Documento de Visão de Casos de Uso

Grupo 6

EasyStop MVP

Escrito por: Thâmara

1. Introdução

Este documento apresenta a visão de casos de uso para o MVP do Sistema EasyStop, um sistema de gestão de estacionamentos que automatiza os processos de check-in, check-out, controle de vagas e geração de relatórios.

2. Escopo do Produto

O EasyStop MVP focará nas funcionalidades essenciais para operação básica de um estacionamento:

- Registro de entrada e saída de veículos
- Cálculo automático de tarifas baseado no tempo
- Controle de vagas disponíveis e ocupadas
- Geração de relatórios básicos de gestão
- Sistema de alertas de capacidade

3. Casos de Uso Principais

UC-01: Realizar Check-in de Veículo

Interface Principal: ICheckinService

java

```
public interface ICheckinService {
    CheckinResponse realizarCheckin(CheckinRequest request);
}
```

Interfaces de Suporte:

IVagaRepository - Busca vagas disponíveis
IRegistroRepository - Persiste registros de entrada
IValidadorPlaca - Valida formato da placa
IValidadorHorario - Valida horário de entrada

Fluxo:

1. Atendente submete placa e horário via frontend
2. Frontend chama ICheckinService.realizarCheckin()
3. Serviço coordena validações e persistência através das interfaces especializadas
4. Sistema retorna vaga sugerida

UC-02: Realizar Check-out e Calcular Valor

Interface Principal: ICheckoutService

java

```
public interface ICheckoutService {
    CheckoutResponse realizarCheckout(String placa);
}
```

Interfaces de Suporte:

- ICalculadoraTarifa - Calcula valor baseado no tempo
- IRegistroRepository - Busca registro de entrada
- IVagaRepository - Libera vaga ocupada

- IRelogioSistema - Fornece horário atual para cálculo

UC-03: Registrar Pagamento

Interface Principal: IPagamentoService

```java
public interface IPagamentoService {
    void registrarPagamento(Long registroId, FormaPagamento
    formaPagamento);
}
```

Interfaces de Suporte:

- IRegistroRepository - Atualiza status do registro
- IProcessadorPagamento - Executa regras específicas por forma de pagamento

---

UC-04: Monitorar Ocupação do Estacionamento

Interface Principal: IMonitorVagasService

```java
public interface IMonitorVagasService {
   EstatisticasVagas obterEstatisticasVagas();
   boolean estaProximoLotacao();
}
```

Interfaces de Suporte:

- IVagaRepository - Contabiliza vagas ocupadas/livres
- IAlertaService - Dispara notificações quando necessário

UC-05: Gerar Relatórios

Interface Principal: IGeradorRelatorios

```java
public interface IGeradorRelatorios {
   RelatorioFaturamento gerarRelatorioFaturamento(LocalDate data);
   RelatorioOcupacao gerarRelatorioOcupacao(LocalDate data);
}
```

Interfaces de Suporte:

- IRegistroRepository - Fornece dados históricos
- ICalculadoraMetricas - Calcula médias e estatísticas

5. Benefícios desta Abordagem

Para Desenvolvedores:

- Testabilidade: Interfaces pequenas são fáceis de mockar em testes unitários
- Manutenibilidade: Mudanças em uma funcionalidade não afetam outras
- Clareza: Cada interface documenta explicitamente sua responsabilidade

Para a Equipe:

- Desenvolvimento Paralelo: Múltiplos desenvolvedores podem trabalhar em interfaces diferentes
- Integração Simplificada: Contratos bem definidos entre módulos
- Reuso: Interfaces especializadas podem ser compostas em novos serviços

6. Exemplo de Implementação

Backend (Java/Spring)

java

```
1.  @Service
2.  @RequiredArgsConstructor
3.  public class CheckinService implements ICheckinService {
4.
5.      private final IVagaRepository vagaRepository;
6.      private final IRegistroRepository registroRepository;
7.      private final IValidadorPlaca validadorPlaca;
8.      private final IValidadorHorario validadorHorario;
9.
10.     @Override
11.     public CheckinResponse realizarCheckin(CheckinRequest request) {
12.         validadorPlaca.validar(request.placa());
13.         validadorHorario.validar(request.horarioEntrada());
14.
15.         Vaga vaga = vagaRepository.findFirstLivre()
16.             .orElseThrow(() -> new EstacionamentoLotadoException());
17.
18.         Registro registro = new Registro(request.placa(), request.horarioEntrada(),
    vaga);
19.         registroRepository.save(registro);
20.
21.         vaga.ocupar();
22.         vagaRepository.save(vaga);
23.
24.         return new CheckinResponse(vaga.getNumero());
25.     }
}
26.
```

4. Principais Requisitos Não Funcionais

- Desempenho: Tempo de resposta máximo de 2 segundos para operações críticas
- Disponibilidade: 99,5% de disponibilidade mensal
- Usabilidade: Interface intuitiva que permita conclusão de check-in/check-out em até 30 segundos
- Segurança: Criptografia de dados sensíveis em trânsito e em repouso

5. Tecnologias

- Frontend: Vue.js
- Backend: Java/Groovy com Spring Boot
- Documentação: Swagger/OpenAPI
- Testes: TestRails para gestão de casos de teste
- Banco de dados: A definir (sugestão: PostgreSQL para produção)