**Sistema de Controle de Estacionamento**

Este documento detalha as etapas para o desenvolvimento de um software para controle de estacionamento: Análise, Projeto, Instalação, Implementação, Testes e Deploy. Documento criado por Jair Alarcón, qualquer dúvida: [jairabu@gmail.com](mailto:jairabu@gmail.com)

**1. Análise**

Na análise deve-se fazer o levantamento de requisitos, com o objetivo de apresentar as principais necessidades do cliente.

**1.1 Requisitos**

Implementar uma solução para controle de estacionamento, os requisitos necessários para a solução são:

• Autenticação via banco de dados

• Cadastro de Clientes (Nome, CPF, Telefone)

• Cadastro de Veículos (Placa, Modelo, Cor)

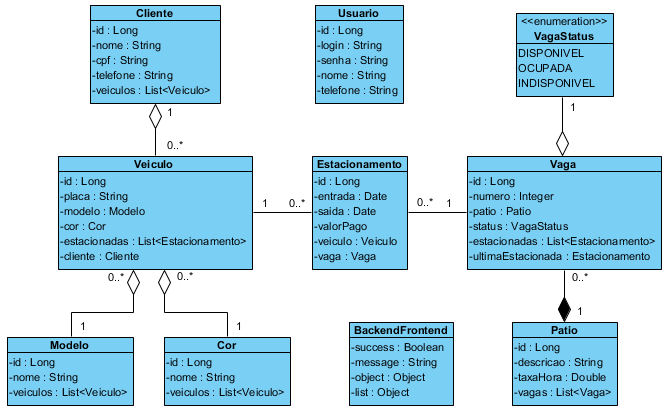
• Cadastro de Pátio (Descrição, Número de Vagas, Taxa/hora)

• Estacionamento: Registrar entrada e saída de veículos tempo de permanência e valor a ser pago.

• Exibir um dash em tela com o total de vagas ocupadas e total livre.

**1.2 Diagrama de Classes**

O Diagrama de Classes abaixo foi feito com base nos requisitos.



**1.3 Observações**

Não foi criada uma classe telefone pois pressumiu-se que o cliente possui apenas um único telefone.

Todos os relacionamentos foram feitos de forma bidirecional, por exemplo, Veiculo possui uma variável cor, e Cor possui uma variável List<Veiculo>.

Na classe Estacionamento o tempo de permanência não foi salvo no Banco de Dados pois é uma variável derivada que pode ser calculada facilmente através da entrada e saída do veículo. No entanto, foi criada a variável valorPago pois a taxaHora pode mudar e não foi decidido armazenar o histórico de mudanças das taxas.

A escolha de agregação foi feita quando a remoção de um registro na Classe container não implica na remoção da classe contida, por exemplo: a remoção de Veiculo não implica em remoção de Cor, portanto é uma agregação e não uma composição.

Pressumiu-se que o veículo somente pertence a um único cliente, caso o veículo seja de mais de um cliente ele será armazenado de forma repetida no Banco de Dados.

**2. Projeto**

No projeto deve-se escolher quais ferramentas, frameworks e linguagens devem ser utilizadas para criar uma solução baseada no problema identificado na etapa de análise. As tecnologias (open-source) escolhidas foram:

• Java 8: para desenvolver o Backend (servidor);

• Spring Boot: para integrar os Frameworks e Ferramentas do Java;

• Hibernate: Framework para salvar os objetos no Banco de Dados;

• Angular 2+: para desenvolver o Frontend (telas);

• Bootstrap com CSS Meta Queries: controlar responsividade (versão mobile das telas);

• PostgreSQL: Sistema Gerenciador de Banco de Dados;

• Lombok: para otimizar o código Java.

Algumas decisões de arquitetura:

• Uso de GenericDAO.java para persistência de objetos no Banco de Dados,

• Uso de GenericController.java para a criação de controladores MVC.

• Uso de Maven: para baixar todos os frameworks utilizados no Java.

• Não foi criada uma camada de serviço e repositório, para simplificar a arquitetura e não haver necessidade a priori.

• Uso de generic.component.ts no Frontned para efetuar o CRUD básico das telas.

• Uso de generic.service.ts no Frontend para efetuar as requisições HTTP com JSON do CRUD básico.

**3. Instalação**

Os links para instalação do software são apresentador a seguir.

