Documentação de Projeto

para o sistema

AutoPark

Versão 1.0

Projeto de sistema elaborado pelo(s) aluno(s) Pedro Teodoro Castro Valadares
como parte da disciplina **Projeto de Software**.

16/11/2025

Tabela de Conteúdo

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão

1. Introdução

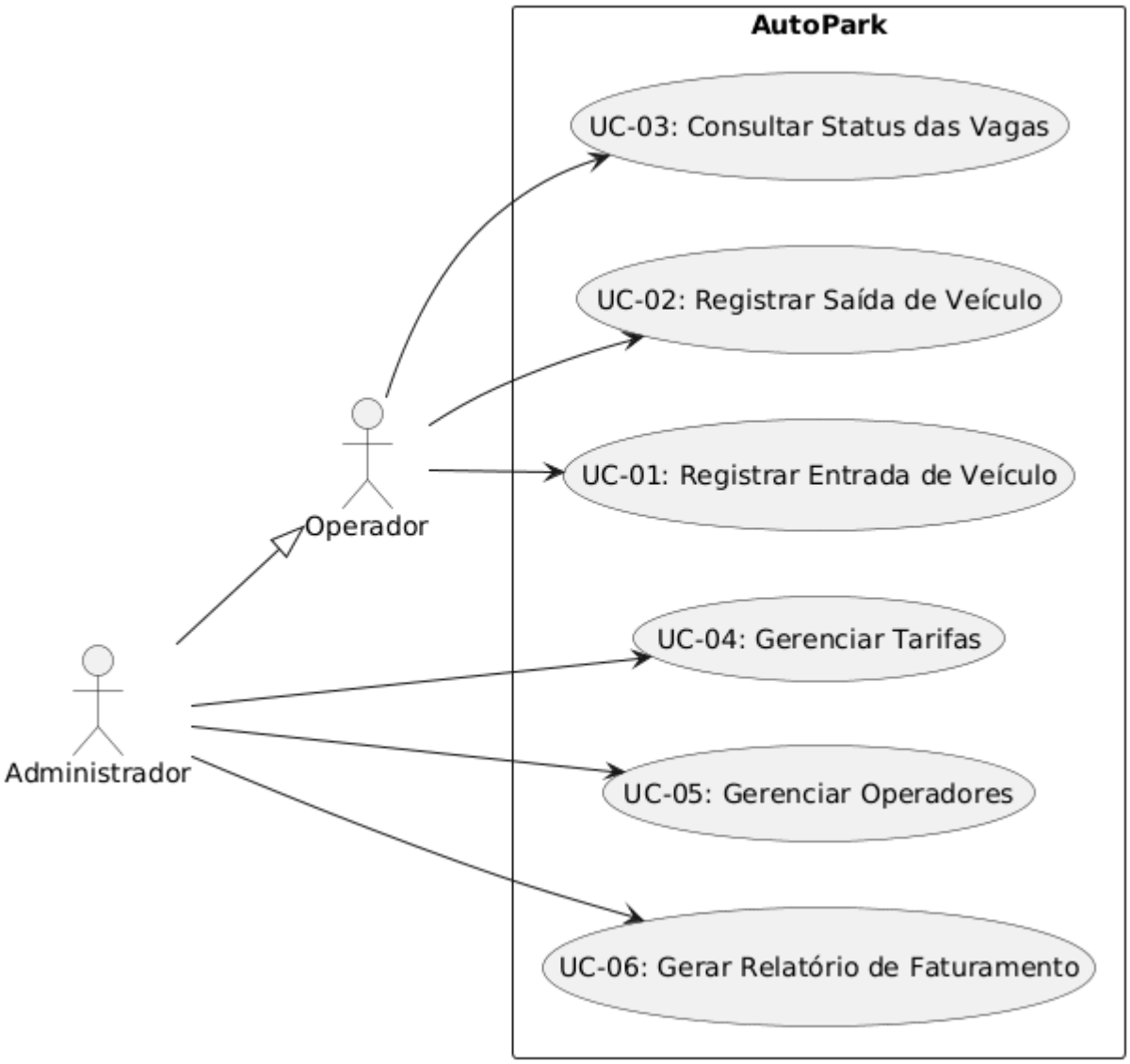
Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o sistema **AutoPark**. O sistema visa gerenciar a entrada, saída e faturamento de um estacionamento rotativo, permitindo o controle de vagas e a geração de relatórios administrativos.

