# Documentação de Projeto

## para o sistema

# Get Route

**Versão 1.0**

Projeto de sistema elaborado pelo aluno Joao Pedro Tavares e Amorim
como parte da disciplina **Projeto de Software**.

**15/11/2025**

# Tabela de Conteúdo

**1. Introdução**

**2. Modelos de Usuário e Requisitos**

2.1 Descrição de Atores

2.2 Modelo de Casos de Uso e Histórias de Usuários

2.3 Diagrama de Sequência do Sistema e Contrato de Operações

**3. Modelos de Projeto**

3.1 Arquitetura

3.2 Diagrama de Componentes e Implantação.

3.3 Diagrama de Classes

3.4 Diagramas de Sequência

3.5 Diagramas de Comunicação

3.6 Diagramas de Estados

**4. Modelos de Dados**

# Histórico de Revisões

| Nome | Data | Razões para Mudança | Versão |
|---|---|---|---|
| | | | |
| | | | |

# 1. Introdução

Este documento apresenta os modelos de domínio, requisitos e projeto do sistema **GetRoute**, desenvolvido para otimizar o gerenciamento de pedidos de entrega e a operação logística de empresas que necessitam de organização, rastreabilidade e integração de informações.

A principal referência para a elaboração deste documento é a descrição de problema e solução produzida durante o desenvolvimento do sistema, que destaca desafios reais enfrentados por equipes de logística, como descentralização de dados, erros manuais, baixa rastreabilidade e dificuldade de integração entre ferramentas.

O GetRoute foi projetado como uma solução completa para centralizar pedidos de entrega, validar informações automaticamente, facilitar consultas e filtragens, associar cada pedido ao arquivo de origem e expor dados via API, além de oferecer módulos adicionais para gestão de frota, veículos e planejamento de rotas.

# 2. Modelos de Usuário e Requisitos

## 2.1 Descrição de Atores

### 1. Dono / Operador do Sistema

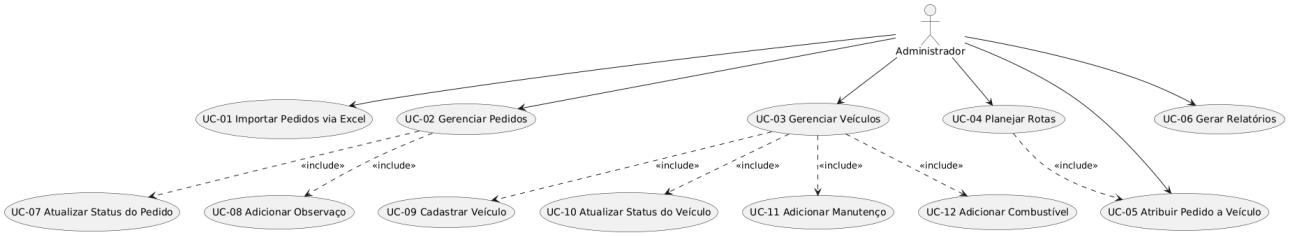
**Descrição:**
É o único usuário humano do sistema. Responsável por cadastrar, importar, consultar e atualizar pedidos, além de gerenciar os veículos e planejar as rotas. Ele opera todas as funcionalidades da plataforma GetRoute.

