Documentação de Projeto

para o sistema

Aluguel de Carros

Versão 1.0

Projeto de sistema elaborado pelo(s) aluno(s) Gabriel Nogueira Vieira Resende
como parte da disciplina **Projeto de Software**.

16/11/2025

Tabela de Conteúdo

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Lançamento	16/11/2025	N.A	1.0

1. Introdução

Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o sistema **Aluguel de Carros**. 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.

2. Modelos de Usuário e Requisitos

2.1 Descrição de Atores OK

- **Clientes:** Representam pessoas físicas que desejam alugar um Automóvel e que precisam realizar um cadastro prévio para tal. Devem fornecer seus dados (RG, CPF, nome, endereço, profissão, entidade empregadora e até três fontes de rendimento) para realizar seu cadastro, pode criar e cancelar pedidos de aluguel, verificar o status de um pedido e cancelá-lo antes da aprovação final.
- **Agentes:** Representação genérica das entidades que não podem criar pedidos, mas atuam nos processos relacionados aos pedidos criados por clientes. Realizam avaliações financeiras, podem alterar dados dos pedidos antes de serem aprovados, gerenciar o catálogo de automóveis e dar o parecer positivo após análise. Tipos especializados de agente incluem:
    - **Bancos:** Representa instituições financeiras que participam do processo de locação. Herdam todas as funções de Agentes genéricos e podem ser registrados como proprietários de veículos, além de poder conceder contratos de crédito.
    - **Empresas:** Representa empresas de aluguel ou qualquer outra empresa envolvida no processo de locação. Herda todas as funções de Agentes e pode ser registrada como proprietária de veículos.

2.2 Modelo de Casos de Uso

- UC1: Consultar pedido
- UC2: Cancelar pedido
- UC3: Efetuar pedido de aluguel
- UC4: Modificar pedido
- UC5: Gerenciar contrato
- UC6: Conceder contrato de crédito
- UC7: Executar contrato
- UC8: Avaliar pedido
- UC9: Modificar pedido
- UC10: Gerenciar catálogo de automóveis

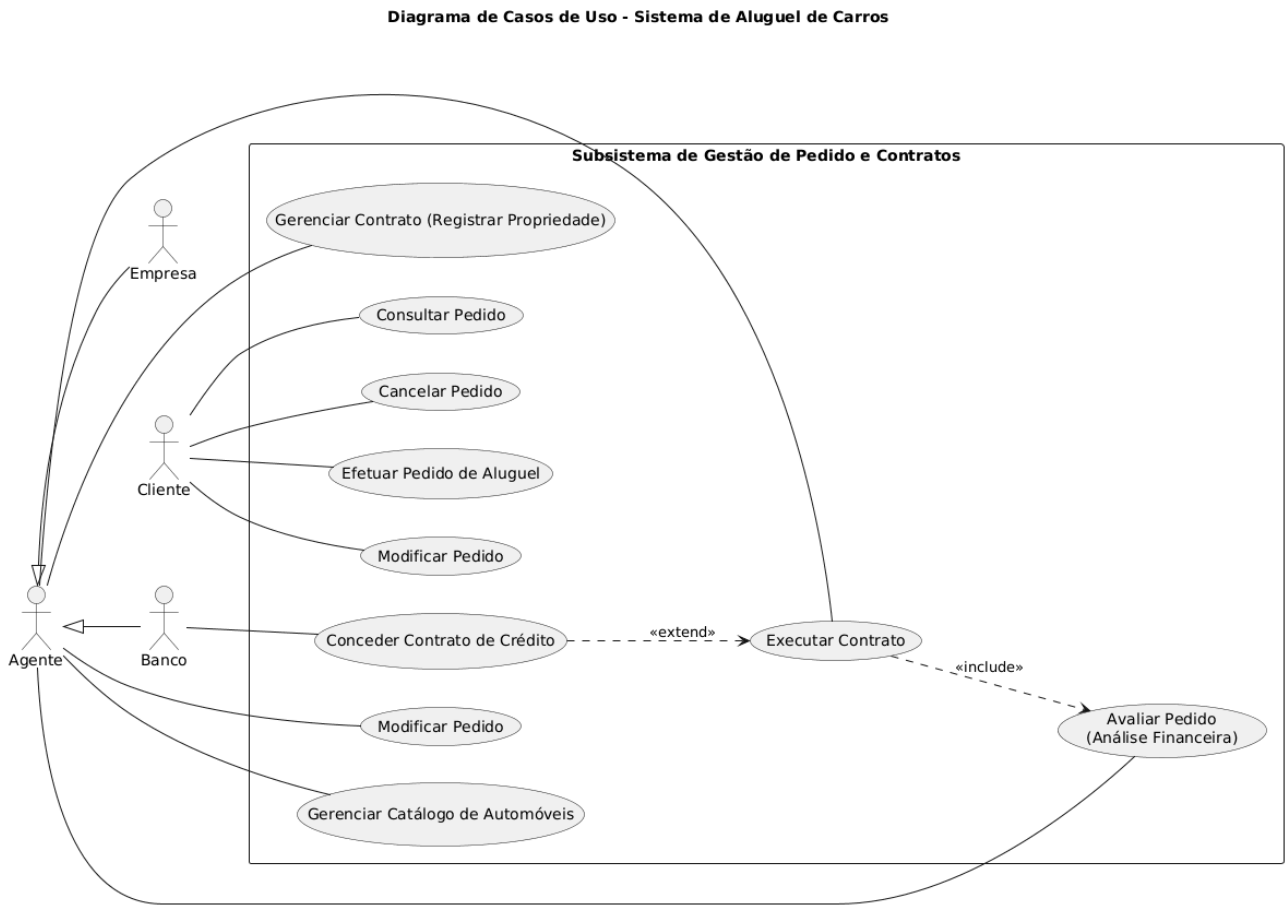
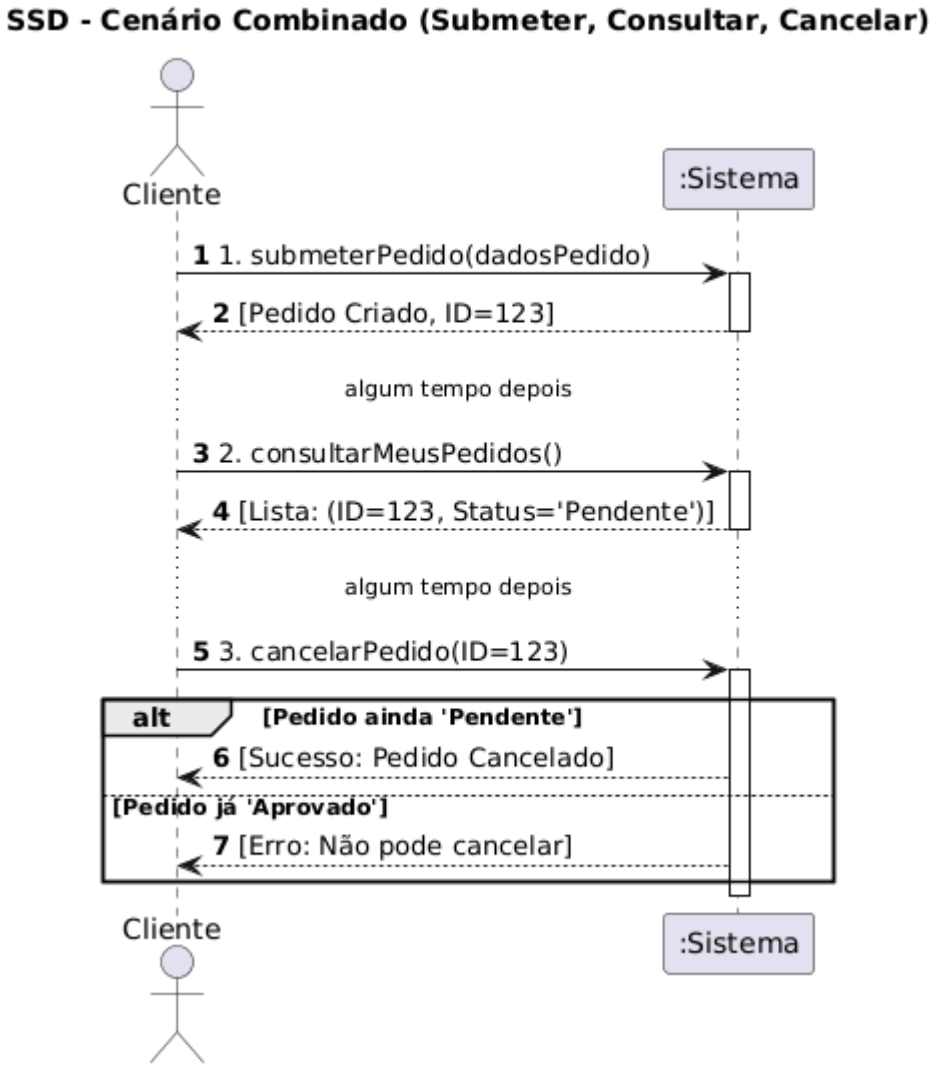