2. Modelos de Usuário e Requisitos

2.1 Descrição de Atores

- **Operador (Atendente):** É o funcionário responsável por operar o sistema no dia a dia. Suas principais funções são registrar a entrada de veículos, registrar a saída (calculando o valor a ser pago) e consultar o status das vagas.
- **Administrador (Gerente):** É o responsável pela configuração e gerenciamento do sistema. Ele pode realizar todas as ações de um Operador, mas também é responsável por definir as tarifas (preço por hora), gerenciar as contas dos operadores e gerar relatórios de faturamento.

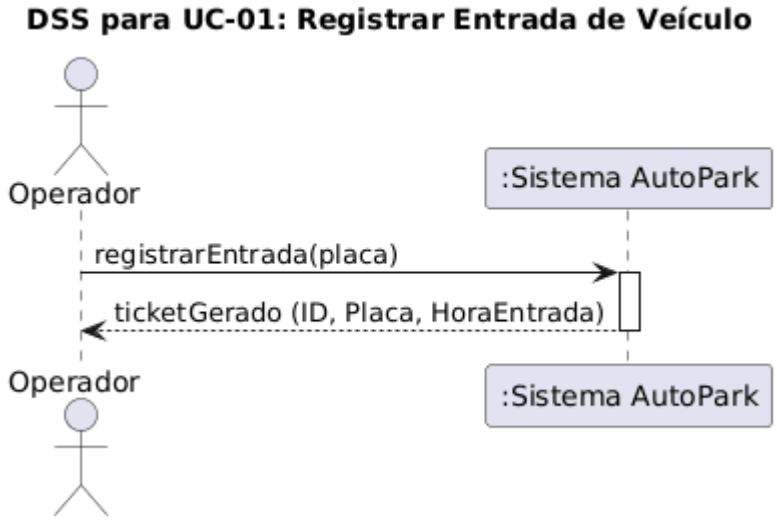
2.2 Modelo de Casos de Uso



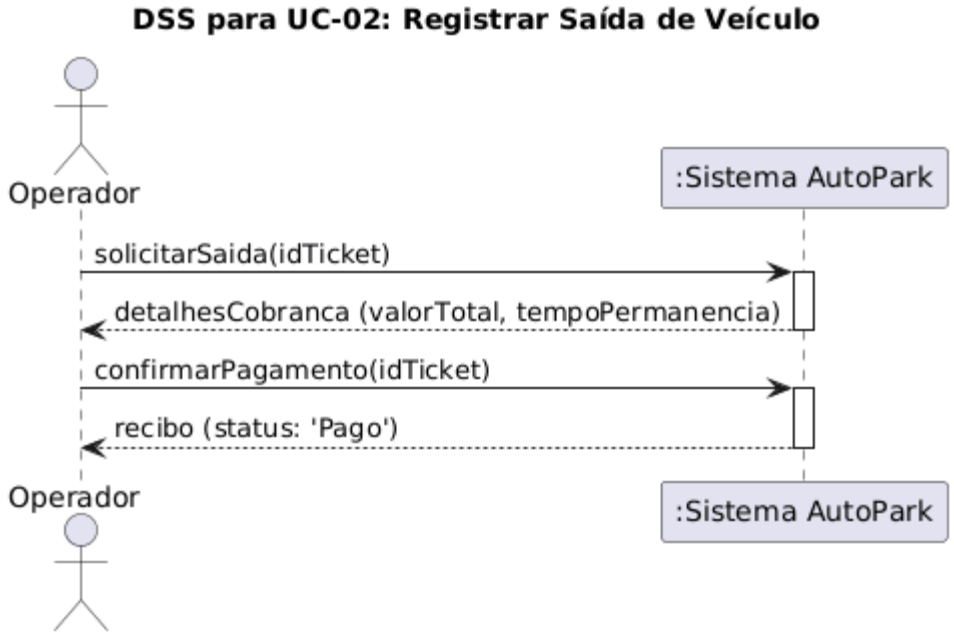
Lista de Referência de Casos de Uso (UCs):