**Responsabilidades:**

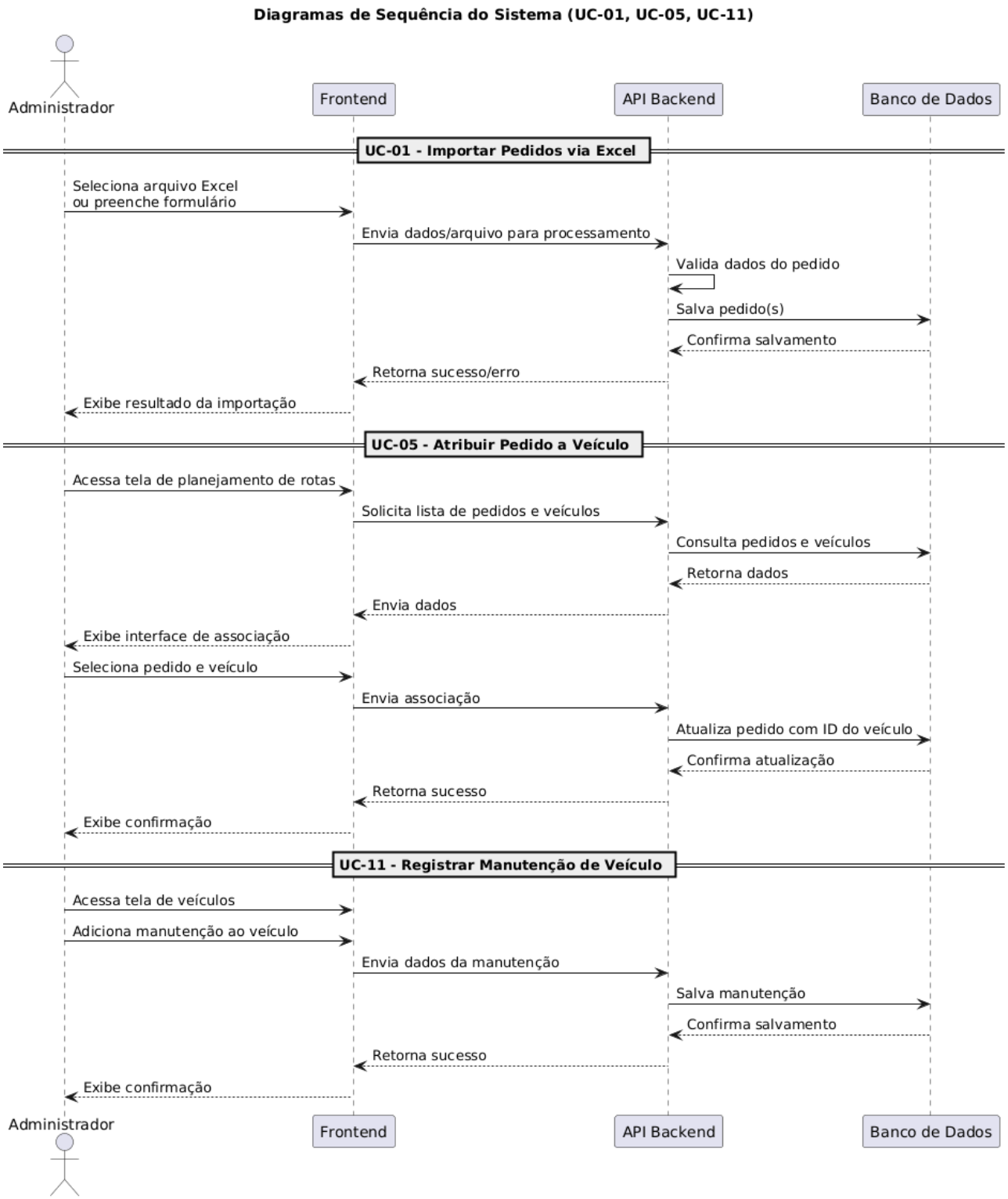
- Cadastrar pedidos manualmente.
- Importar pedidos via arquivo Excel.
- Validar e corrigir dados inconsistentes.
- Consultar pedidos usando filtros avançados.
- Atualizar status, datas, horários e observações.
- Cadastrar e manter dados dos veículos.
- Vincular pedidos aos veículos e planejar rotas.
- Acompanhar a situação da frota (disponível, em rota, manutenção).

- Gerar relatórios e visualizar informações de desempenho.
- Administrar arquivos importados e auditorias.