Diagrama de Casos de Uso - Sistema de Aluguel de Carros
Subsistema de Gestão de Pedido e Contratos
Empresa
Cliente
Agente
Banco
Gerenciar Contrato (Registrar Propriedade)
Consultar Pedido
Cancelar Pedido
Efetuar Pedido de Aluguel
Modificar Pedido
Conceder Contrato de Crédito
«extend»
Executar Contrato
«include»
Avaliar Pedido
(Análise Financeira)
Modificar Pedido
Gerenciar Catálogo de Automóveis

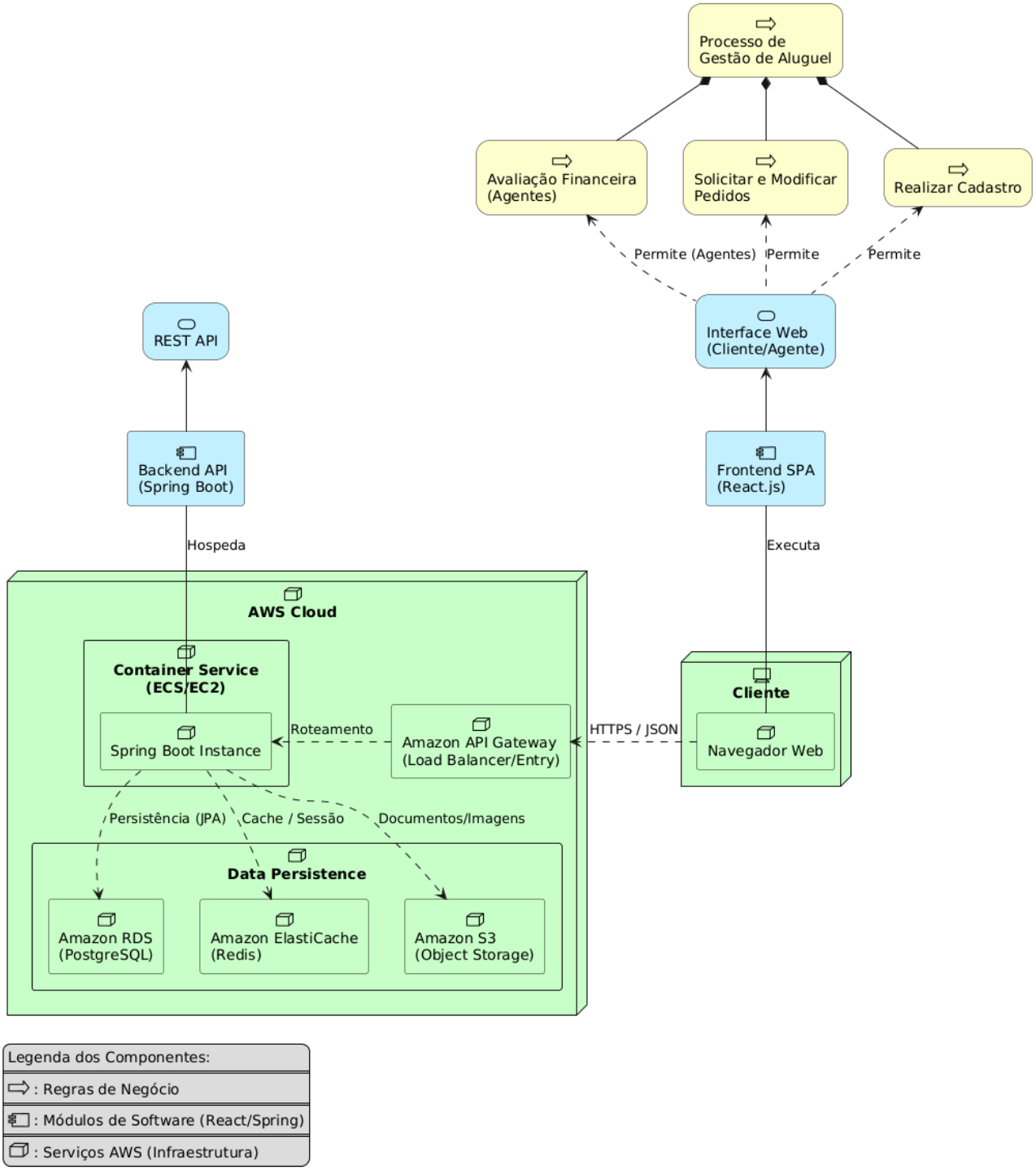
2.3 Diagrama de Sequência do Sistema



Contrato	
Operação	Ciclo de vida do pedido
Referências cruzadas	UC1, UC2, UC3
Pré-condições	Cliente deve estar logado no sistema
Pós-condições	Cliente tem o status do seu pedido alterado para Cancelado

