Igor Nascimento

Persistência com Java

Spring Boot

Primeiramente inicializamos um projeto spring no site  “<https://start.spring.io/>” configuramos os metas dados na parte de project metadata assim como o nome dos pacotes da aplicação.

O próximo passo é colocar todas as dependências necessárias para o spring baixar todas as bibliotecas necessárias para o funcionamento da aplicação utilizaremos as **Spring Web,**

**Spring Data JPA, Validation e como banco de dados utilizamos o PostgreSQL Driver.**

Fazemos o download do zip extraímos em uma pasta e abrimos com uma IDE, logo depois

iremos configurar nossa conexão com o banco de dados no arquivo application properties utilizei essas configurações:

Texto

Descrição gerada automaticamente

Depois dessa configuração feita iremos criar nosso pacotes e classes MODEL, CONTROLLER, SERVICES, REPOSITORY,

Em model iremos criar todas as nossas classes modelos e implementamos a interface Serializable

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

modelo de uma tabela criada

Quando criamos os atributos da classe devemos colocar algumas anotações muito importantes e parecidas com criação de uma tabela no banco de dados..

Texto

Descrição gerada automaticamente

logo depois devemos criar os métodos assessores dos atributos

O Jparepository existem vários métodos prontos para manipulação do banco de dados iremos criar em repository uma interface para ter acesso a esses métodos já prontos

O Servisse iremos criar uma classe com a anotação @Service que ela será a camada intermediária entre o controle e o repository usando injeção de dependência  devemos criar um ponto de injeção para essa comunicação acontecer com spring temos muitas formas de fazer isso podemos  usar a anotação @Autowired ou fazer pelo construtor

Texto

Descrição gerada automaticamente

Logo após devemos criar o nosso controle que irá servir de comunicação com o service devemos criar nosso controller com alguma anotações como é uma Api Rest utilizaremos o @RestController e usaremos as anotações de de @CrossOrigin para permite que seja acessado de qualquer fonte e sem esquecer a URI a nível de classe para acesso desse recurso utilizaremos a /parking-spot como a seguir:

Texto

Descrição gerada automaticamente

Para validar da melhor forma vamos criar um DTO para a nossa aplicação e vamos criar uma classe com vario campos que existe na classe de modelo usamos a anotação @NotBlank para verificação se está vazio se o campo é válido e o @Size para limitarmos os caracteres.

Texto

Descrição gerada automaticamente

Logo com toda essa estrutura pronta podemos dar início aos métodos GET, PUT, POST E DELETE, Começando com o método POST de primeira instância pra isso utilizaremops a notação   @PostMapping vamos fazer uma validação simples com if’s para verificar se tem registro em nosso banco de dados iguais aos que vamos cadastrar inicializamos esses métodos na interface de REPOSITORY   onde a classe SERVICES irá consumir através da injeção de dependência

Texto

Descrição gerada automaticamente

imagem dos métodos abstratos da classe repositories

Texto

Descrição gerada automaticamente

imagem dos métodos implementados na classe service.

Texto

Descrição gerada automaticamente

Método POST concluído

O método Get é mais simples vamos fazer 2 tipos de método GET um que irá trazer todos os cadastros e o outro que irá fazer uma pesquisa por ID,  no método  que irá trazer tudo ele terá o retorno de uma List do tipo da nossa classe modelo irá usar o ResponseEntity para pegar o status http dentro do body irá fazer uma consulta ao serviço que também terá que ter um método chamado findAll() que vem da injeção de dependência da interface Repository que tem todos esses métodos já pronto pois estende de JpaRepository,

no GET por id seguimos o mesmo pensamento mas com uma pequena diferença será passado um id na anotação do método e como parâmetro também iremos usar a classe OPTIONAL para usar o método GET desta classe também teremos que cria o método findById(id) na classe Serviço que seguirá o mesmo raciocínio do método anterior teremos um if para podermos fazer uma verificação se o resultado obtido é realmente válido se passar por esse if vamo da uma hhtp ok e usaremos o método GET da classe Optional para retornamos o valor.

Texto

Descrição gerada automaticamente

métodos get

Texto

Descrição gerada automaticamente

métodos criados na classe de serviço

O método de DELETE é quase igual ao método de consultar por id com uma diferença invés de ele fazer um get irá fazer um delete

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

método delete

Texto

Descrição gerada automaticamente

método consumido no service

Método PUT receberá um parâmetro id em sua anotação e dois parâmetro no método o id e a classe  que será alterada será tratada como os outros métodos o método irá fazer uma busca no banco de dados pelo id e logo após irá verificar se realmente o cadastro existe depois devemos passar os dados coletados para a nossa classe modelo podemos fazer isso de várias formas chamando o método SET da classe modelo passando o método GET da classe DTO

Texto

Descrição gerada automaticamente

1-forma de setar os atributos

E a outra forma e fazendo uma cópia da classe usando o  BeanUtils.copyProperties(parkingSpotDto, parkingSpotModel) onde ele irá fazer uma cópia da classe DTO para a classe Modelo porém ainda temos que usar os métodos sets para setar o id da classe e a hora do registro

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente