# Projeto Arquitetural: UberPB

**Versão:** 1.0 **Data:** 06 de Setembro de 2025

## 1. Introdução e Objetivos

### 1.1. Visão Geral do Sistema

O "UberPB" é uma aplicação de simulação de transporte de passageiros e motoristas. O sistema opera inteiramente através de uma interface de linha de comando (terminal), sem uma interface gráfica de usuário. Ele gerencia entidades fundamentais como Passageiro, Motorista, Gerente, Veiculo e Corrida.

### 1.2. Requisitos Arquiteturais (Drivers)

A arquitetura foi guiada pelos seguintes objetivos principais:

* **Estrutura:** O código é estruturado seguindo o padrão MVC para garantir que as responsabilidades (dados, apresentação e controle) sejam bem definidas e que o sistema seja fácil de entender e modificar.
* **Simplicidade de Implantação:** O sistema é auto suficiente, não exigindo a configuração de um banco de dados externo. A persistência de dados é feita em arquivos locais.
* **Clareza no Tratamento de Erros:** O sistema utiliza exceções customizadas para lidar com erros de negócio de forma explícita e controlada.
* **Modularidade:** O uso de interfaces como UsuarioService e Repository permite que as implementações possam ser trocadas no futuro com impacto mínimo no resto do sistema.

## 2. Visão Lógica: Arquitetura MVC (Model-View-Controller)

O sistema é estruturado em torno do padrão de projeto arquitetural **Model-View-Controller**. Este padrão divide a aplicação em três componentes interconectados para separar a representação da informação da interação do usuário com ela.

### 2.1. Detalhamento dos Componentes

* **View (Visão):**
  + **Responsabilidade:** Apresentar os dados ao usuário e capturar suas interações. A View não contém lógica de negócio.
  + **Implementação:** A View do sistema é a **interface de Linha de Comando (Terminal)**. A classe Main.java é a principal responsável por "desenhar" essa interface: exibir menus, imprimir resultados e ler os comandos do usuário.
* **Controller (Controlador):**
  + **Responsabilidade:** Atuar como intermediário entre a View e o Model. Ele recebe as solicitações da View, aciona a lógica apropriada no Model e gerencia o fluxo da aplicação.
  + **Implementação:** A classe Sistema.java atua como o principal Controller. Ela centraliza operações e mantém o estado da sessão em memória (ex: motoristasOnline, passageirosOnline). Quando o usuário executa uma ação na View, esta invoca métodos no Controller para iniciar os processos de negócio.
* **Model (Modelo):**
  + **Responsabilidade:** O coração da aplicação, representando os dados, as regras de negócio e a lógica de persistência.
  + **Implementação:** O Model é uma camada abrangente composta por três subcomponentes:
    - **Entidades (/model/entity):** Representam a estrutura dos dados. São as classes como Usuario (classe base), Motorista, Passageiro, Corrida e Veiculo. Enums como StatusCorrida e CategoriaVeiculo são usados para representar estados e tipos fixos.
    - **Serviços (/model/service):** Contêm a lógica de negócio (o "cérebro"). Classes como CorridaService, PassageiroService e MotoristaService executam as regras de negócio, validações e cálculos.
    - **Repositórios (/model/repository):** Lidam com a persistência dos dados, abstraindo a fonte de dados. A classe genérica JsonRepository<T> é responsável por ler e escrever em arquivos JSON. Implementações específicas como MotoristaRepository e PassageiroRepository a utilizam para fornecer métodos de acesso aos dados para a camada de serviço.

## 3. Visão de Dados

### 3.1. Mecanismo de Persistência

O sistema utiliza **arquivos de texto no formato JSON** como seu banco de dados, com cada tipo de entidade principal armazenada em seu próprio arquivo. A biblioteca **Google Gson** é utilizada para a serialização e desserialização dos objetos Java.

### 3.2. Diagrama de Entidades (Simplificado)

* (Passageiro, Motorista, Gerente) herdam de [Usuario].
* [Motorista] se associa a [Veiculo] através de um idVeiculo.
* [Corrida] se associa a [Passageiro] e [Motorista] através de passageiroId e motoristaId.