3. Modelos de Projeto

3.1 Arquitetura



3.2 Diagrama de Componentes e Implantação.

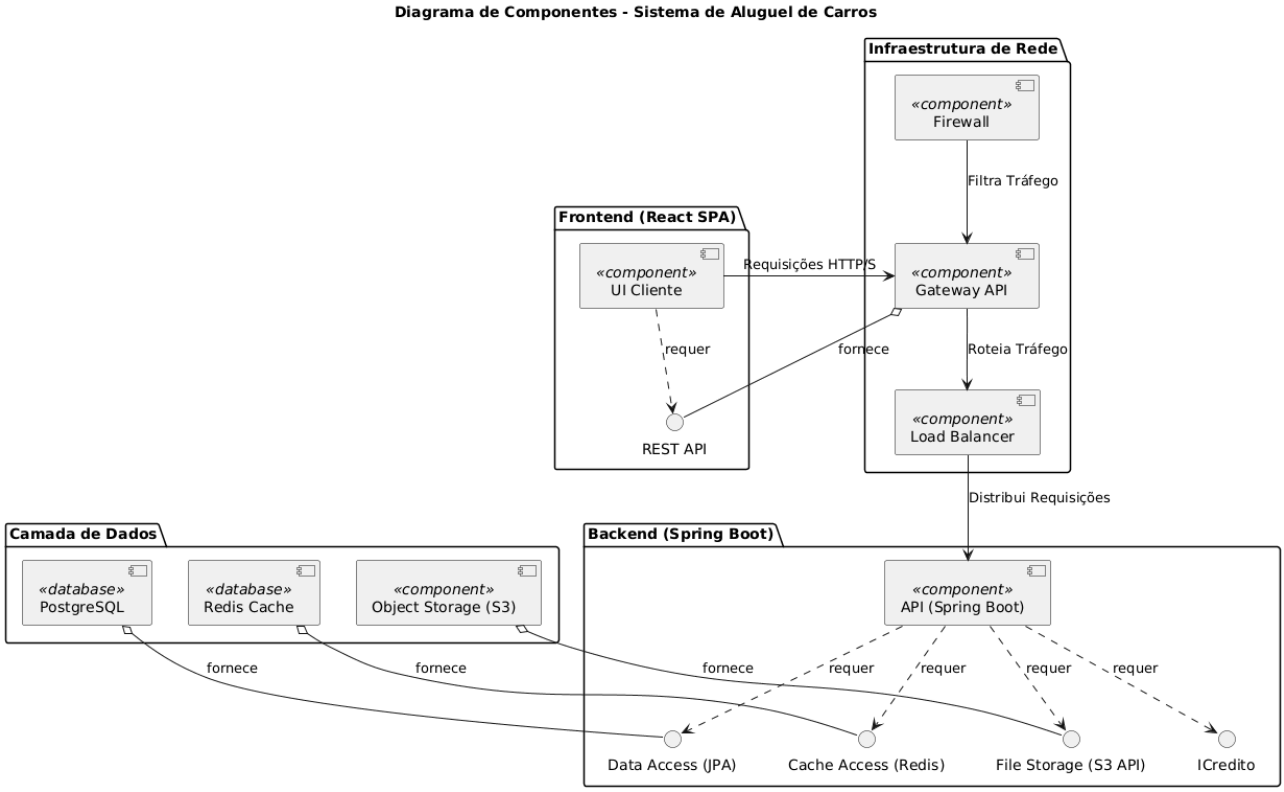
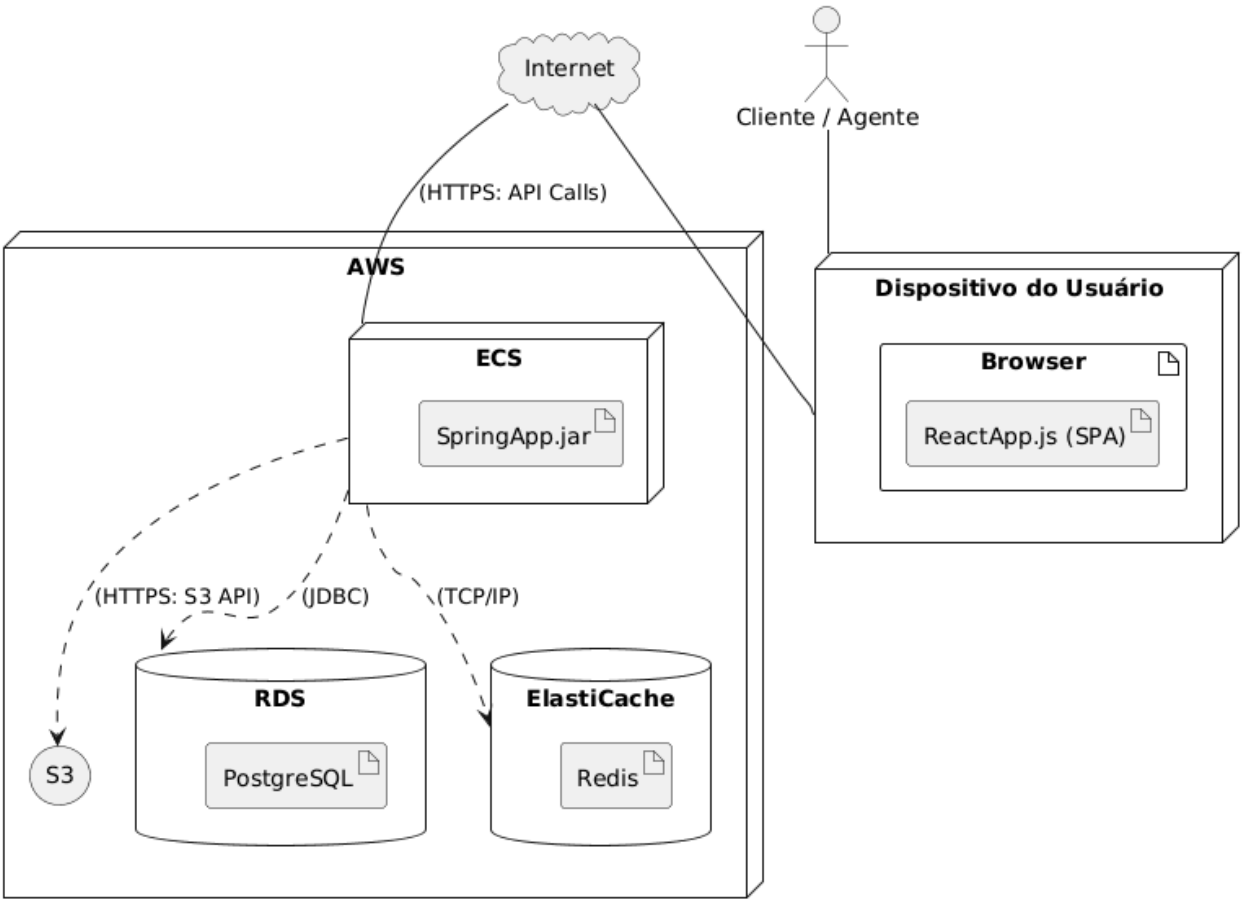
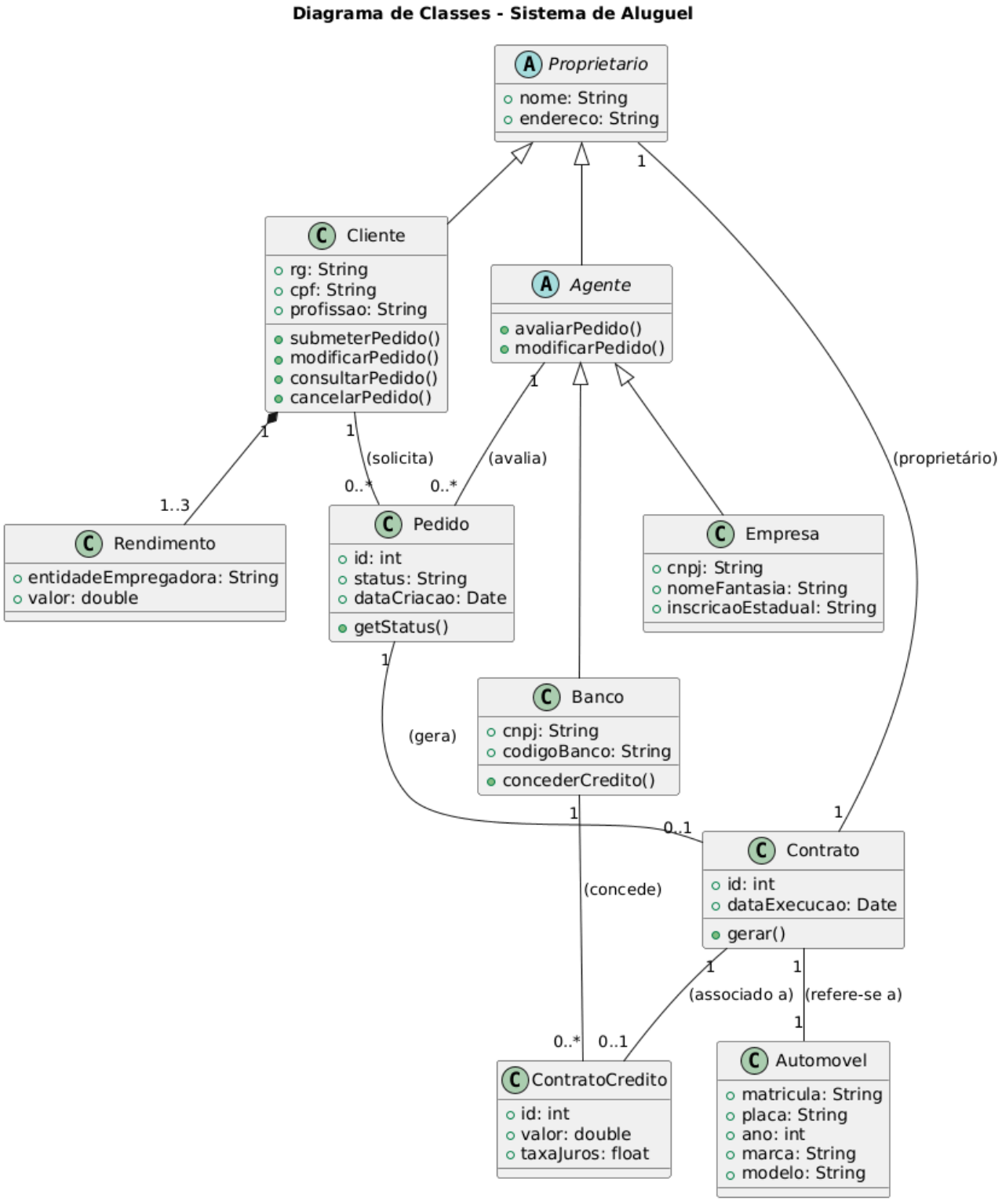