- **UC-01:** Registrar Entrada de Veículo
- **UC-02:** Registrar Saída de Veículo
- **UC-03:** Consultar Status das Vagas
- **UC-04:** Gerenciar Tarifas
- **UC-05:** Gerenciar Operadores
- **UC-06:** Gerar Relatório de Faturamento

2.3 Diagrama de Sequência do Sistema



Contrato	CT-01
Operação	RegistrarEntrada(placa:String)
Referências cruzadas	UC-01: Registrar Entrada de Veiculo
Pré-condições	• O Operador deve estar autenticado no sistema. • Deve existir pelo menos uma vaga de estacionamento com o estado "Livre". • O veículo (identificado pela placa) não deve possuir um ticket "Aberto" no sistema.
Pós-condições	• Uma nova instância de Ticket foi criada, associada à placa. • O Ticket criado possui o estado "Aberto". • O horário de entrada (data/hora atual) foi registrado no Ticket. • Uma instância de Vaga teve seu estado alterado de "Livre" para "Ocupada". • O número total de vagas disponíveis no sistema foi decrementado. • Os dados do Ticket gerado (ID, Placa, HoraEntrada) foram retornados ao Operador.



Contrato	CT-02
Operação	solicitarSaida(idTicket: String)
Referências cruzadas	UC-02: Registrar Saída de Veículo
Pré-condições	• O Operador deve estar autenticado. • O idTicket deve corresponder a um Ticket existente no sistema. • O Ticket deve estar com o estado "Aberto". • O sistema deve ter uma Tabela de Tarifas ativa para o cálculo.
Pós-condições	• O sistema calcula o tempo total de permanência (Hora Saída - Hora Entrada). • O sistema calcula o valorTotal com base no tempo de permanência e na Tabela de Tarifas. • O valorTotal e o tempoPermanencia são retornados ao Operador. • O estado do Ticket permanece "Aberto" (aguardando pagamento).

Contrato	CT-03
Operação	confirmarPagamento(idTicket: String)

Referências cruzadas	UC-02: Registrar Saída de Veículo
Pré-condições	• O Operador deve estar autenticado. • O idTicket deve corresponder a um Ticket com o estado "Aberto". • A operação solicitarSaida (CT-02) já deve ter sido executada.
Pós-condições	• O estado do Ticket é alterado de "Aberto" para "Fechado" (ou "Pago"). • A Vaga que estava associada a este Ticket tem seu estado alterado de "Ocupada" para "Livre". • O número total de vagas disponíveis no sistema é incrementado. • Um recibo (com status "Pago") é retornado.

DSS para UC-03: Consultar Status das Vagas

Contrato	CT-04
Operação	consultarVagas()
Referências cruzadas	UC-03: Consultar Status das Vagas
Pré-condições	O Operador deve estar autenticado no sistema.

Pós-condições	• O sistema consulta o estado de todas as instâncias de Vaga. • O sistema contabiliza o número de vagas "Livres" e "Ocupadas". • O totalLivres e o totalOcupadas são retornados ao Operador.

