Autor: Lucas Gonçalves Lacerda

Data: 01/01/2021

Criando uma API REST para abertura de contas em um banco com Spring.

Vamos construir uma API que basicamente nos permite entrar com os dados pessoais necessários do cliente e assim abrir uma conta bancária, retornar uma resposta sobre a integridade dos dados, status do cadastro e etc.

Ao utilizar a API o cliente vai nos enviar suas informações através de uma requisição, assim retornaremos apenas o resultado da nossa análise perante os dados informados. Para uma requisição de cadastro vamos utilizar o método POST e como base de informação o body a seguir:

***POST/apibanco/conta***

{

"id": 1,

"nome": "Lucas Gonçalves",

"email": "lucalacerdajob@gmail.com",

"cpf": "12345678912",

"nascimento": "2001–05–29T09:59:44.452Z"

}

Caso ocorra alguma falha, a API deve então retornar os devidos códigos de acordo com a validação feita:

* **200**: Em caso de sucesso e conta criada;
* **400:** Em caso de erro do lado cliente (JSON invalido - Bad requisition);
* **409**: Caso em que, determinado campo não esteja de acordo como esperado nas regras de negócios;

Os códigos de respostas ficam a seu critério ou preferência de escolha, entretanto existe alguns grupos/intervalos já padronizados para devidas respostas, são conhecidas como classes:

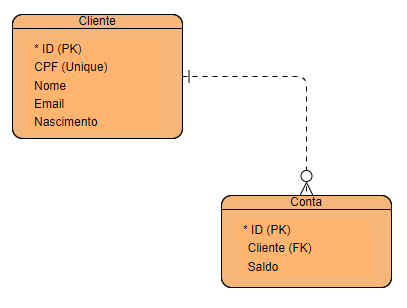
1. Classe 1xx: Respostas de informação (100-199),
2. Classe 2xx: Respostas de sucesso (200-299),
3. Classe 3xx: Redirecionamentos (300-399)
4. Classe 4xx: Erros do cliente (400-499)
5. Classe 5xx: Erros do servidor (500-599).

As classes são padronizadas pela IETF (Internet Engineering Task Force), grupo responsável por basicamente definir padrões, resolver e identificar problemas relacionados à internet entre outras coisas, caso queira, você pode escolher outros status code para sua necessidade de acordo com o padrão de classes já predefinidos.

Modelagem

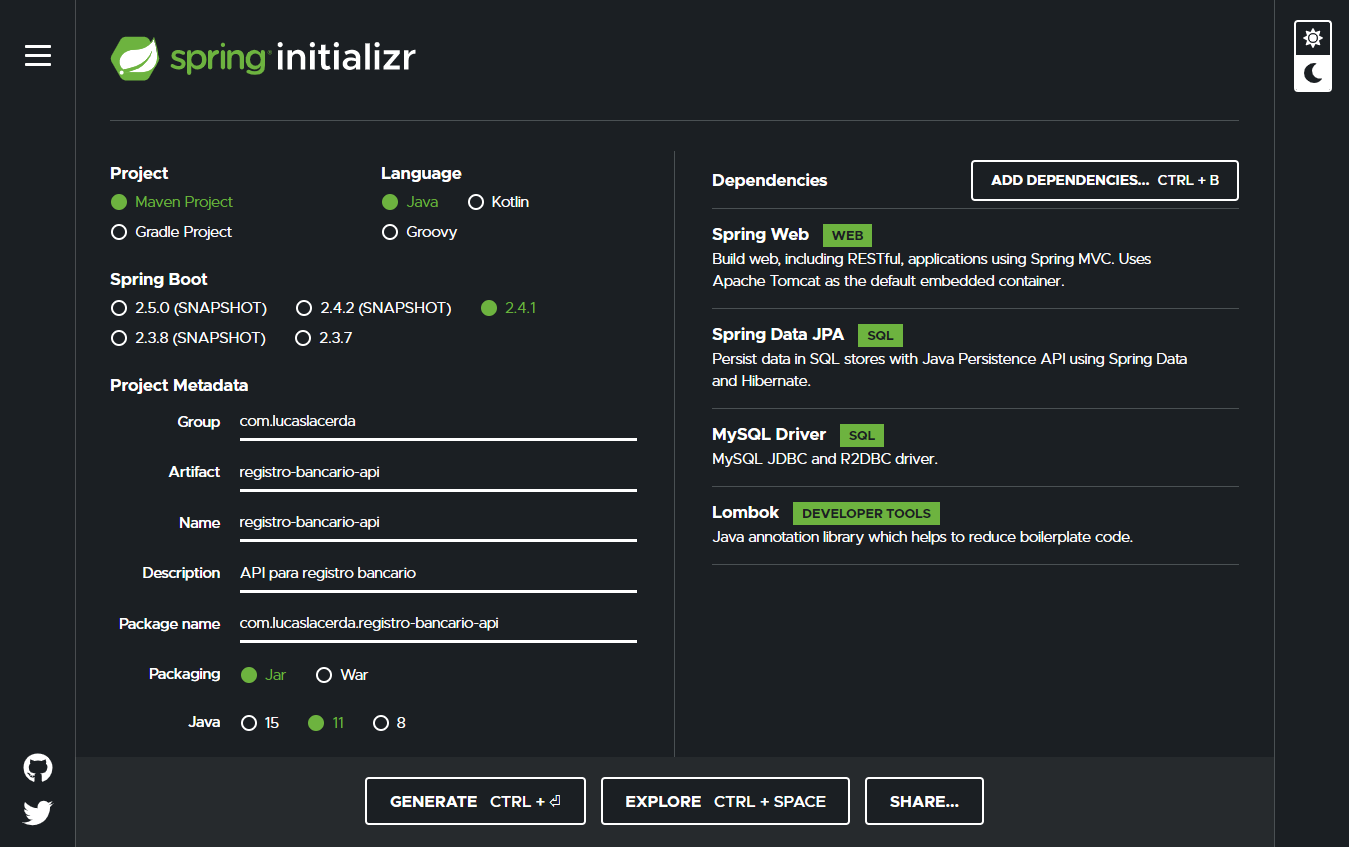
Para realizar o fluxo de cadastro e abertura de contas bancárias, temos que trabalhar com duas entidades, “Cliente” e “Conta”: A entidade ‘Cliente’ tem como chave primária o campo ‘ID’ que será gerado pelo Hibernate, e o CPF como Unique pois um cliente não pode ter um mesmo CPF de outro cliente já cadastrado.