## 2.2 Modelo de Casos de Uso



## 2.3 Diagrama de Sequência do Sistema

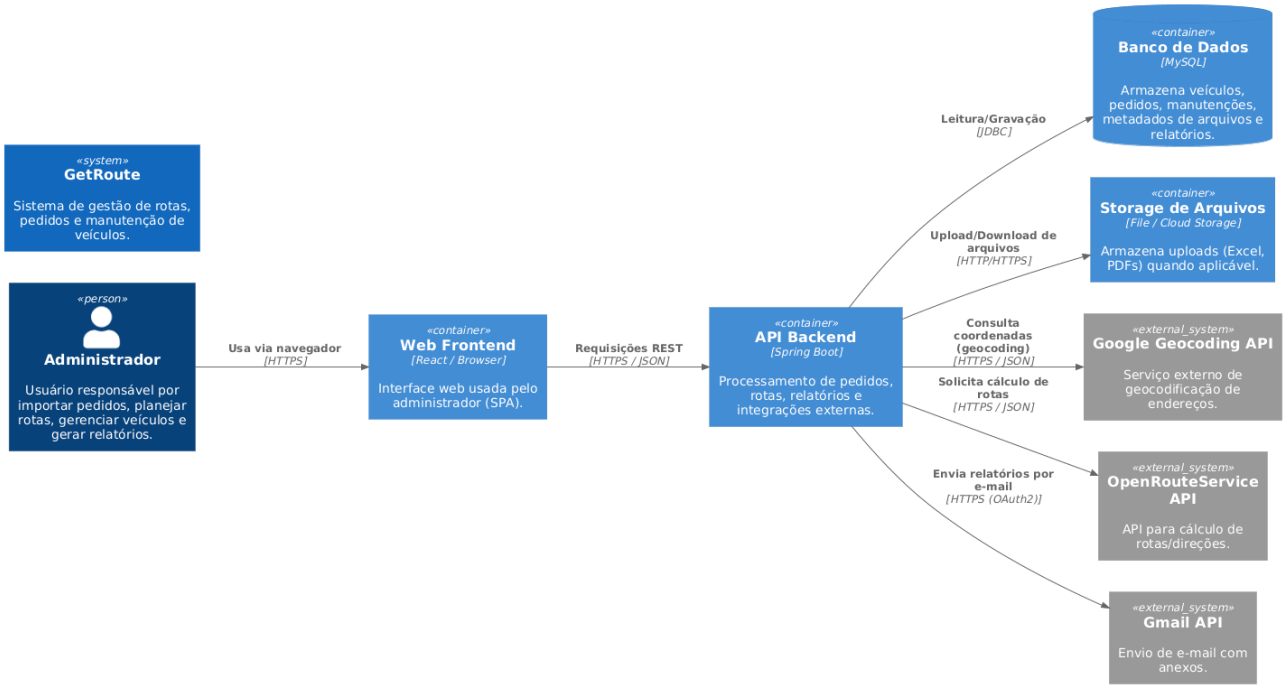


Formato para cada contrato de operação

| | |
|---|---|
| **Contrato** | Processa um arquivo Excel enviado pelo administrador, extrai os pedidos e grava todos no banco de dados vinculados ao arquivo de origem. |
| **Operação** | `importarPedidos(arquivoExcel)` |
| **Referências cruzadas** | UC-01 — Importar Pedidos via Excel} RF-02—Importação de Arquivos RF-03 — Cadastro Automático de Pedidos |
| **Pré-condições** | Administrador deve estar autenticado. Arquivo deve ser `.xlsx` válido. Deve existir conexão com o banco de dados. |
| **Pós-condições** | Arquivo Excel registrado em `excel_file`. Pedidos criados em `pedido` e vinculados ao arquivo. Sistema retorna sucesso ou erros de validação. |