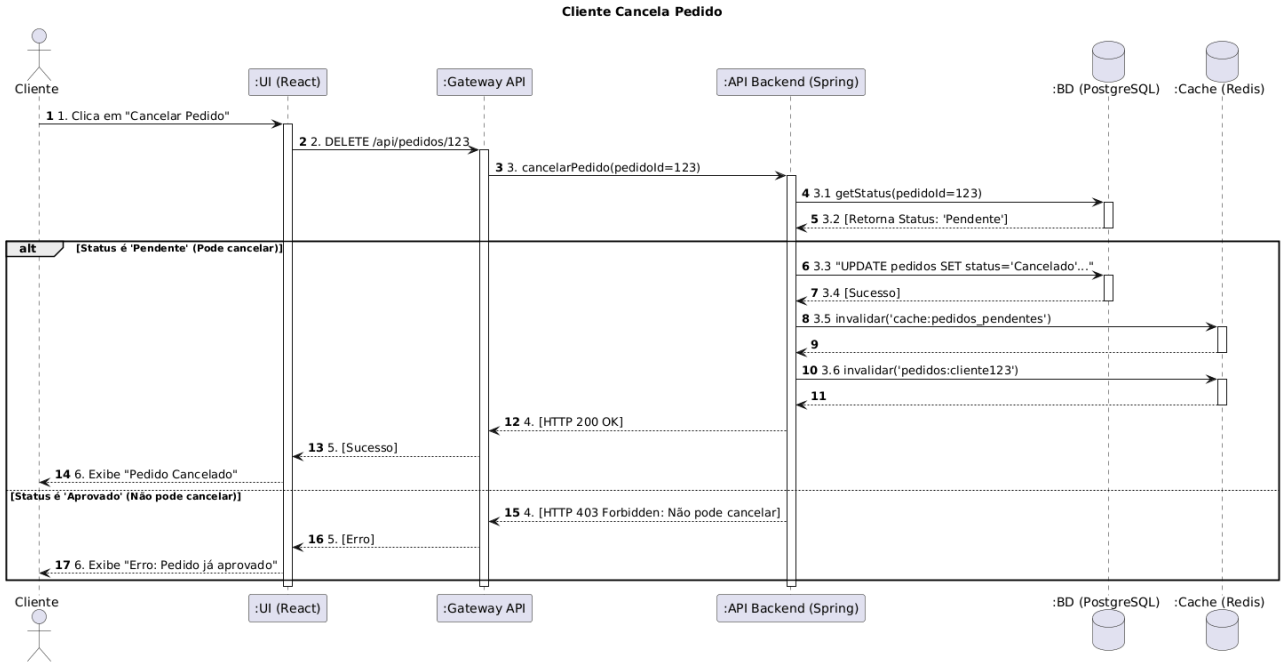
Diagrama de Implantação - Sistema de Aluguel de Carros