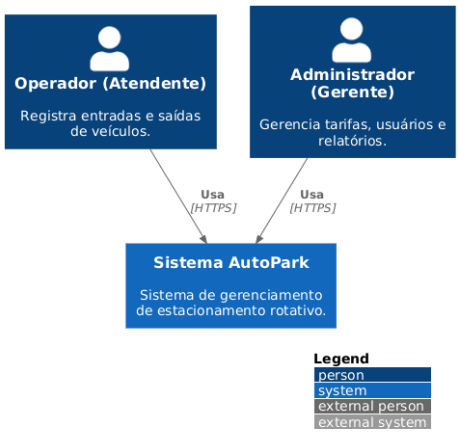
3. Modelos de Projeto

3.1 Arquitetura

O **Sistema AutoPark** é utilizado pelo **Operador** para as rotinas diárias (entrada e saída) e pelo **Administrador** para configuração e relatórios. O sistema não interage com nenhum outro sistema externo, mantendo o escopo simples.

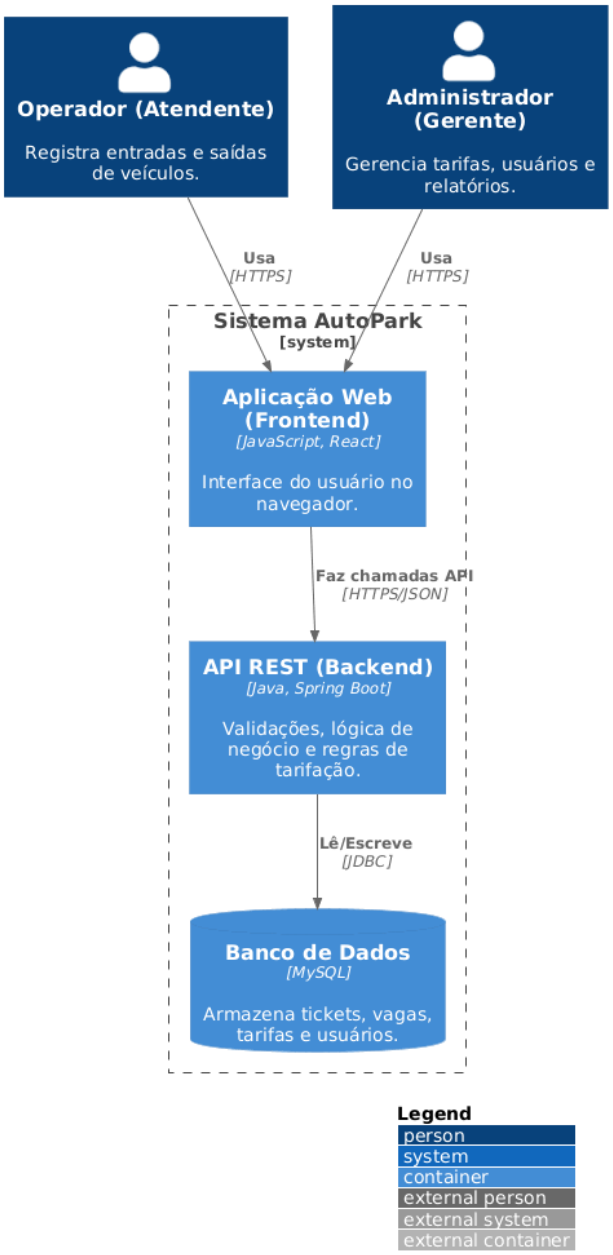
Nível 1: Diagrama de Contexto (C4)

O **Sistema AutoPark** é o centro do nosso domínio. Ele é utilizado pelo **Operador** para as rotinas diárias (entrada e saída) e pelo **Administrador** para configuração e relatórios.