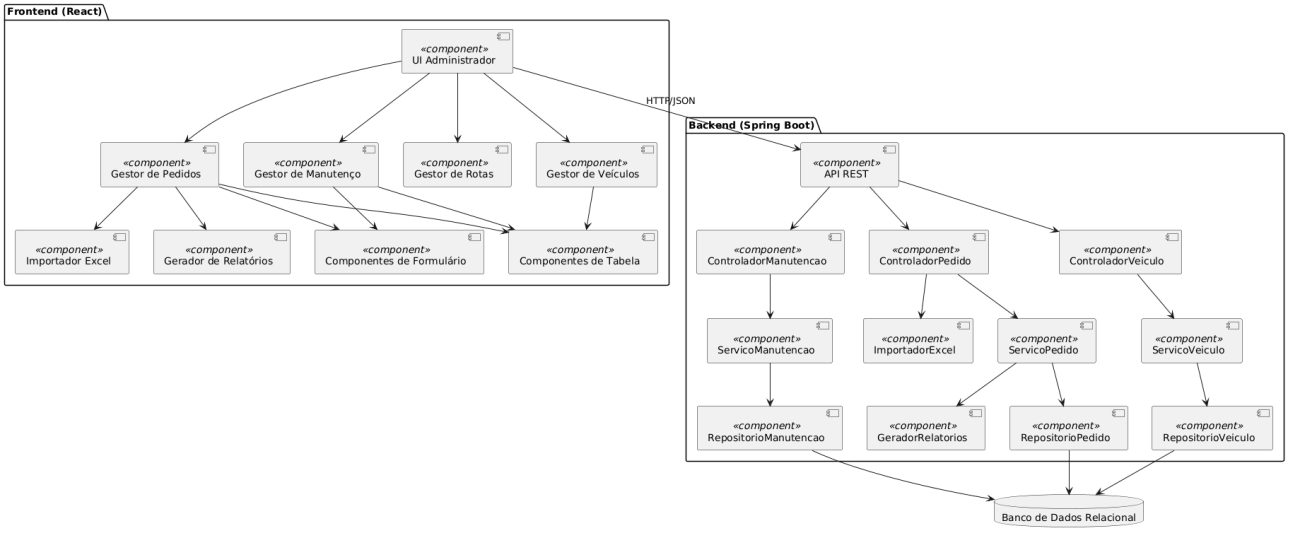
# 3. Modelos de Projeto

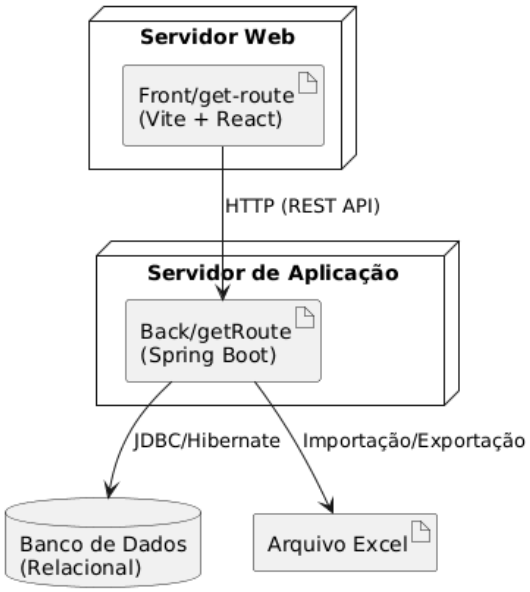