Vamos criar também a entidade “Conta”, possuindo um ‘ID’ como PK que representará o número da conta, e o campo ‘Cliente’ como Foreign Key representando o ‘ID’ do cliente responsável pela conta em questão.



Criação do projeto

Para construir o projeto Spring Boot que facilita toda uma configuração referente ao projeto, podemos utilizar o Web Site <https://start.spring.io/>, ele vai pedir algumas informações relacionadas ao projeto a ser criado e assim gerar um .zip no final, bastando apenas importar para a IDE de sua preferência simplificando e facilitando toda uma criação de projeto e configurações.



Em ‘Project’ vamos escolher “Maven Project”, Maven é uma ferramenta que vamos utilizar caso precise adicionar mais alguma dependência durante o desenvolvimento, basicamente quando utilizamos o Maven não precisamos baixar arquivos e realizar configurações para novas dependências, apenas acessar sites que possuem repositórios, como o <https://mvnrepository.com/>, você busca a dependencia, copia algumas linha de texto (tags de configurações) e cola em um arquivo específico referente ao Maven no projeto (pom.xml) assim o Maven vai automatizar o Build da dependência inserida no arquivo ‘pom.xml’. Exemplo:

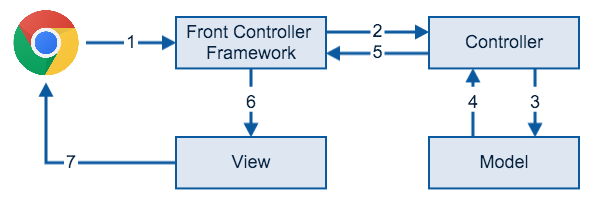


Caso queira adicionar uma outra dependência que ache necessária, basta copiar e colar as tags do box em seu arquivo pom.xml (Project Object Model) do projeto.

Retornando a página de criação do projeto, vamos manter Java como a linguagem a ser utilizada e a versão 2.4.1 do Spring Boot. O ‘Metadata’ fica a seu critério definir, é recomendado utilizar boas práticas em nomes e descrições do projeto.

