Documentação do Projeto

**Back-End Documentation**

**1. Models (Entidades)**

**1.1. Proprietário**

@Entity  
public class Proprietario {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
   
 private String nome;  
 private String email;  
 private String telefone;  
 private String user\_password;  
   
 @OneToMany(mappedBy = "proprietario")  
 private List<Vaga> vagas;  
  
 // Getters, Setters e Construtores  
}  
Descrição: Representa um proprietário que anuncia vagas.  
Atributos:  
 id: Identificador único do proprietário.  
 nome: Nome do proprietário.  
 email: Email utilizado para login.  
 telefone: Número de contato.  
 user\_password: Senha de acesso.  
 vagas: Lista de vagas associadas ao proprietário.

**1.2. Vaga**

@Entity  
public class Vaga {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
   
 private String descricao;  
 private Double preco;  
 private String cep;  
 private String rua;  
 private String numero;  
 private String bairro;  
 private String cidade;  
 private String estado;  
 private String imagemUrls;  
  
 @ManyToOne  
 @JoinColumn(name = "proprietario\_id")  
 private Proprietario proprietario;  
  
 // Getters, Setters e Construtores  
}  
Descrição: Representa uma vaga de estacionamento cadastrada no sistema.  
Atributos:  
 id: Identificador único da vaga.  
 descricao: Descrição detalhada da vaga.  
 preco: Preço cobrado pela vaga (por hora, dia, ou mês).  
 cep, rua, numero, bairro, cidade, estado: Endereço completo da vaga.  
 imagemUrls: URL(s) das imagens associadas à vaga.  
 proprietario: Relacionamento com o proprietário da vaga.

**2. Controllers**

**2.1. ProprietarioController**

@RestController  
@RequestMapping("/proprietarios")  
public class ProprietarioController {  
  
 @Autowired  
 private ProprietarioService proprietarioService;  
  
 @GetMapping("/{id}")  
 public Proprietario getProprietarioById(@PathVariable Long id) {  
 return proprietarioService.findById(id);  
 }  
  
 @PostMapping  
 public Proprietario createProprietario(@RequestBody Proprietario proprietario) {  
 return proprietarioService.save(proprietario);  
 }  
}  
Rota: /proprietarios  
Métodos:  
 GET /{id}: Retorna os dados de um proprietário pelo ID.  
 POST /: Cria um novo proprietário.

**2.2. VagaController**

@RestController  
@RequestMapping("/vagas")  
public class VagaController {  
  
 @Autowired  
 private VagaService vagaService;  
  
 @GetMapping("/{id}")  
 public Vaga getVagaById(@PathVariable Long id) {  
 return vagaService.findById(id);  
 }  
  
 @GetMapping("/proprietario/{proprietarioId}")  
 public List<Vaga> getVagasByProprietario(@PathVariable Long proprietarioId) {  
 return vagaService.findByProprietario(proprietarioId);  
 }  
  
 @PostMapping  
 public Vaga createVaga(@RequestBody Vaga vaga) {  
 return vagaService.save(vaga);  
 }  
}  
Rota: /vagas  
Métodos:  
 GET /{id}: Retorna uma vaga específica pelo ID.  
 GET /proprietario/{proprietarioId}: Retorna todas as vagas de um proprietário pelo ID do proprietário.  
 POST /: Cria uma nova vaga.

**3. Services**

Os serviços para as entidades Proprietario e Vaga são responsáveis pela lógica de negócio. Eles interagem com os repositórios e são injetados nos controllers.

**Front-End Documentation**

**1. HomeScreen**

Descrição: A tela inicial que exibe um mapa com as vagas disponíveis.  
Componentes:  
 Mapa: Um mapa interativo que mostra as vagas cadastradas em um raio de proximidade.  
 Barra de Pesquisa: Permite ao usuário buscar localizações específicas.  
 Botão de Voltar à Localização Atual: Traz o mapa de volta para a posição atual do usuário.  
  
Principais Funções:  
 \_geocodeAddress(): Converte endereços em coordenadas.  
 updateVagasOnMap(): Atualiza as vagas exibidas no mapa com base na localização do usuário.

**2. LoginScreen**

Descrição: Tela de login para usuários e proprietários.  
Componentes:  
 Campos de Input: E-mail e senha.  
 Botão de Login: Valida e envia os dados para autenticação.  
 Link para Registro: Leva à página de registro.  
  
Principais Funções:  
 \_login(): Realiza a autenticação do usuário chamando a API.

**3. MinhasVagasPage**

Descrição: Exibe as vagas cadastradas pelo proprietário logado.  
Componentes:  
 Lista de Vagas: Exibe as vagas cadastradas em forma de cards clicáveis.  
 Botão 'Anunciar Nova Vaga': Direciona para um formulário de cadastro de vagas.  
  
Principais Funções:  
 \_fetchMinhasVagas(): Requisita as vagas do proprietário logado via API.  
 \_criarVaga(): Cria uma nova vaga ao enviar os dados para a API.

**4. VagaDetailScreen**

Descrição: Mostra os detalhes completos de uma vaga específica.  
Componentes:  
 Carrossel de Imagens: Exibe as imagens da vaga.  
 Detalhes: Mostra informações como descrição, preço, endereço e proprietário.  
  
Principais Funções:  
 \_fetchVagaDetail(): Requisita os detalhes da vaga ao backend.  
 \_displayImagesCarousel(): Exibe as imagens no formato de carrossel.

**5. ProprietarioPage**

Descrição: Exibe os dados do proprietário e suas vagas.  
Componentes:  
 Informações do Proprietário: Nome, email e telefone.  
 Lista de Vagas: Vagas cadastradas pelo proprietário, exibidas em um carrossel lateral.  
  
Principais Funções:  
 \_fetchProprietarioInfo(): Busca as informações do proprietário via API.

**Como Rodar o Projeto**

**Backend:**

1. Requisitos: Java 11+, Maven, Banco de Dados H2 ou DB2.  
2. Passos:  
 - Clone o repositório.  
 - Abra o projeto na IDE  
 - Rode a classe parkingSearchApplication em:

Parking\_app-main\parking\_search\src\main\java\br\com\challenge\parkingSearch

- Tornar o Backend visível para o Frontend usando o LocalTunnel, Instale o Localtunnel globalmente (exige o NodeJS).

- Abra o CMD, digite "lt --port 8080". Após o link ser gerado, abra-o no navegador, abaixo conseguirá obserar outro link em azul, basta clicar nele e copiar a senha que vai estar na página, volte a página anterior e cole no campo password. e clique no botão "Click to submit". Após esses passos o nosso backend estará visível.

Agora basta copiar o link da pagina que apareceu o acesso negado.

**Frontend:**

1. Requisitos: Flutter SDK.  
2. Passos:  
 - Clone o repositório.  
 - Instale as dependências com flutter pub get.  
 - Com o link copiado entraremos no arquivo lib/components/ApiUrl e trocaremos o link da variável baseUrl, (excluindo a barra que virá ao final). Salve as alterações.

- Agora basta abrir a classe main do projeto e selecionar o dispositivo que será usado para rodar a aplicação, funciona tanto para o Chrome quanto para o Emulador.

- Rode a aplicação e usufrua da nossa solução!