## 3.1 Arquitetura




Pode ser descrita com um diagrama apropriado da UML

## 3.2 Diagrama de Componentes e Implantação.



## 3.3 Diagrama de Classes

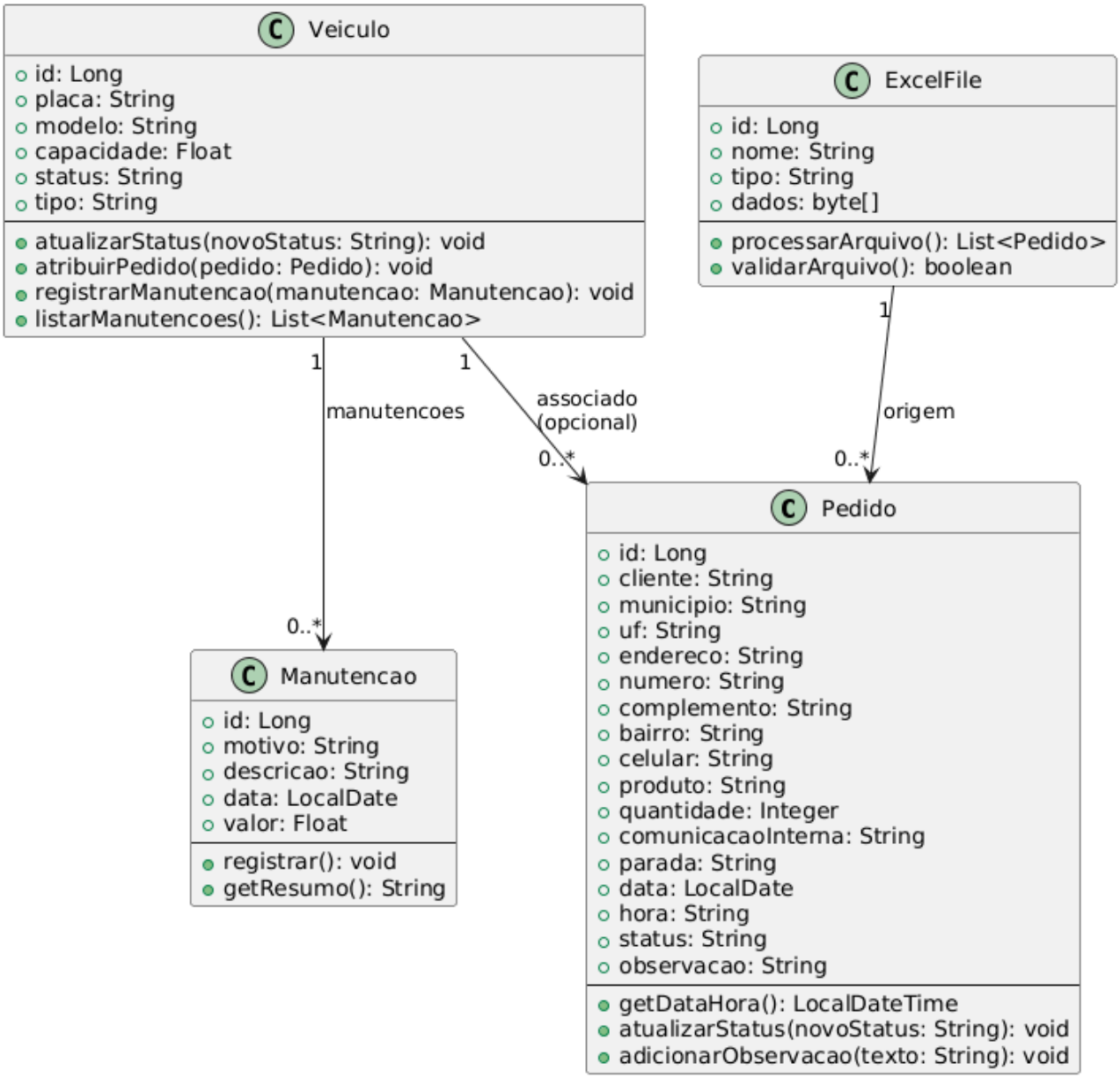
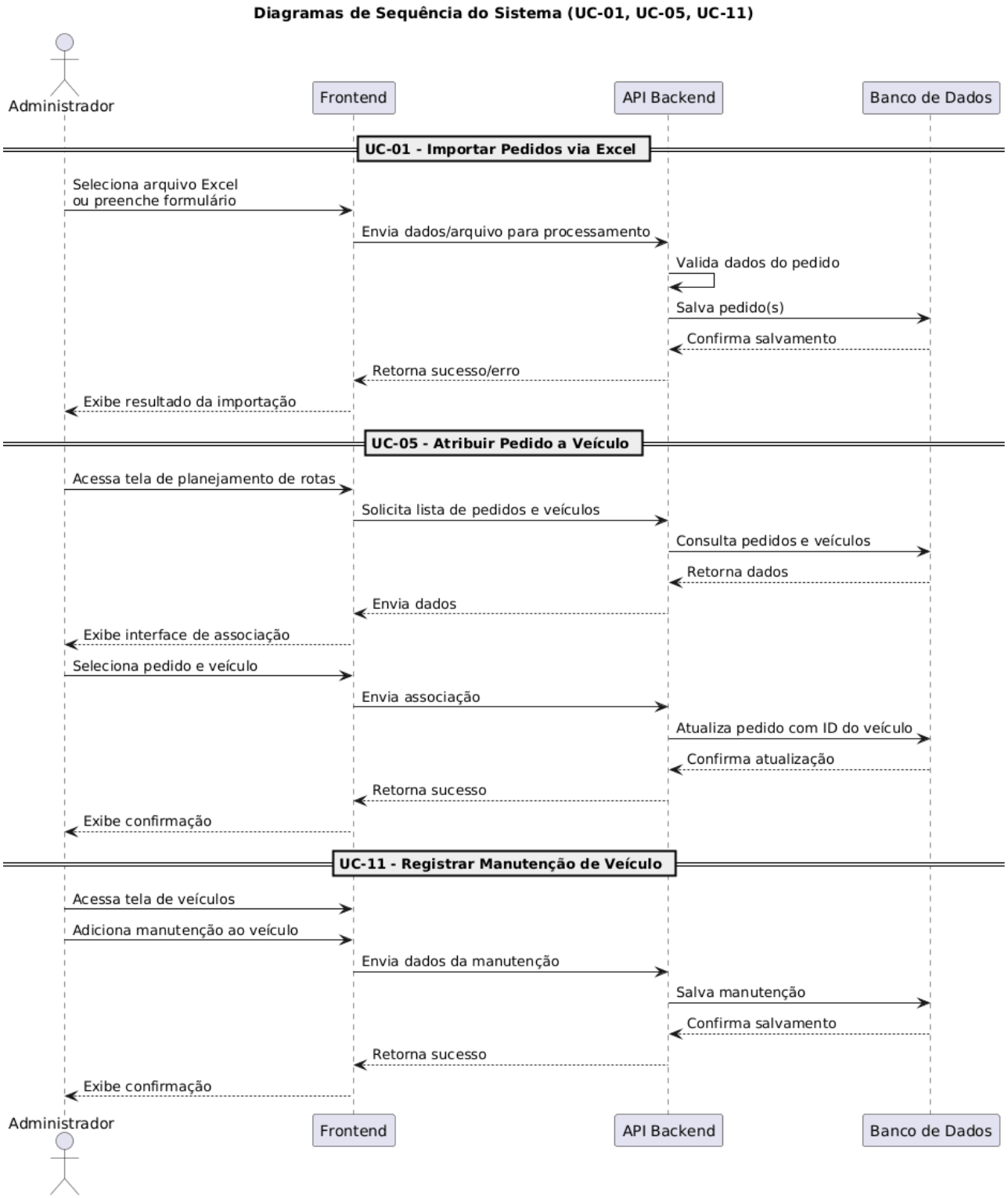