3.1 Instalar JDK do Java (Versão 8):

<http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html>

3.2 Instalar Eclipse IDE for Java EE Developers (Versão Photon):

<http://www.eclipse.org/downloads/eclipse-packages/>

3.3 Instalar PostgreSQL (Versão 9.6 ou superior):

<https://www.postgresql.org/download/windows/>

3.4 Instalar Atom:

<https://atom.io>

3.5 Instalar Node.js e NPM (LTS):

<https://nodejs.org/en/download>

3.6 Instalar Angular CLI globalmente:

npm install -g npm

npm install -g @angular/cli@latest

3.7 Criar projeto

cd \

ng new estacionamento-frontend

cd estacionamento-frontend

3.8 Instalar módulos dentro da pasta angular-crud do projeto:

npm install --save @angular/material @angular/cdk

npm install --save angular/material2-builds angular/cdk-builds

npm install --save @angular/animations

npm install --save hammerjs

npm install --save bootstrap

npm install --save @ng-bootstrap/ng-bootstrap

npm install --save ng2-currency-mask

npm install --save web-animations-js

npm audit fix –force

3.9 Adicionar as 3 linhas abaixo dentro da tag “styles” que fica dentro do arquivo angular.json (removendo a linha “styles.css”:

"styles": [

"src/styles.scss",

"./node\_modules/bootstrap/dist/css/bootstrap.css",

"./src/app/util/tema.scss"

],

Obs.: Em caso de erro remover a pasta node\_modules e digitar:

npm install

npm link

Obs.: Para executar o Angular:

ng serve --open

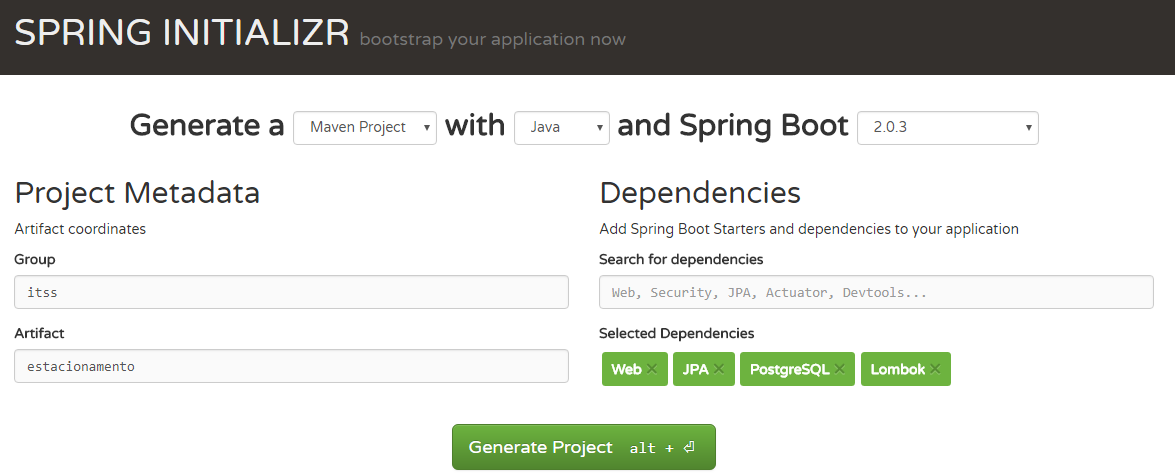
3.10 Criar banco de dados **estacionamento** no PostgreSQL

3.11 Instalar lombok por fora do Eclipse

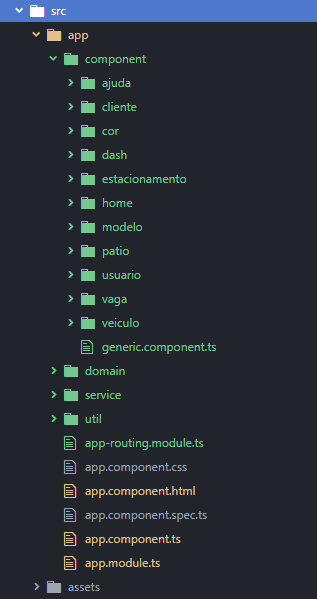
Dar update maven

Dar clean no projeto

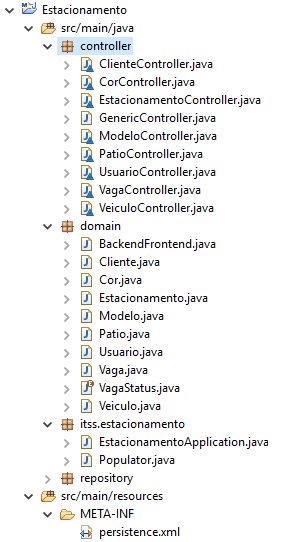
3.12 Foi utilizado o Spring Initializr (<https://start.spring.io>) para a criação inicial do projeto Maven:



3.13 A estrutura de arquivos do Frontend pode ser vista na próxima figura:

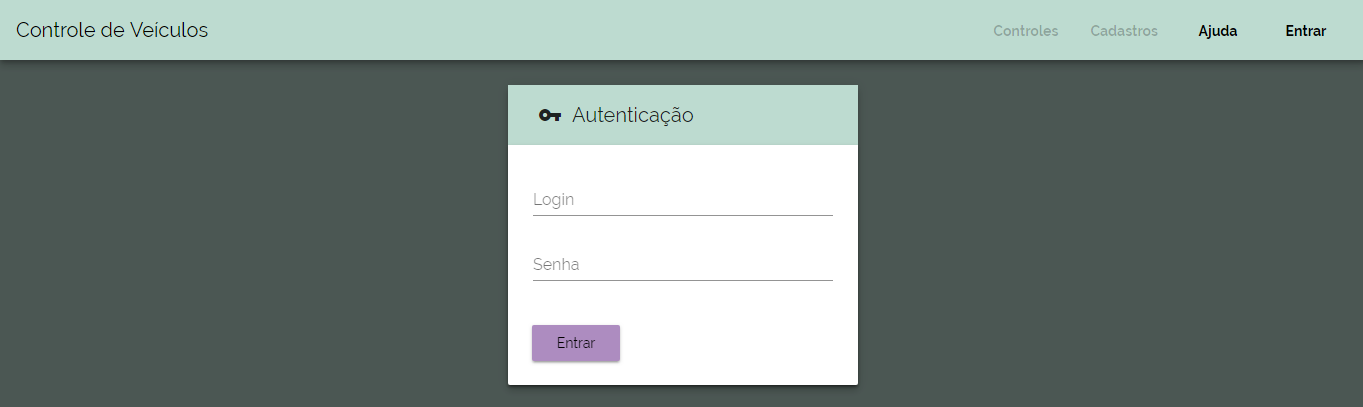


3.14 A estrutura de arquivos do Backend pode ser vista na próxima figura:

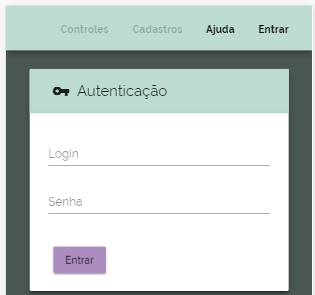


**4. Testes de Telas**

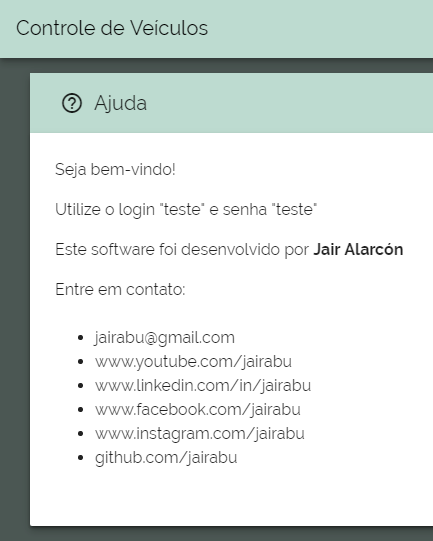
4.1 Tela Inicial de Login



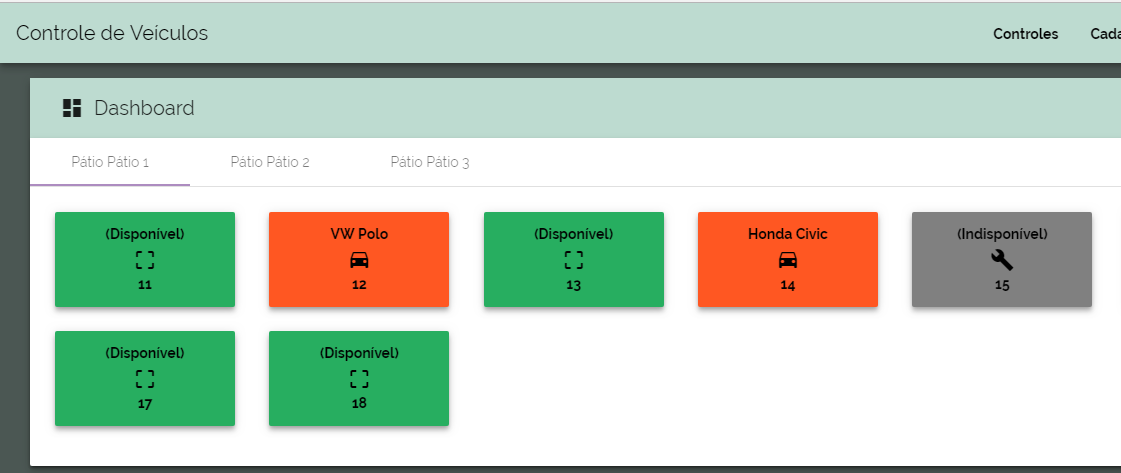
4.2 Tela Inicial de Login (versão mobile)