O Sistema AutoPark é composto por três contêineres principais:

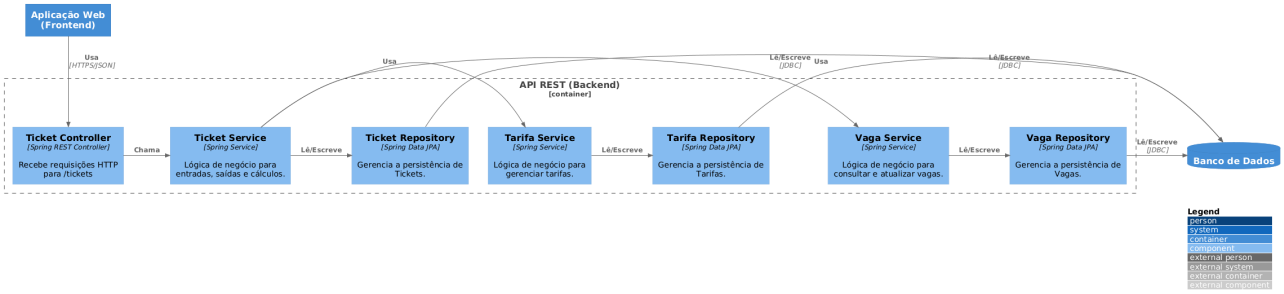
1. **Aplicação Web (Frontend):** Uma aplicação *Single-Page Application* (SPA) que roda no navegador do usuário. É a interface com a qual o Operador e o Administrador interagem.
2. **API REST (Backend):** O servidor de aplicação que contém toda a lógica de negócio (cálculo de preço, gerenciamento de vagas) e expõe os dados para o frontend.
3. **Banco de Dados (Storage):** Onde armazenamos de forma persistente os dados sobre tickets, veículos, tarifas e usuários.



3.2 Diagrama de Componentes e Implantação.

API REST (Backend) é organizada em uma arquitetura de 3 camadas (Controllers, Services, Repositories).

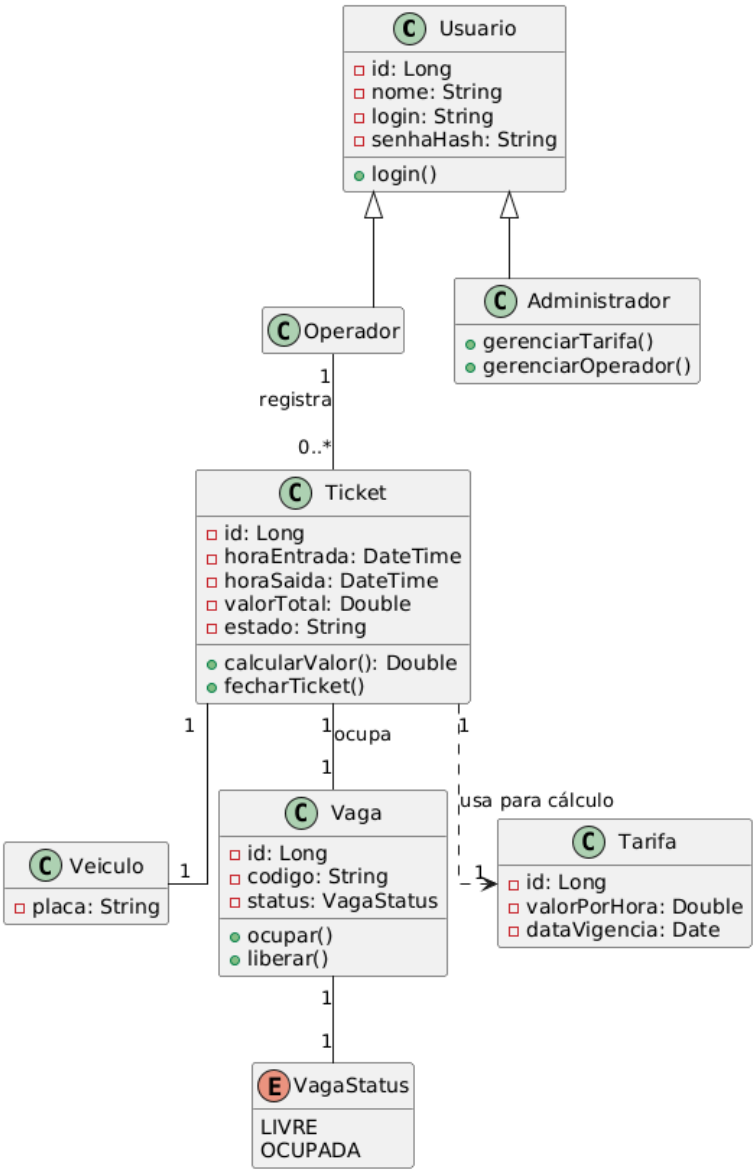
- **Controllers (Camada de Apresentação):** Recebem as requisições HTTP da Aplicação Web.
- **Services (Camada de Negócio):** Onde a lógica principal do sistema (regras de negócio, cálculos) acontece.
- **Repositories (Camada de Acesso a Dados):** Componentes responsáveis por se comunicar com o Banco de Dados.



