Sumário

[Regras de Negocio 2](#_Toc121169909)

[RNF 2](#_Toc121169910)

[Introdução 3](#_Toc121169911)

[Spring Boot 3](#_Toc121169912)

[Spring Framework 3](#_Toc121169913)

[Spring Dependencias 3](#_Toc121169914)

[Implantando Codigo no Heroku 3](#_Toc121169915)

[Rest 4](#_Toc121169916)

[Verbos HTTP 4](#_Toc121169917)

[Get 4](#_Toc121169918)

[Post 4](#_Toc121169919)

[Put 4](#_Toc121169920)

[Delete 4](#_Toc121169921)

[Path 4](#_Toc121169922)

[Declarando Controller 5](#_Toc121169923)

[Criando Service 5](#_Toc121169924)

[Inversão De Controle (IOC) - Service Em Controller 5](#_Toc121169925)

[Deixando Codigo mais Limpo 5](#_Toc121169926)

[Transferindo o que declarado em (Entidade)Parking para (Data Transfer 🡪 DTO)ParkingDTO 6](#_Toc121169927)

[ModelMapper 7](#_Toc121169928)

[Como fazer para não aparecer os campos Null / Nulos 7](#_Toc121169929)

[Implementando Get By Id através do ModelMapper 8](#_Toc121169930)

[Em controller 8](#_Toc121169931)

[Em Service 8](#_Toc121169932)

[Modelando Post 8](#_Toc121169933)

[No Service 9](#_Toc121169934)

[No Controller 9](#_Toc121169935)

[No Controller Com Classe Requisições (ATUALIZADO) 9](#_Toc121169936)

[No ModelMapper Com Classe Requisições (ATUALIZADO) 9](#_Toc121169937)

[Sucesso na postagem 10](#_Toc121169938)

[Modelando Delete 10](#_Toc121169939)

[No Controller 10](#_Toc121169940)

[No Service (NOTA: Classe para remover arquivos é o Remove, e não delete, então...) 11](#_Toc121169941)

[Swagger 11](#_Toc121169942)

[Classe Config para Swagger 11](#_Toc121169943)

[Configurando Swagger nos pacotes 12](#_Toc121169944)

[@ApiIgnore 🡪 Quando quero ignorar aquela classe 12](#_Toc121169945)

[@Api 🡪 Quando quero que aquela classe apareca 12](#_Toc121169946)

[@ApiOperation 🡪 Quando quero renomear alguma requisição 12](#_Toc121169947)

[Mas e se Caso ocorrer Algum Erro... Como tratar? 12](#_Toc121169948)

[Erro de NotFound(404) apresentando como Internal Server Error(500) Como resolver? 12](#_Toc121169949)

[ParkingNotFoundException 13](#_Toc121169950)

[Aparecendo um erro muito extenso, como deixar melhor a exibição para Usuário ou FrontEnd (tratar erro nos Recursos) 13](#_Toc121169951)

[Preparando Persistência no Banco (JPA, HIBERNATE) 13](#_Toc121169952)

[Criando a Annotation @Entity 14](#_Toc121169953)

[Configurações de Persistência no Banco em “Resources” 14](#_Toc121169954)

[Deu Erroooooo, mas já resolvido 14](#_Toc121169955)

[Após implementação Banco, Preparando camadas para requisições no banco! 15](#_Toc121169956)

[Repositório finalmente começa a funcionar 15](#_Toc121169957)

[Implementar o Service com JPA 15](#_Toc121169958)

[@Transactional 16](#_Toc121169959)

[Aplicando Spring Security (Salvando em memoria) 17](#_Toc121169960)

[@EnableWebSecurity 17](#_Toc121169961)

# 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

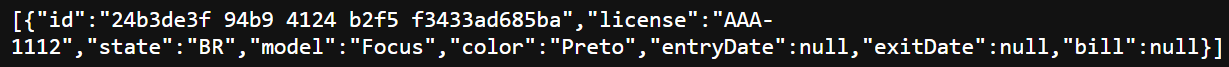
Descrição gerada automaticamenteApagar o que feito em Controller