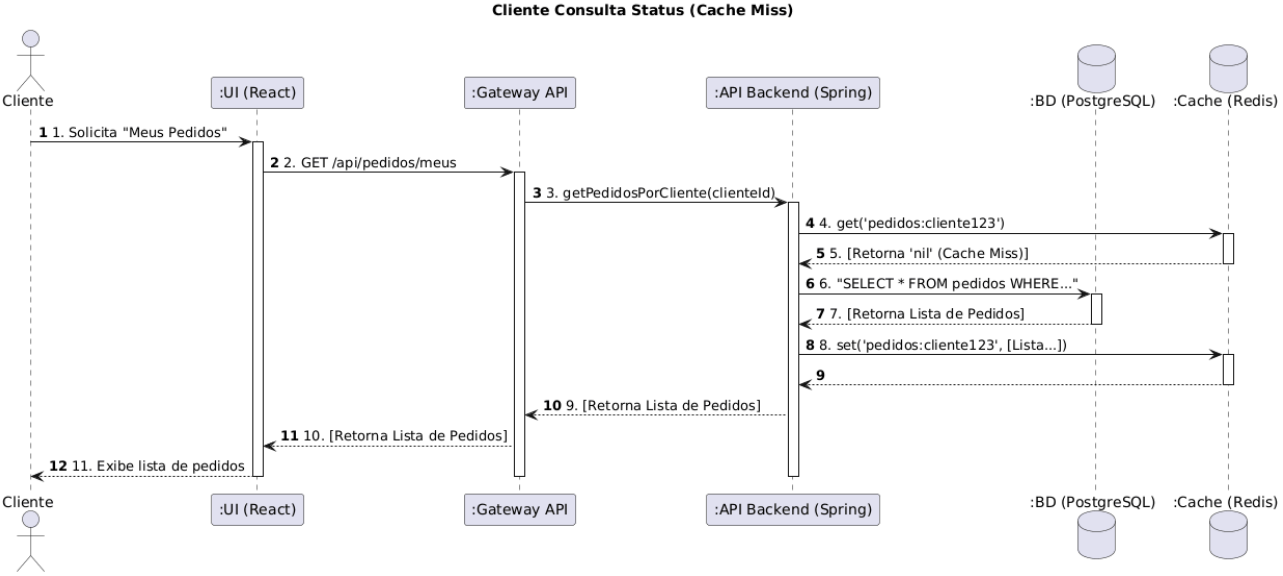


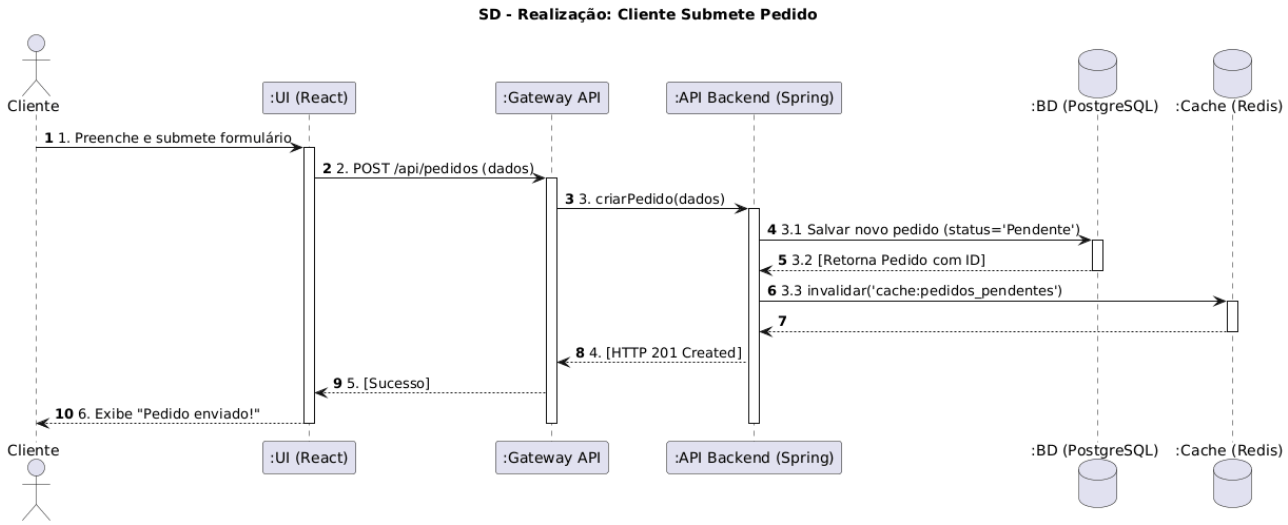
3.3 Diagrama de Classes



3.4 Diagramas de Sequência







3.5 Diagramas de Comunicação

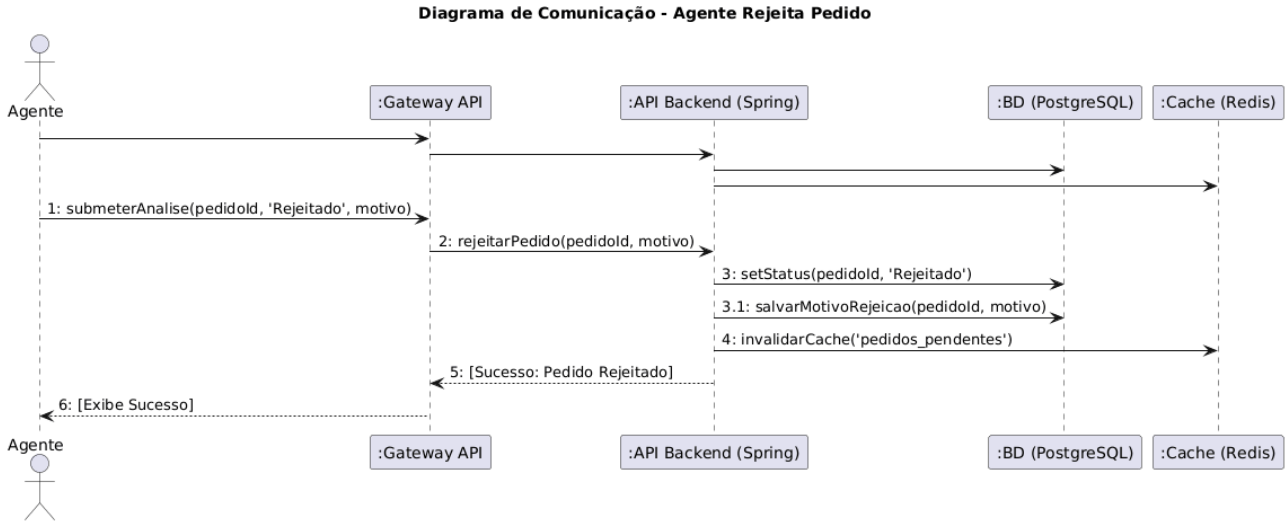


Diagrama de Comunicação - Cliente Consulta Status (Cache Hit)