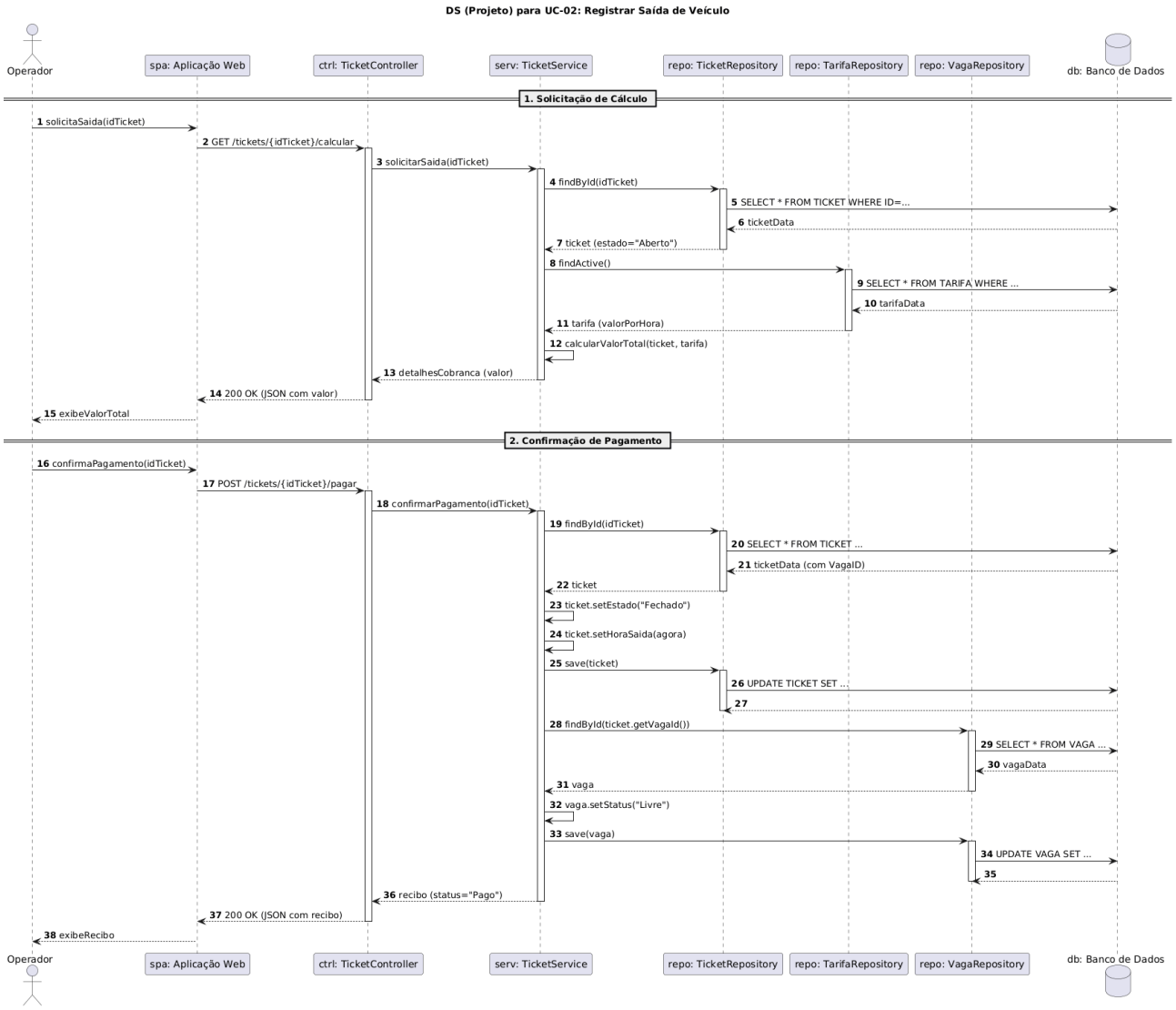
- O **Navegador do Cliente** (seja do Operador ou Administrador) executa a Aplicação Web (Frontend).
- Um **Servidor de Aplicação** (Ex: AWS EC2) executa a API REST (Backend).
- Um **Serviço Gerenciado de Banco de Dados** (Ex: AWS RDS) hospeda nosso Banco de Dados MySQL.