Declarar dessa forma em Service:

Tela de computador com texto preto sobre fundo branco

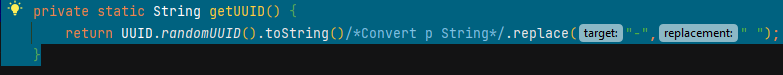
Descrição gerada automaticamenteComo pode ver

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

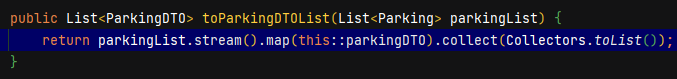
então o que deve fazer para converter, primeiro mostrando **Classe Controller**, e depois da classe criada **MODEL\_MAPPER**

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente



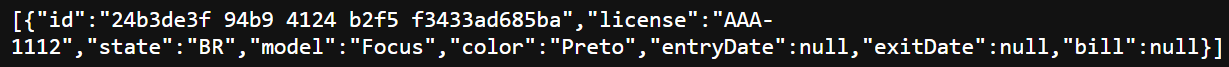
Voltando para Controller

Texto

Descrição gerada automaticamente

Dessa forma os códigos ficam bem mais empacotados, é necessário ter um conhecimento pois é uma complexidade maior de código, ter a determinação para ler documentações(uma coisa que eu nn tenho, mas vamo indo neh)

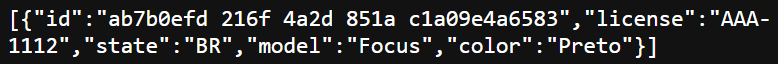
# Como fazer para não aparecer os campos Null / Nulos



No DTO

Colocar a Annotation @Json

@JsonInclude(JsonInclude.Include.*NON\_NULL*)



Dessa forma ficando bem mais apresentável, em os campos null

# Implementando Get By Id através do ModelMapper

Quando implementado buscando por ID, ele não retorna mais uma lista, então o método no ModelMapper é o mesmo, no entanto o modo de busca é diferente

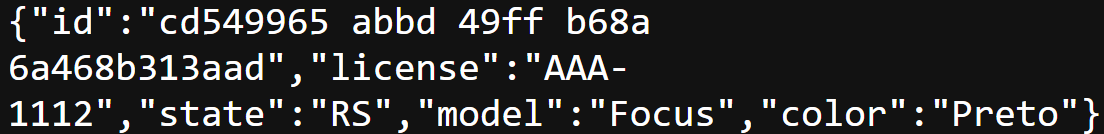
## Em controller

As chamadas quase as mesmas:Texto

Descrição gerada automaticamente

## Em Service

public Parking findById(String id) {  
 return *parkingMap*.get(id)**;**}

Busca por ID [localhost:8080/parking/cd549965 abbd 49ff b68a 6a468b313aad](http://localhost:8080/parking/cd549965%20abbd%2049ff%20b68a%206a468b313aad)

# Modelando Post

Responde ao DTO

Por receber informações, não tem mais o PathVariable, e sim um ParkingDTO, pois recebe um corpo de informações

Para receber essas informações em DTO e passar para a Entidade, declarar assim no Mapper:

public Parking convertForParking(ParkingDTO parkingDTO) {  
 return *MODEL\_MAPPER*.map(parkingDTO**,** Parking.class)**;**}

## No Service

Texto

Descrição gerada automaticamente

## No Controller

Texto

Descrição gerada automaticamente

Agora, para fazer com que os parâmetros de envio sejam padrão , deve criar uma classe que so recebe esses parâmetros 🡺 Converte ele em DTO 🡺 Em Seguida em Service

