirma que a API atualiza o estado da corrida e registra o pagamento corretamente.

<b>Identificador</b>	TI7 – Finalizar corrida com pagamento em dinheiro
<b>Sistemas Envolvidos</b>	Mobile App (Motorista/Passageiro) e MobU API
<b>Interface</b>	Requisição HTTP REST ( <code>POST /rides/{id}/finish</code> )
<b>Pré-condição</b>	Corrida em estado <b>EM_ANDAMENTO</b> ; método de pagamento configurado como <b>DINHEIRO</b> .
<b>Entradas</b>	1. Motorista aciona “Finalizar corrida” ao chegar ao destino. 2. App envia para a API o identificador da corrida e o valor final calculado.
<b>Resultados</b>	MobU API deve atualizar o estado da corrida para <b>FINALIZADA</b> e registrar o pagamento como <b>CONFIRMADO (DINHEIRO)</b> ; o histórico de corridas do passageiro e do motorista deve exibir a corrida com status concluído e valores corretos.

Tabela 34. Caso de teste de integração 7

A Tabela 35 descreve o caso de teste utilizado para validar a consulta de histórico de ganhos e corridas realizadas. O teste garante o funcionamento adequado da API ao retornar dados consolidados com base nos filtros selecionados pelo motorista.

<b>Identificador</b>	TI8 – Consultar histórico de corridas e ganhos			
<b>Sistemas Envolvidos</b>	Mobile App (Motorista) e MobU API			
<b>Interface</b>	Requisição <code>/drivers/{id}/earnings?periodo=...</code>	HTTP	REST	(GET)
<b>Pré-condição</b>	Motorista autenticado e com corridas finalizadas registradas no sistema.			
<b>Entradas</b>	1. Motorista acessa a tela “Histórico e ganhos” e seleciona um período (ex.: última semana).			
<b>Resultados</b>	MobU API deve retornar lista de corridas do período, com valores individuais e total acumulado; o aplicativo deve exibir o extrato com as corridas e o total de ganhos no intervalo selecionado.			

Tabela 35. Caso de teste de integração 8

A Tabela 36 apresenta o teste de integração referente ao módulo administrativo do sistema, verificando a geração de relatórios por meio de consultas agregadas realizadas pela API. O teste comprova que filtros selecionados pelo administrador resultam em indicadores consistentes para gestão.

<b>Identificador</b>	TI9 – Relatórios administrativos			
<b>Sistemas Envolvidos</b>	Painel Web/Admin (ou módulo futuro), MobU API e SGBD			
<b>Interface</b>	HTTP REST entre Painel/Admin e MobU API; conexão SQL entre API e banco de dados			
<b>Pré-condição</b>	Usuário administrador autenticado; dados de corridas, pagamentos e usuários previamente cadastrados.			
<b>Entradas</b>	1. Administrador seleciona filtros de período, cidade e tipo de usuário. 2. Painel envia requisição à API (GET <code>/admin/reports?filtros=...</code> ).			
<b>Resultados</b>	MobU API executa consultas agregadas no banco de dados e retorna indicadores (quantidade de corridas, valor movimentado, média de avaliação, etc.); o painel exibe os dados em tabela ou gráfico, permitindo download ou exportação se previsto.			

Tabela 36. Caso de teste de integração 9

## 7. Cronograma e Processo de Implementação

Nesta seção é apresentado o cronograma de desenvolvimento do sistema MobU ao longo do primeiro semestre de 2026. O cronograma está organizado por quinzenas, iniciando na primeira quinzena de fevereiro de 2026 e encerrando na segunda quinzena de junho de 2026. Em cada período são listadas as atividades a serem desenvolvidas, contemplando a construção da aplicação móvel, API, infraestrutura, testes, documentação e validações finais.