3.3 Diagrama de Classes

- **Usuario:** Classe base para quem usa o sistema.
- **Operador** e **Administrador:** Herdam de Usuario, representando os atores.
- **Ticket:** A entidade central. Registra a permanência de um veículo, associando-se a um Veiculo e a uma Vaga.
- **Veiculo:** Identificado unicamente pela placa.
- **Vaga:** Possui um status (Livre/Ocupada), gerenciado por um Enum VagaStatus.
- **Tarifa:** Define o valorPorHora que o Ticket usa para o cálculo.

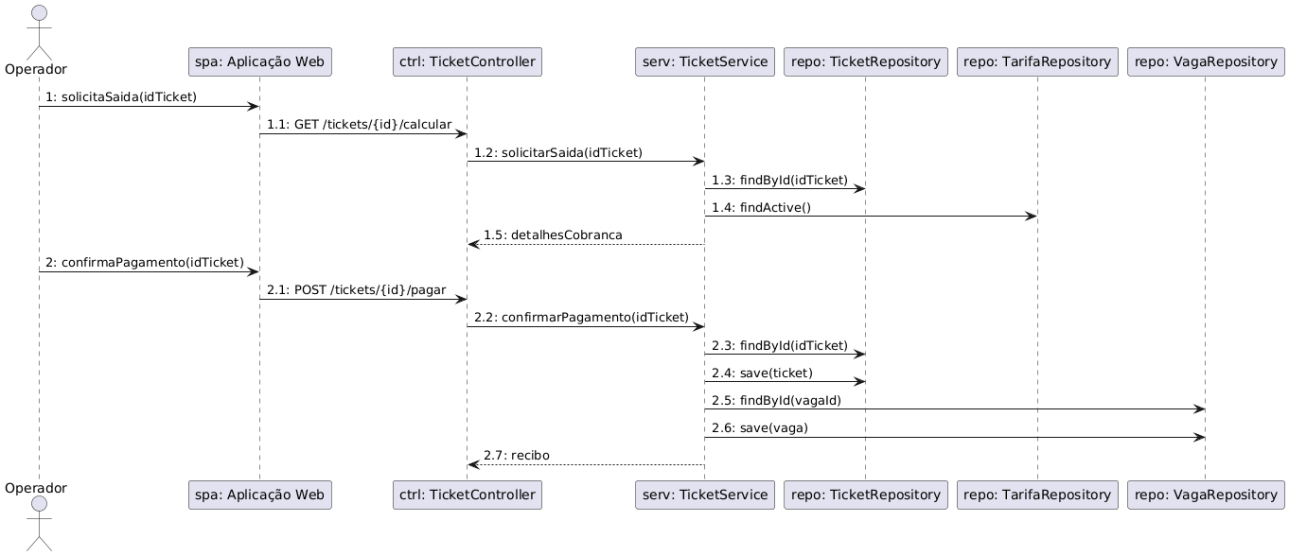

Usuario
id: Long
nome: String
login: String
senhaHash: String
login()
Operador
Administrador
gerenciarTarifa()
gerenciarOperador()
1
registra
0..*
Ticket
id: Long
horaEntrada: DateTime
horaSaida: DateTime
valorTotal: Double
estado: String
calcularValor(): Double
fecharTicket()
1
1
ocupa
1
1
usa para cálculo
Veiculo
placa: String
1
Vaga
id: Long
codigo: String
status: VagaStatus
ocupar()
liberar()
1
Tarifa
id: Long
valorPorHora: Double
dataVigencia: Date
1
1
VagaStatus
LIVRE
OCUPADA

3.4 Diagramas de Sequência



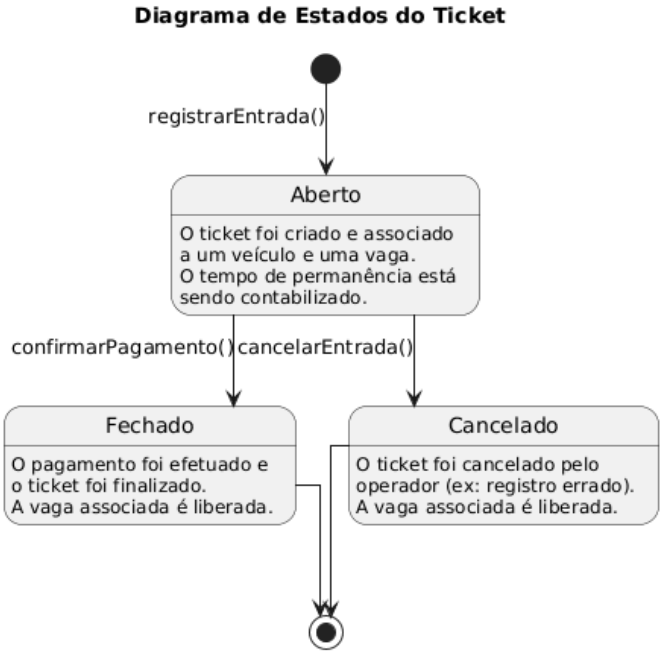
3.5 Diagramas de Comunicação

Este diagrama modela o ciclo de vida de um objeto Ticket dentro do sistema AutoPark. Ele mostra os estados pelos quais o ticket passa e as transições (eventos) que causam a mudança de um estado para outro.