*MUITO CONFUSO SLK*

## No Controller Com Classe Requisições (ATUALIZADO)

Texto

Descrição gerada automaticamente com confiança média

No Service Com Classe Requisições

Ele permanece o mesmo, sem alterações

## No ModelMapper Com Classe Requisições (ATUALIZADO)

public Parking parkingCreateDTO(ParkingRequisicoesDTO parkingRequisicoesDTO) {  
 return *MODEL\_MAPPER*.map(parkingRequisicoesDTO**,** Parking.class)**;**}

## Sucesso na postagem

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

# Modelando Delete

Depois de feito as outras requisições,

Com DELETE não tem segredo

Só fazer como fez para GET ONE / FOR ID:

## No Controller

Texto

Descrição gerada automaticamente

Corrigindo na aula

Texto

Descrição gerada automaticamente

## No Service (NOTA: Classe para remover arquivos é o Remove, e não delete, então...)

public Parking deleteId(String id) {  
 return *parkingMap*.remove(id)**;**}

Corrigindo na aula

public void deleteId(String id){  
 Parking parkingDelete = findById(id)**;** *parkingMap*.remove(id)**;**}

Modelando Put – Atualização

O Metodo é praticamente o mesmo que Post, mas com algumas alterações

Atualizar o Id existente, senão 🡪 Tratamento de exceções (mais abaixo)

# Swagger

Depois de tudo Ok as requisições primarias

Adicionar o **SWAGGER** através do **Spring Fox**

* Ir no POM.XML
  + Digitar <dependência – quando digitado, ele se auto-completa
  + Digitar springfox-swagger2 - quando digitado, ele se auto-completa
  + Selecionar o relacionado ao SpringFox – quando aparecer a opção de groupID
* Fazendo isso e atualizando o POM, RODE O PROGRAMA
  + Procure o localhost:8080/Swagger.ui.html
* No entanto, quando se faz sem os códigos JSON, ele não tem como configurar, dessa forma...

## Classe Config para Swagger

[...]Dessa forma, quando iniciar ele em vez de dar erro, já começa a aparecer os métodos implantados:

@Component  
@EnableSwagger2  
public class SwaggerConfig {  
}

Quando implementado, pega todas as classes de requisições

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamenteMas so quero que apareca as Requisições de Parking

Para isso...

Texto

Descrição gerada automaticamente

## Configurando Swagger nos pacotes

### @ApiIgnore 🡪 Quando quero ignorar aquela classe

Lembrando que se quiser ignorar outra classe que nn tenha nada haver com o projeto, assim como as requisições padrões do SpringFox Swagger

Colocar @Api Ignore

### @Api 🡪 Quando quero que aquela classe apareca

São o pacote onde as requisições do controller estão,

É possível renomear elas com @Api(“API CONTROLLER PARKING”)

### @ApiOperation 🡪 Quando quero renomear alguma requisição

As requisições, como GetOne GetAll GetNome... ou as demais operações

Para mapea-las com mais facilidade, declarar em cima do método com

@ApiOperation(“Get All IDs”) 🡪 Aparecera essa mensagem nessa requisição

# Mas e se Caso ocorrer Algum Erro... Como tratar?

Há formas de tratamento de Excessoes

Todas através da SUPER CLASSE EXCEPTIONS, que extende para RUNTIMEEXCEPTIONS ou Demais

## Erro de NotFound(404) apresentando como Internal Server Error(500) Como resolver?

Nesse projeto criaremos uma classe que Extende de RunTimeExceptions

Quando dá um erro de solicitação incorreta

Como no exemplo, o ID. Trataremos criando uma classe nova, dela estendendo RunTimeExceptions, com a Annotation para retornar um status 404 de não encontrada...:Texto

Descrição gerada automaticamente

### ParkingNotFoundException

Fazendo isso, buscando a classe ParkingNotFoundException

Texto

Descrição gerada automaticamente

Fazendo essa Annotation

Ele capta quando ocorrer o erro, e visualiza o que é para dar retorno em vez de **500**