4.3 Tela de Ajuda



4.4 Tela de Dashboard



Obs. 1: Cada estado da vaga possui uma cor diferente.

Obs. 2: Cada alteração feita nos estacionamentos atualiza os dados do Dash.

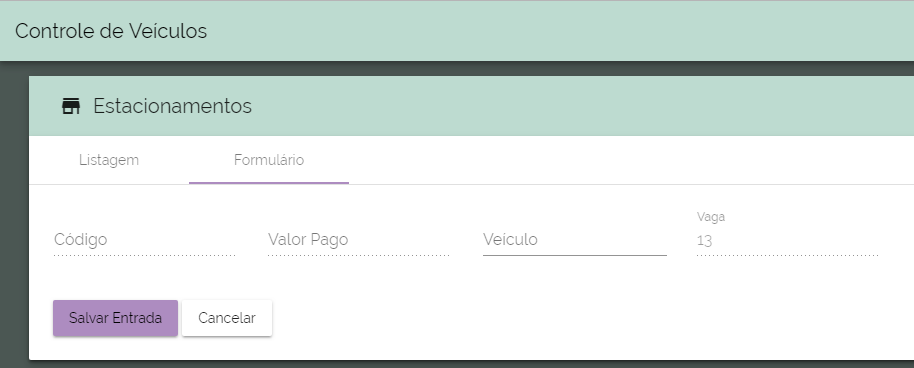
Obs. 3: As abas com os pátios é gerada dinâmicamente

Obs. 4: As vagas dos pátios são geradas dinâmicamente

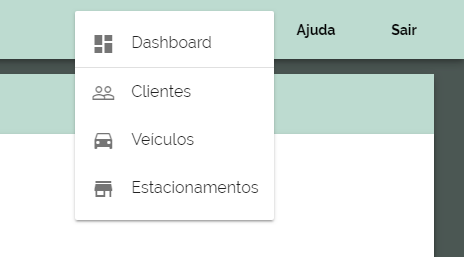
4.5 Ao clicar em uma vaga ocupada será aberta a tela de Estacionamentos



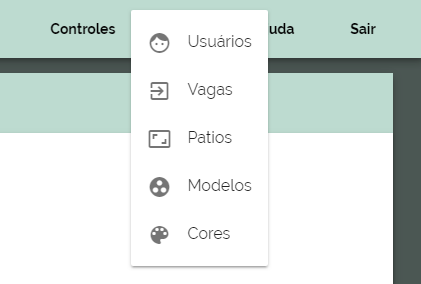
4.6 Ao clicar em uma vaga livre será aberta a tela de Estacionamentos