### 7.1 Cronograma

O cronograma do projeto MobU foi estruturado em períodos quinzenais ao longo do primeiro semestre de 2026, permitindo uma organização clara das entregas e um acompanhamento contínuo da evolução do desenvolvimento. Cada quinzena contempla um conjunto de atividades planejadas, abrangendo desde a preparação inicial dos ambientes, definição da arquitetura e modelos de dados, até a implementação progressiva dos fluxos da aplicação móvel e da API, testes, ajustes e validações.

A divisão quinzenal possibilita que o projeto seja desenvolvido de forma incremental, garantindo que funcionalidades essenciais sejam entregues gradualmente e validadas ao longo do processo. Dessa forma, cada período apresenta objetivos específicos que contribuem diretamente para a construção completa do sistema, incluindo desenvolvimento do aplicativo móvel, camada de serviços, integração com APIs externas, persistência de dados, testes de aceitação e integração, além de atividades de documentação e refinamentos finais.

A Tabela a seguir apresenta a distribuição das atividades planejadas para cada quinzena, detalhando o escopo de trabalho previsto para todo o ciclo de desenvolvimento do MobU.

Período	Atividades
01/02 – 15/02 (1ª Quinzena de Fevereiro)	<ul style="list-style-type: none"><li>● Configuração inicial dos repositórios (Mobile + API).</li><li>● Definição da arquitetura geral do sistema (camadas, padrões e tecnologias).</li><li>● Criação da estrutura de pastas do aplicativo móvel:<ul style="list-style-type: none"><li>○ Estrutura base</li><li>○ Tema da aplicação</li></ul></li></ul>

	<ul style="list-style-type: none"> <li>○ Instalação de bibliotecas essenciais</li> <li>○ Arquivo de rotas e navegação</li> <li>● Criação da estrutura inicial da API (Node/Express):           <ul style="list-style-type: none"> <li>○ Estrutura de pastas</li> <li>○ Instalação de dependências principais</li> <li>○ Configuração de ambiente e variáveis (.env)</li> </ul> </li> <li>● Configuração inicial do banco de dados PostgreSQL.</li> <li>● Criação do primeiro DER rascunho.</li> </ul>
16/02 – 29/02 (2ª Quinzena de Fevereiro)	<ul style="list-style-type: none"> <li>● Implementação da tela de login e fluxo de autenticação no Mobile.</li> <li>● Implementação dos casos de uso:           <ul style="list-style-type: none"> <li>○ UC01 Criar conta / Entrar</li> <li>○ UC02 Gerenciar Perfil</li> </ul> </li> <li>● Implementação dos primeiros endpoints de autenticação da API.</li> <li>● Integração mobile ↔ API (login, criação de usuário).</li> <li>● Implementação da estrutura de tokens (JWT).</li> <li>● Configuração do ambiente Docker da API.</li> </ul>

01/03 – 15/03 (1ª Quinzena de Março)	<ul style="list-style-type: none"> <li>● Implementação dos casos de uso relacionados às corridas:           <ul style="list-style-type: none"> <li>○ UC03 Informar origem/destino</li> <li>○ UC04 Estimar tarifa</li> <li>○ UC05 Solicitar corrida</li> </ul> </li> <li>● Comunicação com API externa de mapas/geo (cálculo de rota e distância).</li> <li>● Implementação dos endpoints de cálculo de tarifa e solicitação de corrida.</li> <li>● Modelagem final das entidades:           <ul style="list-style-type: none"> <li>○ Usuário</li> <li>○ Motorista</li> <li>○ Veículo</li> <li>○ Corrida</li> <li>○ Localização</li> </ul> </li> </ul>
16/03 – 31/03 (2ª Quinzena de Março)	<ul style="list-style-type: none"> <li>● Implementação das telas do motorista:           <ul style="list-style-type: none"> <li>○ UC10 Ficar online/offline</li> <li>○ UC11 Receber solicitações</li> <li>○ UC12 Aceitar/recusar corrida</li> </ul> </li> <li>● Implementação do fluxo de despacho de corridas.</li> <li>● Implementação de notificações via FCM (entrada e aceite de corrida).</li> </ul>