## Aparecendo um erro muito extenso, como deixar melhor a exibição para Usuário ou FrontEnd (tratar erro nos Recursos)

Quando aparece um erro enorme, deixa muita poluição visual, acabando deixando o usuário se perder no próprio erro

Para tratar isso, deve ir no pacote recursos e deixar dessa forma as configurações

server.error.include-exception=false  
server.error.include-stracktrace=never  
server.error.include-message=always

Deixa a aplicação e aparição dos erros mais simples e prático entendimento

# Preparando Persistência no Banco (JPA, HIBERNATE)

Colocar Dependencia do Data JPA com Driver Postgress

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
</dependency>

## Criando a Annotation @Entity

Quando cria uma entidade, o primeiro passo é criar a anotação mostrando que se trata de uma entidade

E com isso, ele pede um ID que sera a chave primaria da classe (COISA DE BANCO DE DADOS)

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

## Configurações de Persistência no Banco em “Resources”

Mesmo fazendo isso ele apresenta erro quando “Run” no projeto

Para isso em Recursos...

## Deu Erroooooo, mas já resolvido

Deu erro: ***org.hibernate.tool.schema.spi.CommandAcceptanceException: Error executing DDL "create table [...]***

Descobrir mais p frente como arrumar, vai estar informado no Commit quando descobrir ( :/ )

Tela de computador com jogo

Descrição gerada automaticamente

Deu certo, testando conexão, incluindo alguns milhões de códigos, que não sei se alguns deles foram necessários, mas agora deu UHUUULsksksk

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

spring.datasource.url=jdbc:postgresql://localhost:5432/parking  
spring.datasource.username=postgres  
spring.datasource.password=admin  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=create-drop  
spring.jpa.properties.hibernate.jdbc.time\_zone=UTC  
spring.jpa.properties.hibernate.show\_sql=true  
spring.jpa.properties.hibernate.format\_sql=true  
spring.jpa.defer-datasource-initialization=true  
#

O começo não se aplica ao projeto pois tem haver com o BD Azure

Mas em questão de linkar o JPA e DLL é isso...

<https://learn.microsoft.com/pt-br/azure/developer/java/spring-framework/configure-spring-data-jpa-with-azure-sql-server>

# Após implementação Banco, Preparando camadas para requisições no banco!

Após feito todo o preparatório do banco

As requisições devem ser feitas através do banco e persistir nele

Para isso, Implementar o Service com JPA e o Repositório finalmente começa a funcionar

## Repositório finalmente começa a funcionar

@Repository  
public interface ParkingRepository extends JpaRepository<Parking**,** String> { }

## Implementar o Service com JPA

Após criado, vamos parar de utilizar a classe Map para as requisições

Dessa forma utilizar da classe, ou melhor, **interface Repository**

(*LOGO AGR QUE COMECEI A APRENDER SOBRE @AUTOWIRED ELE NÃO É MAIS UMA BOA PRÁTICA ~Stonks*)

Então... private final ParkingRepository parkingRepo;

E criar seu construtor

Implementar ele em vez do map, dessa forma

Texto

Descrição gerada automaticamente

Como a interface extende do JPA Repository,

O JPA já tem os métodos simples de um crud,

Dessa forma ele pode ser bem compactado

## @Transactional

No caso, ele serve caso estoure algum erro

Ele faz a tratativa de erro caso desse erro e não salvasse e desse algum problema, ele faria o rolback

Tem seus tipos de transaçãoTexto

Descrição gerada automaticamente

# Aplicando Spring Security (Salvando em memoria)

Lembrando que colocando dependência

spring-boot-starter-security

Ele já libera uma chave de acesso para entrar na aplicação

***Using generated security password: 7b287efa-c7b0-47ea-807f-107ef995375d***

## @EnableWebSecurity

Criar através do config

Uma nova classe, onde gera tokens para os usuários, ou de forma mais fácil,

Criar um usuário e senha em memoria

Texto

Descrição gerada automaticamente