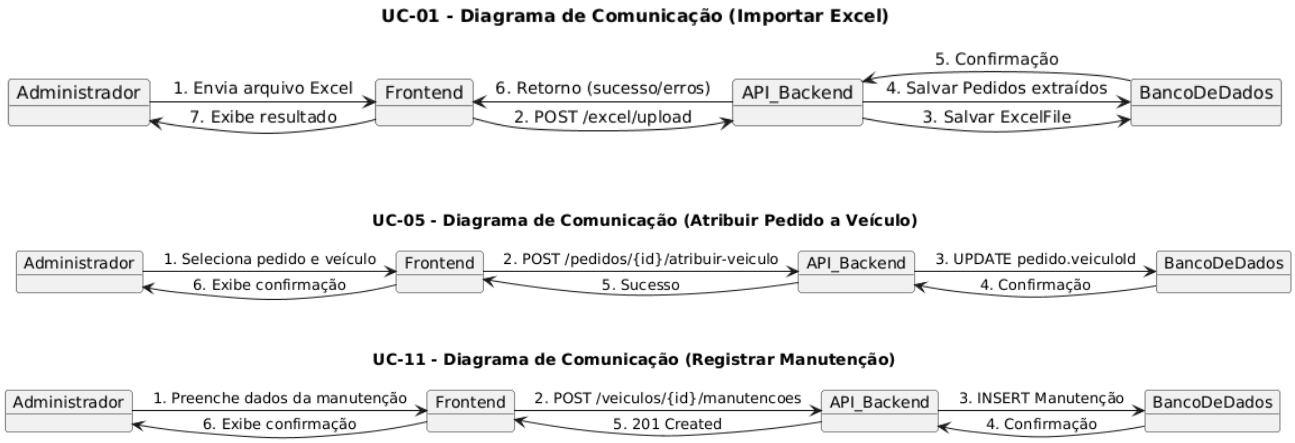

Diagrama de classes do sistema.

## 3.4 Diagramas de Sequência

Diagramas de sequência para realização de casos de uso.