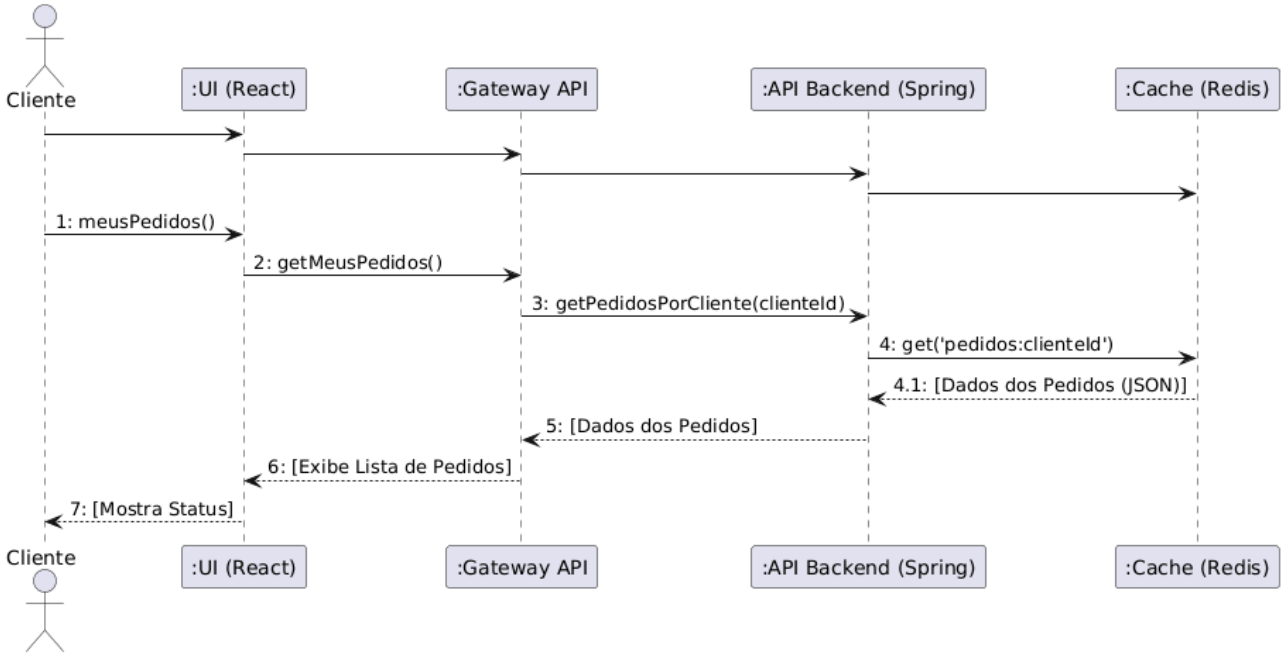
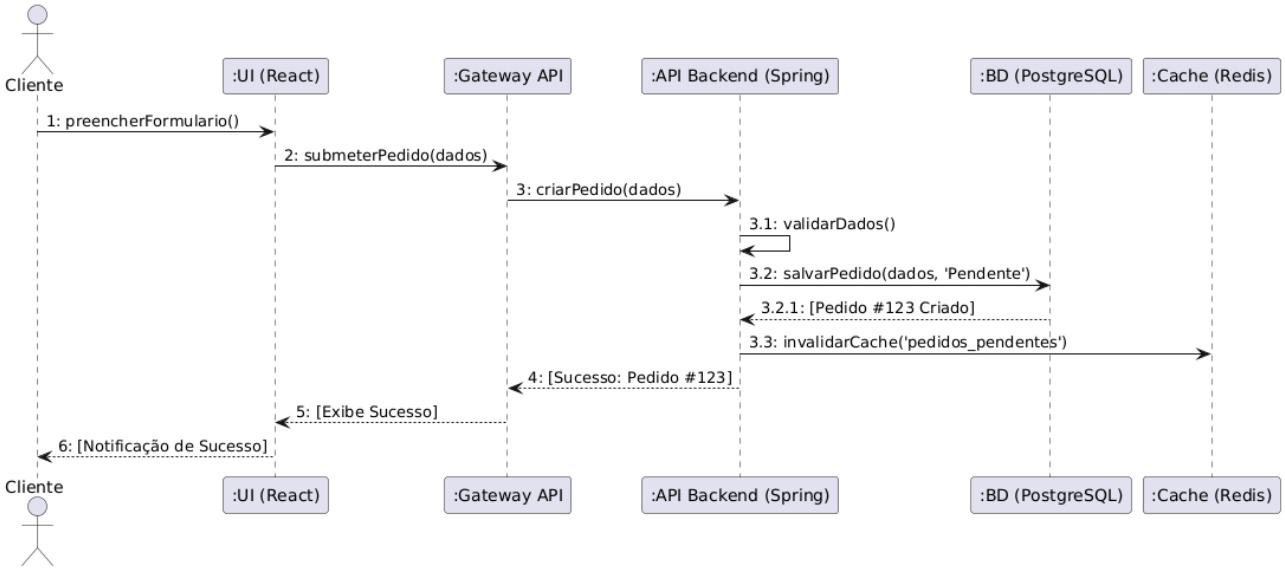
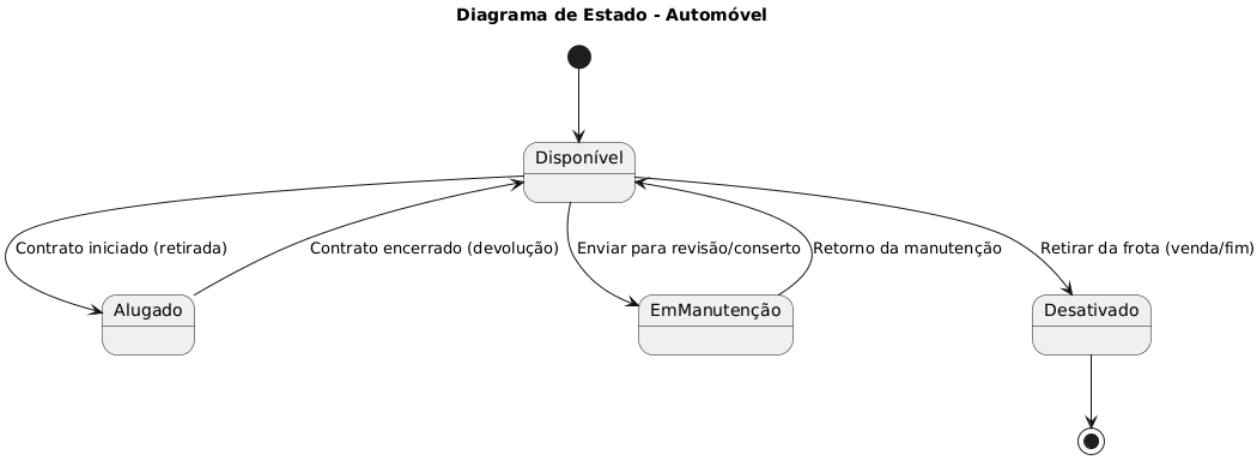
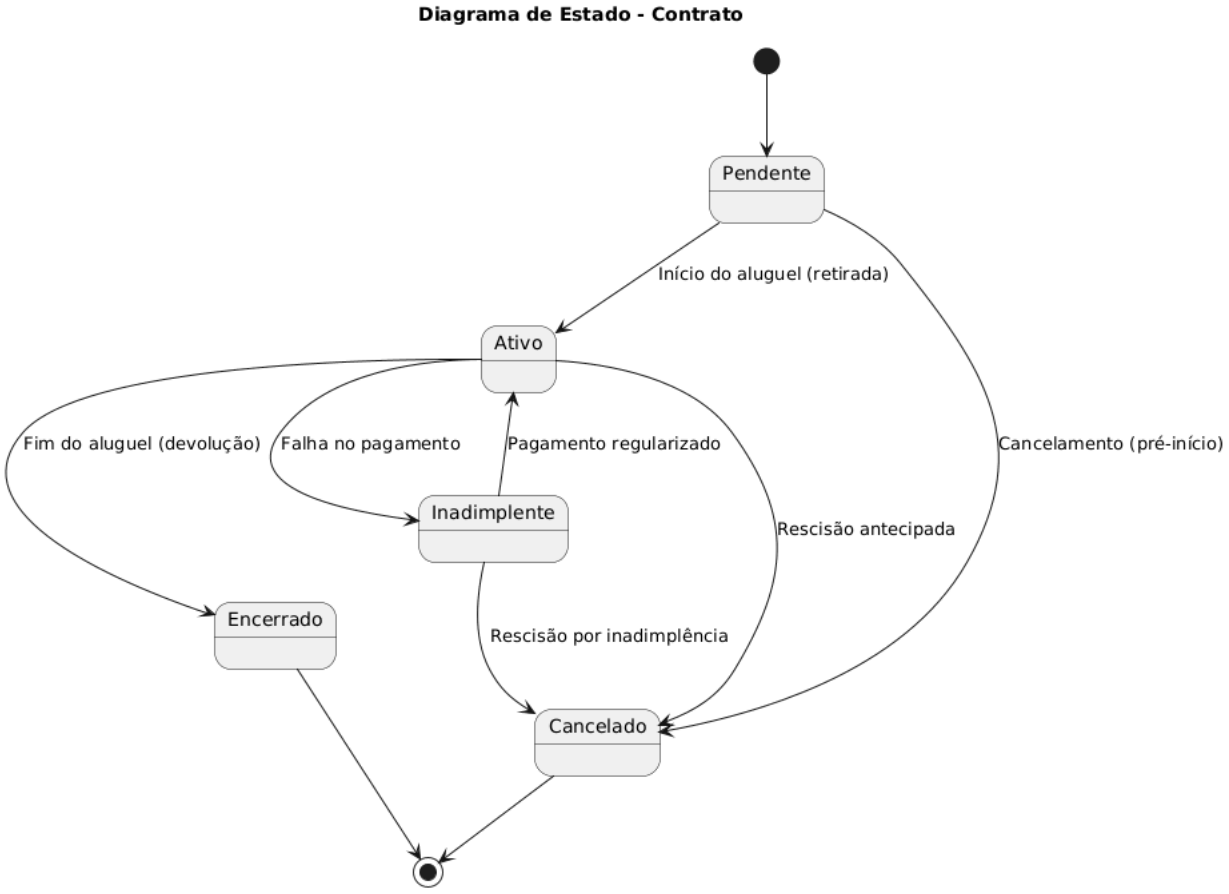