3.6 Diagramas de Estados

Este diagrama modela o ciclo de vida de um objeto Ticket dentro do sistema AutoPark. Ele mostra os estados pelos quais o ticket passa e as transições (eventos) que causam a mudança de um estado para outro.



4. Modelos de Dados

Estratégia: A estratégia utilizada será a de **Mapeamento Objeto-Relacional (ORM)**.

Tecnologia: Conforme definido em nossos diagramas de arquitetura (Seção 3.1) e componentes (Seção 3.2), que especificam **Spring Boot** e **Spring Data JPA**, usaremos a **JPA (Java Persistence API)** como padrão de implementação.

Como funciona:

Cada classe do Diagrama de Classes (Seção 3.3) que precisa ser persistida será anotada com @Entity (Ex: class Ticket -> @Entity public class Ticket).

O nome da tabela no banco será mapeado usando @Table (Ex: class Ticket -> @Table(name = "tickets")).

Os atributos da classe (camelCase) serão mapeados para as colunas do banco (snake_case) usando @Column (Ex: horaEntrada -> @Column(name = "hora_entrada")).

Os relacionamentos entre classes serão mapeados usando anotações de associação da JPA (Ex: private Vaga vaga na classe Ticket será anotado com @ManyToOne ou @OneToOne, que gerenciará a chave estrangeira vaga_id).