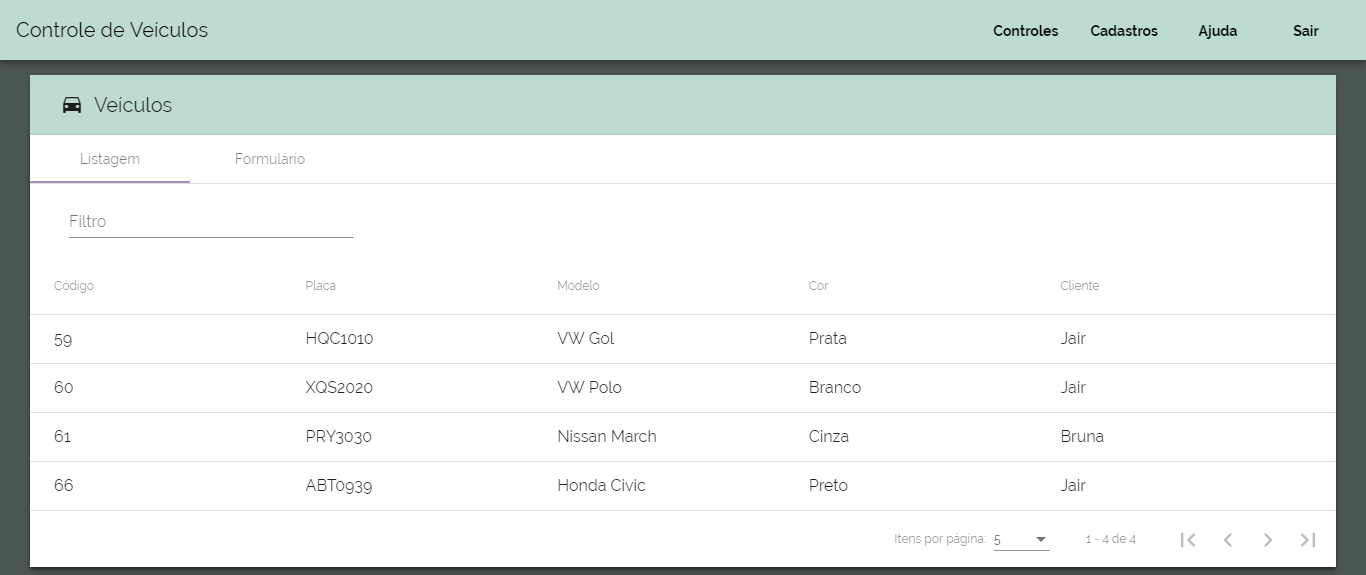
4.7 Menú de Controles



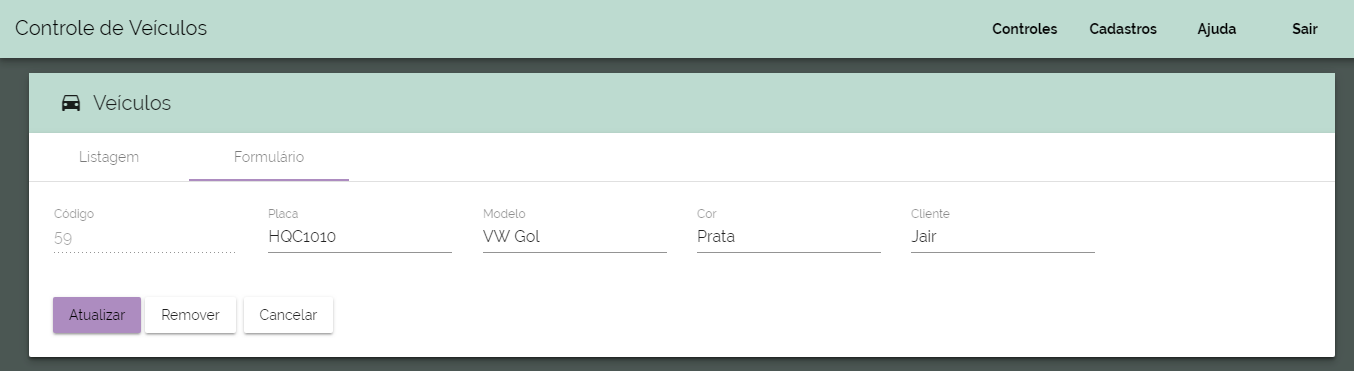
4.8 Menú de Cadastros



4.9 Tela de Veículos mostrando a aba Listagem



4.10 Tela de Veículos mostrando a aba Formulário

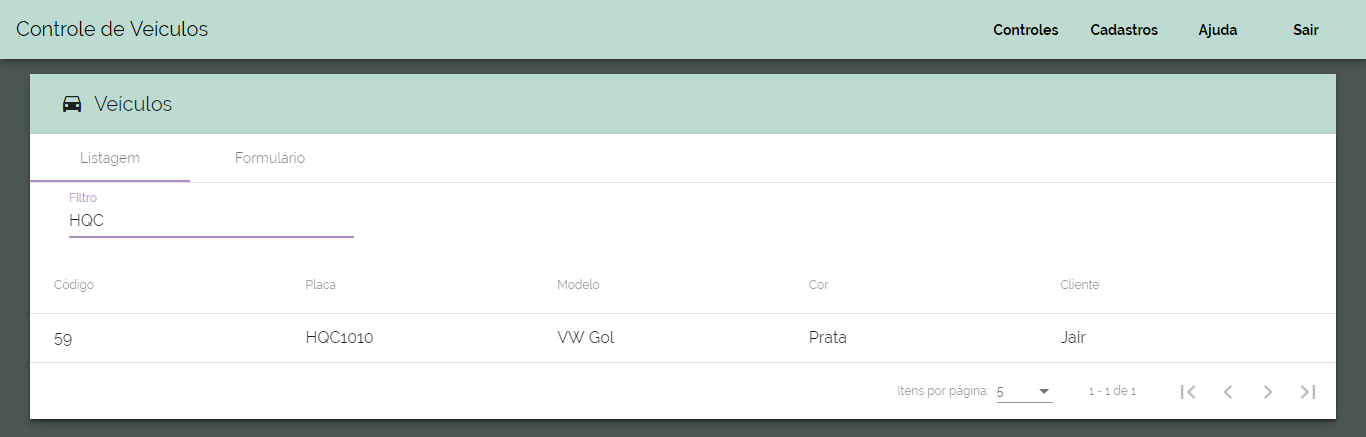


Obs.: Todas as telas de CRUD estão implementadas, não serão mostradas apenas para resumir o documento.

4.11 Tela de Veículos mostrando componente combo ou dropdown



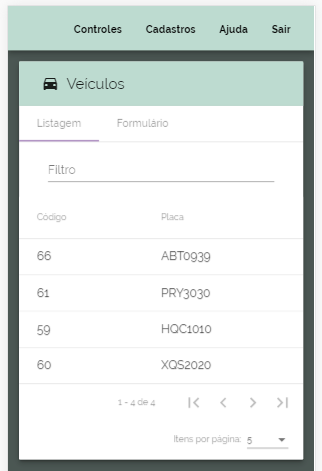
4.12 Tela de Veículos mostrando o uso de Filtro na aba de Listagem



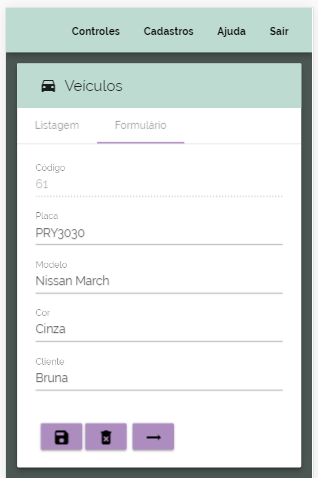
4.13 Tela de Veículos mostrando o uso de ordenação por Modelo



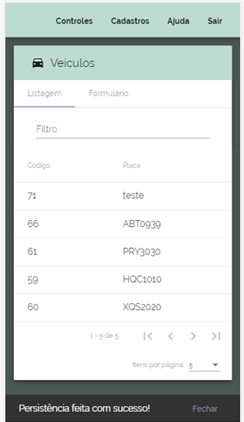
4.14 Tela de Veículos mostrando Listagem versão mobile com 2 colunas apenas



4.15 Tela de Formulário versão mobile com botões utilizando ícones



4.16 Tela de Veículo mostrando Snack (mensagem embaixo da tela)



**5. Deploy**

5.1 Gerar script do banco de dados (dentro da pasta bin do PostgreSQL):

pg\_dump -U postgres --schema-only --no-owner estacionamento > tabelas.sql

Obs.: Comentar as seguintes 3 linhas:

SET idle\_in\_transaction\_session\_timeout = 0;

CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg\_catalog;

COMMENT ON EXTENSION plpgsql IS 'PL/pgSQL procedural language';

5.2 Alterar variável **urlService** em C:\angular-crud\src\app\util\globals.ts

static urlService:string = "http://estacionamento.web7695.kinghost.net";

5.3 Efetuar Build na pasta do frontend:

**ng build -prod --base-href http://estacionamento.com.br**

5.4 Copiar pasta gerada em dist na pasta do backend (Java) em /src/main/resources/webapp

5.5 Alterar **persistence.xml** em src/main/resources/META-INF com URL do banco remoto

5.6 Gerar o arquivo estacionamento-0.0.1-SNAPSHOT.war, digitando:

**mvn clean package**

5.7 Enviar WAR para Servidor:

Renomear estacionamento-0.0.1-SNAPSHOT.war para ROOT.war

Enviar ROOT.war para /home/estacionamento/webapps via FileZilla

5.8 Executar Scripts dentro do PgAdmin:

tabelas.sql

registros.sql

5.9 Importação de Banco de Dados do Servidor:

Após criar banco de dados estacionamento:

create user estacionamento;

alter user estacionamento with superuser;

**psql -U postgres estacionamento.com.br < estacionamento.sql**