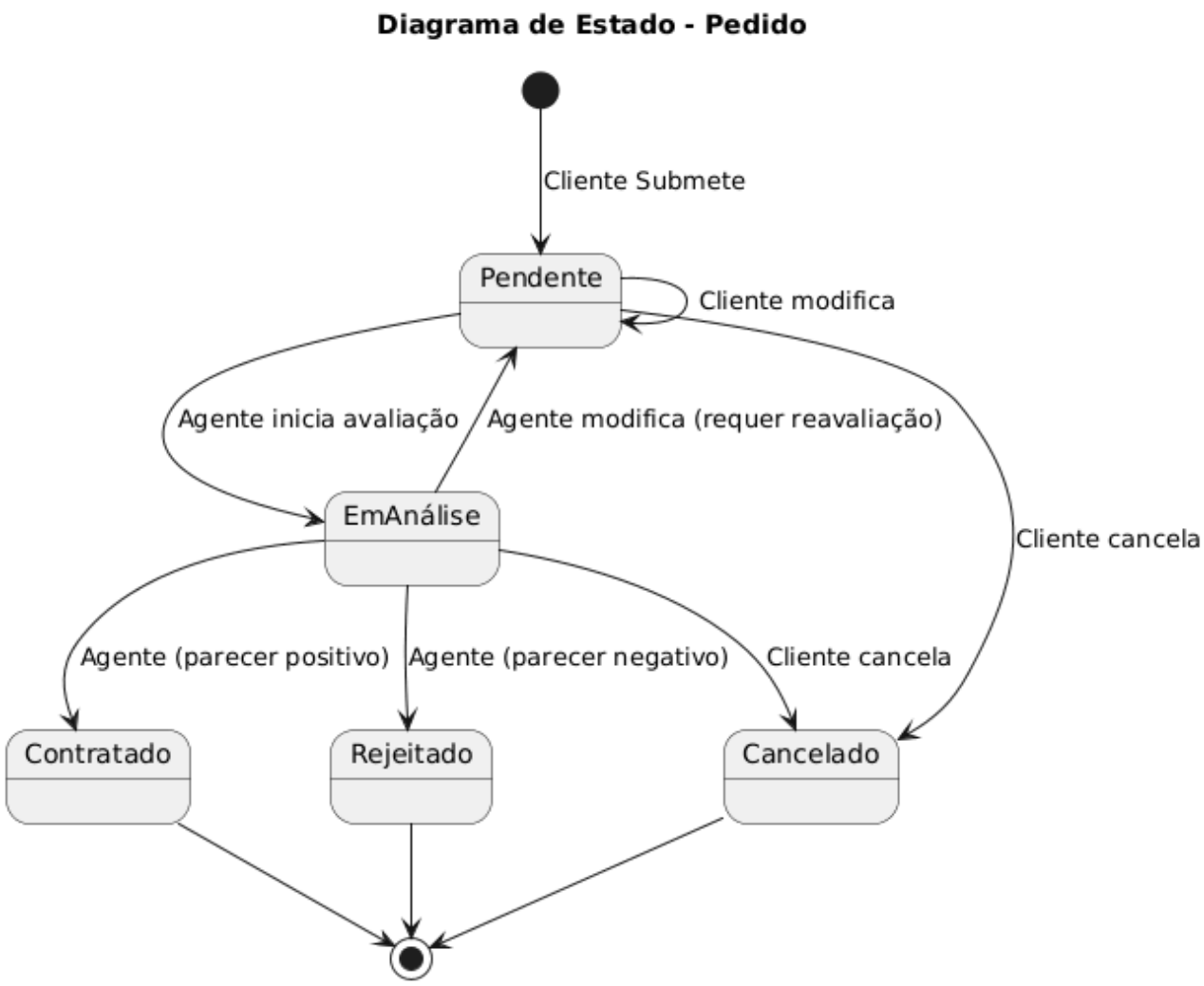
Diagrama de Comunicação - Cliente Submete um Pedido



3.6 Diagramas de Estados







4. Modelos de Dados

Esta seção descreve as estratégias usadas para mapear o Diagrama de Classes (representação de objetos) para o esquema do PostgreSQL (representação não-objeto/relacional).

Em um *stack* com Spring Boot, essas estratégias seriam implementadas usando anotações do Spring Data JPA (ex: `@Entity`, `@Inheritance`, `@OneToMany`).

1. Mapeamento de Classes para Tabelas (1:1 Básico)

A estratégia mais simples é o mapeamento direto de uma classe para uma tabela.

- **Objeto:** Classe `Automovel`
- **Não-Objeto:** Tabela `AUTOMOVEL`
- **Estratégia:** Cada atributo da classe (`placa`, `marca`, `ano`) é mapeado para uma coluna na tabela com o tipo de dado correspondente.

- `Automovel` -> `AUTOMOVEL`
- `Pedido` -> `PEDIDO`
- `Contrato` -> `CONTRATO`

2. Mapeamento de Associações (Relacionamentos)

Os relacionamentos entre objetos são representados por Chaves Estrangeiras (Foreign Keys) no modelo relacional.

a. Associação One-to-Many (1..N)

- **Objeto:** `Cliente "1" -- "0..*" Pedido` (Um Cliente tem muitos Pedidos).
- **Não-Objeto:** Tabela `PEDIDO`.
- **Estratégia:** A chave estrangeira é colocada na tabela do lado "Muitos" (N). A tabela `PEDIDO` recebe uma coluna `FK_Cliente` que referencia a `PK_Cliente` da tabela `CLIENTE`.

b. Associação One-to-One (1..1)

- **Objeto:** `Pedido "1" -- "0..1" Contrato` (Um Pedido gera um Contrato).
- **Não-Objeto:** Tabela `CONTRATO`.
- **Estratégia:** A chave estrangeira é colocada na tabela "filha" ou dependente. A tabela `CONTRATO` recebe uma coluna `FK_Pedido`. Para garantir que a relação é 1-para-1 (e não 1-para-N), essa coluna `FK_Pedido` deve ter uma restrição `UNIQUE`.

3. Mapeamento de Composição (Entidade Fraca)

- **Objeto:** `Cliente "1" *-- "1..3" Rendimento` (Um Rendimento só existe como parte de um Cliente).
- **Não-Objeto:** Tabela `RENDIMENTO`.
- **Estratégia:** A tabela `RENDIMENTO` é uma "entidade fraca". Ela deve ter uma chave estrangeira `FK_Cliente` que aponta para `CLIENTE`. Esta FK é `NOT NULL` e deve ter uma regra de `ON DELETE CASCADE`, garantindo que se o `CLIENTE` for deletado, seus `RENDIMENTOS` (que não podem existir sozinhos) também sejam.

4. Mapeamento de Herança (O Ponto Crítico)

Este é o desafio mais complexo do ORM. Temos uma hierarquia de herança: `Proprietario` é a classe-mãe de `Cliente` e `Agente`, e `Agente` é a classe-mãe de `Empresa` e `Banco`.