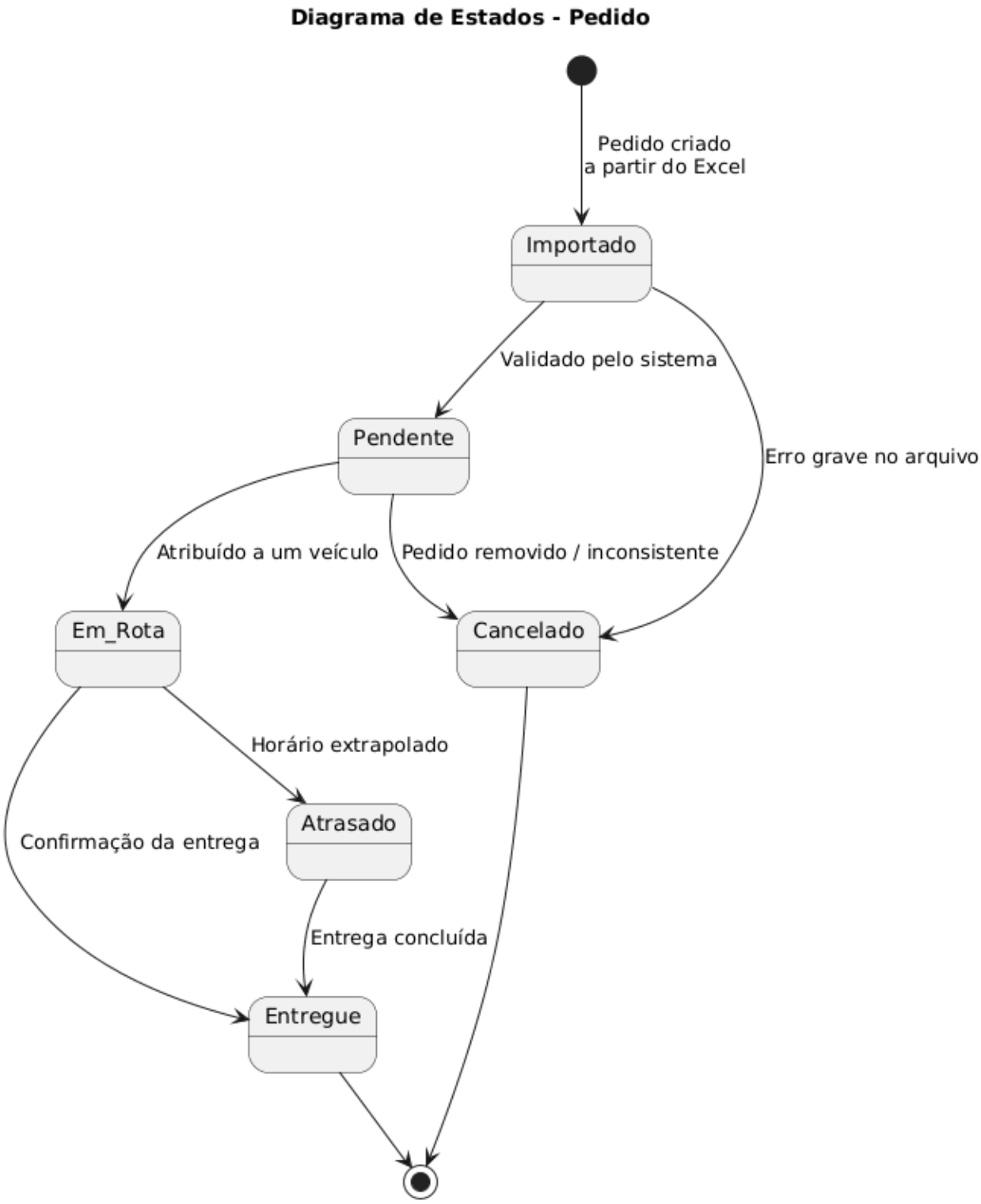

Diagramas de Sequência do Sistema (UC-01, UC-05, UC-11)
Administrador
Frontend
API Backend
Banco de Dados
UC-01 - Importar Pedidos via Excel
Seleciona arquivo Excel ou preenche formulário
Envia dados/arquivo para processamento
Valida dados do pedido
Salva pedido(s)
Confirma salvamento
Retorna sucesso/erro
Exibe resultado da importação
UC-05 - Atribuir Pedido a Veículo
Acessa tela de planejamento de rotas
Solicita lista de pedidos e veículos
Consulta pedidos e veículos
Retorna dados
Envia dados
Exibe interface de associação
Seleciona pedido e veículo
Envia associação
Atualiza pedido com ID do veículo
Confirma atualização
Retorna sucesso
Exibe confirmação
UC-11 - Registrar Manutenção de Veículo
Acessa tela de veículos
Adiciona manutenção ao veículo
Envia dados da manutenção
Salva manutenção
Confirma salvamento
Retorna sucesso
Exibe confirmação
Administrador
Frontend
API Backend
Banco de Dados

## 3.5 Diagramas de Comunicação




Diagramas de comunicação para realização de casos de uso.

## 3.6 Diagramas de Estados




Diagramas de estados do sistema.

# 4. Modelos de Dados

Esta seção apresenta o modelo de dados utilizado no sistema, incluindo o esquema do banco e as principais estratégias de mapeamento objeto–relacional adotadas com JPA/Hibernate.

---

### 4.1 Esquema do Banco de Dados

O banco segue o modelo relacional e inclui as seguintes tabelas principais:

- **veiculo** — dados de veículos utilizados nas entregas.
- **manutencao** — registros de manutenção associados a um veículo.
- **excel_file** — informações e bytes de arquivos Excel importados.
- **pedido** — pedidos extraídos dos arquivos importados.
- **funcionario** e **adicional** — dados de funcionários e adicionais financeiros.
- **relatorio** — histórico de relatórios gerados.

Os relacionamentos relevantes são:

- Veículo **1:N** Manutenções
- ExcelFile **1:N** Pedidos
- Veículo **1:N** Pedidos
- Funcionário **1:N** Adicionais

Todas as chaves estrangeiras preservam a integridade referencial.

---

### 4.2 Estratégias de Mapeamento (JPA ↔ Relacional)

Principais estratégias utilizadas:

**Tipos de Dados**

- `LocalDate` → `DATE`
- `LocalDateTime` → `TIMESTAMP`
- `BigDecimal` → `NUMERIC` (valores monetários)
- `byte[]` → `BYTEA`/BLOB

**Chaves e Relacionamentos**

- IDs gerados com `GenerationType.IDENTITY`.
- `@ManyToOne` e `@OneToMany` com **LAZY** para evitar carregamento excessivo.
- Cascade usado apenas quando necessário (evitando exclusão em cadeia).

**Serialização e DTOs**

- Uso de `@JsonIgnore` para evitar ciclos.
- Métodos derivados (como `getDataHora()`) marcados com `@Transient`.
- API utiliza DTOs, não as entidades JPA diretamente.

**Auditoria e Transações**

- Campos `created_at` e `updated_at` preenchidos automaticamente.
- Operações complexas (como importação de Excel) executadas dentro de `@Transactional`.