## 4. Visão de Implantação (Deployment)

* **Ambiente de Execução:** Java Virtual Machine (JVM).
* **Empacotamento:** A aplicação é empacotada como um arquivo JAR executável.
* **Dependências Externas:** A única dependência externa é a biblioteca com.google.code.gson.

## 5. Sprints

* **Sprint 1 (26/08 – 08/09)**

Durante a Sprint 1, a equipe concentrou-se no desenvolvimento inicial das principais funcionalidades do sistema, garantindo a base para as próximas entregas. As atividades realizadas foram:

* + **Cadastro e autenticação:** CRUD de Passageiros, Persistência dos dados, Implementação de login do sistema, Implementação das categorias de veículos, CRUD de Motorista e CRUD de Gerente.
  + **Solicitação de corrida:** Cálculo de preço, Escolha de categoria para o usuário, Classe corrida, Estimativas da corrida e Notificação aos motoristas selecionados.
  + **Testes de unidade:** Testes das funcionalidades de cadastro/autenticação e solicitação de corrida.
  + **Documentação:** Registro das funcionalidades de Cadastro e Autenticação, além da Solicitação de Corrida.

Todos os itens planejados foram concluídos com sucesso, permitindo validar as primeiras interações entre usuários e o sistema.

* + **Papéis**
    - **Erick:** Testador
    - **Lucas:** Gerente, Desenvolvedor
    - **Manoel:** Desenvolvedor, Engenheiro de software
    - **Mateus:** Desenvolvedor
* **Sprint 2 (09/09 – 22/09)**

Na Sprint 2, o foco foi ampliar as funcionalidades já desenvolvidas, adicionando camadas de acompanhamento e aceitação da corrida, além de melhorias na interface e na documentação. As atividades realizadas foram:

* + **Documentação:** Funcionalidades de Aceite da Corrida e Acompanhamento da Corrida.
  + **Acompanhamento da Corrida:** Estimativa da chegada do motorista e visualização da melhor rota até o destino.
  + **Aceite da Corrida:** Confirmação da corrida pelo motorista e atribuição da corrida ao motorista mais próximo.
  + **Refatoração:** Melhoria da documentação de veículos.
  + **Interface do Usuário:** Implementação da Interface CLI.
  + **Testes de unidade:** Testes das funcionalidades de Aceite da Corrida e Acompanhamento da Corrida.

Grande parte das tarefas foi concluída, enquanto as atividades com os requisitos com funcionalidade de localização do motorista ainda faltam terminar a implementação.

* + **Papéis**
    - **Erick:** Testador
    - **Lucas:** Desenvolvedor
    - **Manoel:** Desenvolvedor, Engenheiro de software
    - **Mateus:** Gerente, Desenvolvedor
* **Sprint 3 (25/09 – 08/10)**

Durante a Sprint 3, o foco da equipe esteve na implementação das funcionalidades relacionadas a **pagamentos**, **refatoração de código**, **testes de unidade** e **documentação**. Todas as atividades planejadas foram concluídas dentro do prazo estabelecido, garantindo a continuidade do desenvolvimento com qualidade e estabilidade.  
  
As principais entregas desta sprint foram:

* + **Refatoração**: Ajustes e melhorias no código, incluindo a adição de requisitos com a funcionalidade de localização do motorista.
  + **Pagamentos**: Implementação das funcionalidades de emissão de recibo eletrônico, cálculo do valor da corrida e integração com múltiplas formas de pagamento (cartão, PayPal e dinheiro).
  + **Testes de unidade**: Desenvolvimento e execução de testes para validar as funcionalidades de emissão de recibo eletrônico, cálculo de valor da corrida e pagamento.
  + **Documentação**: Registro técnico das funcionalidades implementadas, abrangendo pagamento, cálculo de valor da corrida e emissão de recibo eletrônico.

A sprint foi finalizada com 100% das atividades concluídas, consolidando uma base sólida para a próxima etapa do projeto, com foco em garantir qualidade e manutenção das novas funcionalidades.

* + **Papéis**
    - **Erick:** Testador
    - **Lucas:** Desenvolvedor
    - **Manoel:** Gerente, Engenheiro de software
    - **Mateus:** Desenvolvedor