Existem três estratégias principais:

1. **Tabela Única (Single Table):** Criar uma única tabela `PROPRIETARIO` gigante com *todas* as colunas de todos os filhos (ex: `rg`, `cpf`, `cnpj`, `codigoBanco`). Muitas colunas ficariam nulas.
2. **Tabela por Classe Concreta (Table per Concrete Class):** Criar tabelas apenas para as classes "folha" (`CLIENTE`, `EMPRESA`, `BANCO`). O problema é que atributos comuns (como `nome` e `endereco`) seriam duplicados em todas as tabelas.

3. **Tabela por Classe (Joined Subclass) - ESTRATÉGIA ESCOLHIDA.**

- **Estratégia Escolhida:** Mapeamento "Joined Subclass".
- **Descrição:** Criamos uma tabela para *cada* classe na hierarquia, refletindo o Diagrama de Classes e o DER que fizemos.
    - `PROPRIETARIO` (Tabela-Mãe): Contém a PK e atributos comuns (`nome`, `endereco`).
    - `CLIENTE` (Tabela-Filha): Contém sua PK, que é *também* uma FK apontando para `PROPRIETARIO.PK_Proprietario`. Contém apenas atributos específicos (`rg`, `cpf`).
    - `AGENTE` (Tabela-Mãe/Filha): Contém sua PK (que é FK de `PROPRIETARIO`) e seus atributos (`cnpj`).
    - `BANCO` (Tabela-Filha): Contém sua PK (que é FK de `AGENTE`) e seus atributos (`codigoBanco`).
- **Vantagem:** É a forma mais normalizada e limpa. Reflete o modelo de objetos perfeitamente, sem duplicar dados e sem colunas nulas.
- **Anotação JPA (Spring):** `@Inheritance(strategy = InheritanceType.JOINED)`

5. Diagrama de entidade relacionamento

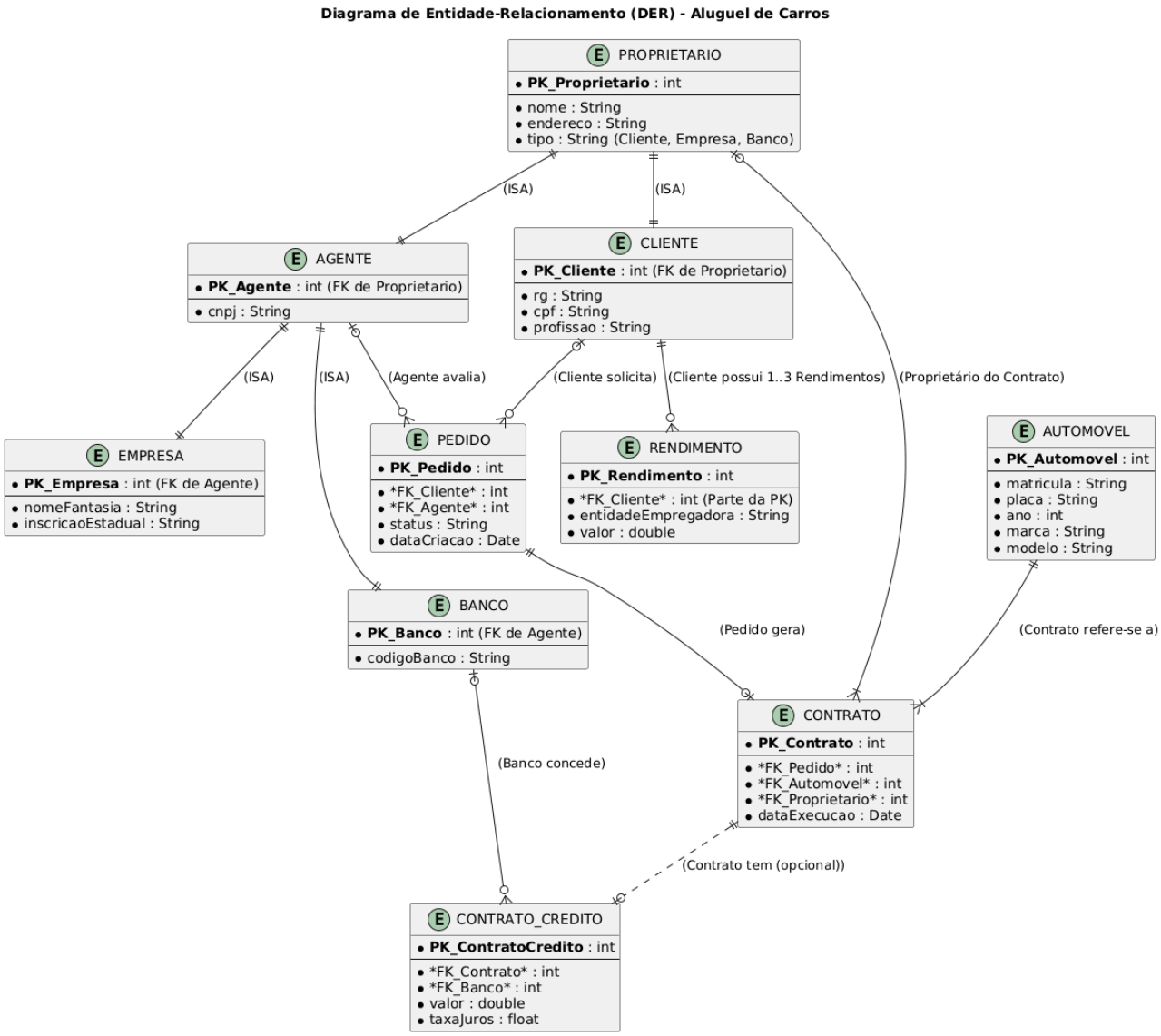


Diagrama de Entidade-Relacionamento (DER) - Aluguel de Carros
PROPRIETARIO
• PK_Proprietario : int
• nome : String
• endereco : String
• tipo : String (Cliente, Empresa, Banco)
(ISA)
(ISA)
AGENTE
• PK_Agente : int (FK de Proprietario)
• cnpj : String
CLIENTE
• PK_Cliente : int (FK de Proprietario)
• rg : String
• cpf : String
• profissao : String
(ISA)
(ISA)
(Agente avalia)
(Cliente solicita)
(Cliente possui 1..3 Rendimentos)
(Proprietário do Contrato)
EMPRESA
• PK_Empresa : int (FK de Agente)
• nomeFantasia : String
• inscricaoEstadual : String
PEDIDO
• PK_Pedido : int
• *FK_Cliente* : int
• *FK_Agente* : int
• status : String
• dataCriacao : Date
RENDIMENTO
• PK_Rendimento : int
• *FK_Cliente* : int (Parte da PK)
• entidadeEmpregadora : String
• valor : double
AUTOMOVEL
• PK_Automovel : int
• matricula : String
• placa : String
• ano : int
• marca : String
• modelo : String
BANCO
• PK_Banco : int (FK de Agente)
• codigoBanco : String
(Pedido gera)
(Contrato refere-se a)
CONTRATO
• PK_Contrato : int
• *FK_Pedido* : int
• *FK_Automovel* : int
• *FK_Proprietario* : int
• dataExecucao : Date
(Banco concede)
(Contrato tem (opcional))
CONTRATO_CREDITO
• PK_ContratoCredito : int
• *FK_Contrato* : int
• *FK_Banco* : int
• valor : double
• taxaJuros : float