

* Tabela de bikes: descrição, modelo, preço que pagou, data de compra, nome do comprador, nome da loja em que comprou

Estrutura: model + endpoint + service + repository

Model: Cria os argumentos + Construtores(sem o Id) + Getter/setters + Dependencia JPA + Dependency H2 + @Entity + @Id + @GeneratedValue(strategy = GenerationType.*AUTO*)

Obs.: Criei com Implements Serializable(não implementei variavel Estatica) + toString + hashCode/Equals mas depois tirei porque neste momento não precisa.

EndPoint: @RestController + @RequestMapping("v1") + @Autowired   
private BikeService bikeService; + @...mapping

Service: @Service + .......

Repository: interface BikeRepository extends JpaRepository<Bike, Long>

TestConfig: implements CommandLineRunner para criar dados iniciais

Application.properties + application-test.properties

Commitei versao basica.

----------------------------------------------------------------------------

Persistir dados Adicionar Banco Postgree

Adicionado dependencia no .pom

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

+ @Table(name = "tb\_bike") + Adicionado application-dev.properties + Alterado application.properties para spring.profiles.active=dev

----------------------------------------------------------------------------

Validar os Dados

Adicionado dependencia em .pom

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

Adicionado anotações conforme o tipo de dado no model bike @NotBlank, @Past, @ etc

Adicionado @Valid no end point

@PostMapping(path = "/bikes")  
public ResponseEntity<?> insert(@Valid @RequestBody Bike bike)

Referencia:

SpringBootValidation.docx --> <https://onedrive.live.com/edit.aspx?resid=B441B70F8E9D4017%2158302&nd=1>

----------------------------------------------------------------------------

Adicionar Segurança em memoria

----------------------------------------------------------------------------

Adicionar Exceptions

----------------------------------------------------------------------------

Adicionar testes unitarios

----------------------------------------------------------------------------

Adicionar Segurança com Banco de Dados

----------------------------------------------------------------------------

Refatorar Testes unitarios

----------------------------------------------------------------------------

Melhorar as Tabelas adicionando outras(dividir a existente)

----------------------------------------------------------------------------

Refatorar Testes unitarios

----------------------------------------------------------------------------

Publicar no Heroku

----------------------------------------------------------------------------

Pendente:

* Usar as features do Java 8/11 de streams apropriadamente.

Após terminar:

Colocar versao para banco MySQL

Adicionar segurança com token

Refatorar os testes com token

Publicar na AWS DigitalOcean e GCP

Aprenda a usar o SonarLint(para inspecionar o código escrito), Travis-ci(Configurar o travis-ci na sua máquina para garantir a alta qualidade de suas entregas(Automação das entregas com Travis-ci)), Lombok, Swagger

Spring HATEOAS.

Realizar cache das operações

Avaliar a cobertura de código testado com o coverage.