# Regras de Negocio

Contruir uma API para controle de um estacionamento

Dados cadastrados em BD Relacional

Exposta na nuvem

Controle de acesso

# RNF

Guardar placa cor e modelo

Guardar data e hora de entrada

Calcular valor para saída

Um tipo de usuário, o operador do estacionamento

# Introdução

## Spring Boot

Framework para facilitar processo de configuração e publicação dos Serviços Web

Rodar a aplicação mais rápido possível

## Spring Framework

Criado para evitar a complexidade do Java EE, é um complemento do Java EE para facilitar nas requisições

## Spring Dependencias

* Spring Data – Vincular ao Banco de dados
* Spring Cloud – Armazenar na nuvem
* Spring Security – Seguranca da aplicação
* Spring Web – API na Web
* Lombok – ajuda nos get e sets
* DevTools – Não é necessário rodar toda vez, quando salva a aplicação atualiza solo

E para atualizar solo 🡪 Build – Recompile...;

* H2 – banco de dados na nuvem

## Implantando Codigo no Heroku

Heroku é uma aplicação que pega o código e compila, após compilado, disponibiliza um link web para poder ver a aplicação

Por default ele utiliza o Java 8, para mudar do padrão, implantar o código no projeto, dizendo que vc quer colocar para a versão, no meu caso “Java 17”;

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente Dentro desse File...:

java.runtime.version=17

Quando da erro no Heroku por não achar a linguagem

-----> Building on the Heroku-22 stack

-----> Determining which buildpack to use for this app

! No default language could be detected for this app.

HINT: This occurs when Heroku cannot detect the buildpack to use for this application automatically.

See https://devcenter.heroku.com/articles/buildpacks

! Push failed

Provavelmente por o arquivo não estar logo de cara, como nesse meu:

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Ele tem varias pastas p aí chegar no SRC ...

Deve especificar o caminho para o sistema...

**... Não sei ainda como resolver :/**

## Rest

Rest são os padrões de requisições de HTTP – modelo de arquitetura para sistemas distribuídos

Há os níveis de Rest, no total 4, o que torna quando os 4, REST FULL (Já falado em arquivos anteriores)

### Verbos HTTP

#### Get

Anotado com **@GetMapping** (No controller como BOAS PRATICAS)

Faz a requisição de PEGAR os dados do Banco, *GetAll* para todos os dados, ou com *@GetMapping (“/id”)* para um ID em especifico

#### Post

Anotado com **@PostMapping** (Pacote Controller[...])

Postagem dos dados no banco, sempre em formato JSON*(mas isso nao vem ao caso agora)*

#### Put

Anotado com **@PutMapping(“/id”)** (Pacote Controller[...])

Atualização dos dados, sempre especificando na URL o objeto que quer atualizar/alterar

#### Delete

Anotado com **@DeleteMapping(“/id”)** (Pacote Controller[...])

Deletar os dados do banco, sempre especificando na URL o objeto que quer deletar

#### Path

-\_-\_-\_- Não usei ainda

Quando quer alterar algum campo em especifico, informar **Campo e Valor**

# Declarando Controller

Declarado com @RestController & @RequestMapping(“/***urlDeReferencia***”)

Foi declarado a Entidade Própria do nosso Banco de Dados, **isso não é uma boa pratica**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Por isso deve ser feito o DTO

***Nunca se expõe o objeto de domínio na aplicação***

Para isso, criar camada de Serviço, para definir o domínio

# Criando Service

Declarado com @Service, injetando dependência Service vinculado ao controller

Quando implementa Service 🡪

## Inversão De Controle (IOC) - Service Em Controller

EM VEZ DE @AUTOWIRED ParkingService parkingService

COLOCAR private final ParkingService parkingService

## Deixando Codigo mais Limpo

Agora, em vez de feito como acima, declarando tudo aquilo em Controller

Texto

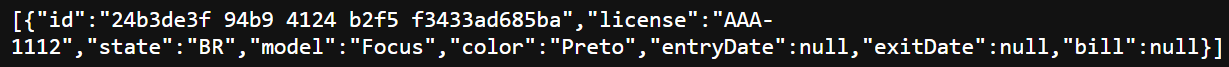
Descrição gerada automaticamenteApagar o que feito em Controller

Declarar dessa forma em Service:

Tela de computador com texto preto sobre fundo branco

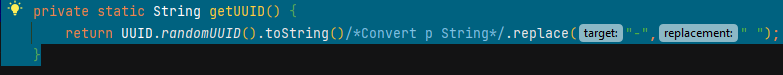
Descrição gerada automaticamenteComo pode ver

Toda aquela poluição visual no controller, foi para o Service, onde ficam as regras de negócio



/\*LEMBRANDO Arrays.asList -> adiciona manualmente os itens dentro do construtor, por isso Contrutor Vazio\*/

**UUID** atualmente não está mais sendo feita na ***Entidade***, mas sim no ***Service*** como uma Regra de Negócio, **PEGANDO UUID 🡪 CLASSE DO JAVA E CONVERTENDO PARA STRING**



## Transferindo o que declarado em (Entidade)Parking para (Data Transfer 🡪 DTO)ParkingDTO

**BOAS PRATICAS ISSO!!!**

Se fosse fazer essa transferência de Entidade Para DTO manual ficaria um código muito Extenso...

ENVOLVENDO **FOR EACH**,

**Convertendo dentro de Controller os GetENTIDADE para SetDTO**

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

Entao utiliza-se da biblioteca Mapper

### ModelMapper

*Essa biblioteca deve-se adicionar no POM.XML digitando* ***modelmapper, já aparece – org.modelmapper***

public class ParkingMapper { //responsável por fazer conversão em vez de converter lá dentro de cada classe

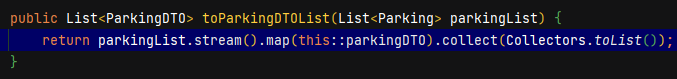
então o que deve fazer para converter, primeiro mostrando **Classe Controller**, e depois da classe criada **MODEL\_MAPPER**

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente



Voltando para Controller

Texto

Descrição gerada automaticamente