	<ul style="list-style-type: none"> <li>● Implementação dos endpoints:           <ul style="list-style-type: none"> <li>○ Despacho</li> <li>○ Aceite</li> <li>○ Rejeição</li> </ul> </li> <li>● Testes de integração Mobile ↔ API ↔ FCM.</li> </ul>
01/04 – 15/04 (1ª Quinzena de Abril)	<ul style="list-style-type: none"> <li>● Implementação do fluxo operacional da corrida:           <ul style="list-style-type: none"> <li>○ UC13 Navegar até embarque/destino</li> <li>○ UC14 Marcar etapas (cheguei/iniciei/finalizei)</li> </ul> </li> <li>● Implementação de trilhas de localização e mapas no aplicativo do motorista.</li> <li>● Configuração final do sistema de tracking em tempo real.</li> <li>● Implementação de testes de aceitação para todo o fluxo até finalização da corrida.</li> </ul>
16/04 – 30/04 (2ª Quinzena de Abril)	<ul style="list-style-type: none"> <li>● Implementação do sistema de pagamento:           <ul style="list-style-type: none"> <li>○ UC08 Pagar (PIX/Dinheiro)</li> </ul> </li> <li>● Geração do QR Code PIX dentro da aplicação (offline, sem PSP).</li> <li>● Implementação dos estados de</li> </ul>

	<p>pagamento (Figura XX):</p> <ul style="list-style-type: none"> <li>○ Aguardando</li> <li>○ PIX_GERADO</li> <li>○ Aguardando confirmação</li> <li>○ Confirmado / Cancelado</li> </ul> <ul style="list-style-type: none"> <li>● Implementação dos endpoints de pagamento e confirmação do motorista.</li> <li>● Implementação das telas e fluxos relacionados.</li> <li>● Testes de integração Mobile ↔ API ↔ Pagamentos.</li> </ul>
01/05 – 15/05 (1ª Quinzena de Maio)	<ul style="list-style-type: none"> <li>● Implementação do histórico de corridas: <ul style="list-style-type: none"> <li>○ UC07 Cancelar corrida</li> <li>○ UC09 Avaliar motorista</li> <li>○ UC15 Ver histórico e ganhos</li> </ul> </li> <li>● Implementação dos relatórios internos (admin): <ul style="list-style-type: none"> <li>○ UC17 Ver relatórios e métricas</li> <li>○ UC18 Configurar taxas e regiões</li> </ul> </li> <li>● Criação de Jobs assíncronos (limpeza, cálculos, fechamentos).</li> <li>● Testes de integração ampliados.</li> </ul>
16/05 – 31/05 (2ª Quinzena de Maio)	<ul style="list-style-type: none"> <li>● Desenvolvimento do painel administrativo (versão web simples opcional).</li> </ul>

	<ul style="list-style-type: none"> <li>● Finalização da documentação técnica:           <ul style="list-style-type: none"> <li>○ Diagramas UML</li> <li>○ DER final</li> <li>○ Diagramas de componentes e implantação</li> </ul> </li> <li>● Finalização do módulo de notificações e eventos.</li> <li>● Testes de carga e otimização de consultas.</li> <li>● Correções gerais com base nos testes das sprints anteriores.</li> </ul>
01/06 – 15/06 (1ª Quinzena de Junho)	<ul style="list-style-type: none"> <li>● Preparação do ambiente de deploy (AWS/Firebase).</li> <li>● Configuração de pipelines CI/CD.</li> <li>● Publicação do aplicativo em ambiente de testes.</li> <li>● Homologação interna com cenários completos.</li> </ul>
16/06 – 30/06 (2ª Quinzena de Junho)	<ul style="list-style-type: none"> <li>● Ajustes finais após homologação.</li> <li>● Últimos testes de aceitação.</li> <li>● Preparação e entrega da documentação do projeto.</li> <li>● Apresentação e encerramento do semestre.</li> <li>● Pós-mortem do projeto.</li> </ul>

Tabela 16. Cronograma de Implementação