Em Dependencies, vamos utilizar:

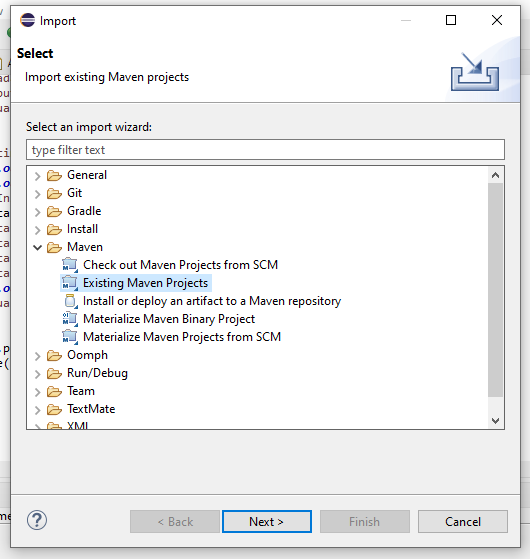
* Spring MVC: O Spring MVC é uma estrutura que vai nos auxiliar e ajudar durante o desenvolvimento, pois ela tem várias funcionalidades e ferramentas voltadas para o padrão de projeto MVC (Model- View - Controller), modelo na qual vamos utilizar:



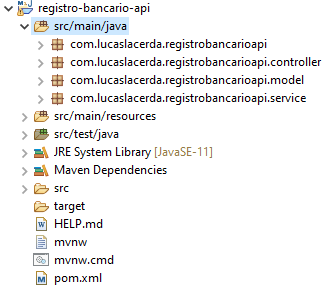
O diagrama nos mostra todo o processo de um requisição quando utilizamos o Spring MVC, basicamente o ‘Front Controller Framework’ é uma classe denominada como ‘DispatcherServlet’. A class é responsável por receber a requisição inicial, em seguida identificar quem é o responsável pela requisição e mapear para onde a mesma deve prosseguir, seja Controller, View, Model...

* JPA: A dependência JPA é necessária para possibilitar o uso do Hibernate na nossa aplicação, basicamente o JPA nos fornece ferramentas para mapear objetos relacionais do nosso banco de dados através de annotations dentro do nosso próprio código fonte da API, como também nos possibilita fazer diversas persistência de dados apenas com annotations no código.
* MySQL Driver: É um driver do tipo JDBC (Java Database Connectivity) responsável por nos permitir que API se comunique e interaja com o banco de dados que vamos utilizar, que no caso o MySQl, podemos então enviar qualquer instrução SQL necessário para o banco através da conexão que o Driver nos abre.
* Lombok: Lombok é uma ferramenta para nos ajudar com ‘assistência de código’, produtividade e afins, como a criação de getters, setters, Construtores entre outros (fica a seu critério utilizar ou não).

Após preencher todos os campos, podemos gerar o projeto em “Generate”, assim irá gerar um ‘.zip’. Em seguida, podemos importar para a IDE de preferência, no caso vou utilizar o Eclipse.

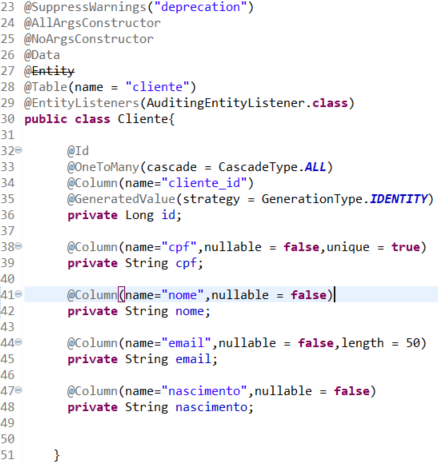


Após importar o Projeto Maven percebesse que o Download das dependências irá se inicializar automaticamente, e em seguida podemos criar os pacotes do nosso modelo de projeto:

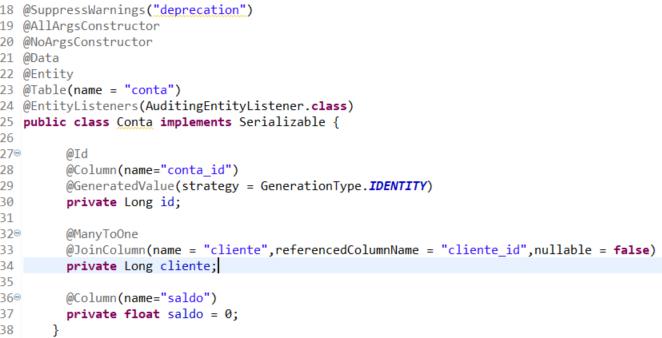


Vamos começar a criar o nosso model, como demonstrado no tópico de Modelagem, devemos criar duas entidades ‘Cliente’ e ‘Conta’:

Obs.:Existem algumas alternativas para construir nosso banco de dados, podemos deixar que o Hibernate crie todos os campos a partir de um mapeamento com annotations no código ou então montamos nosso próprio SQL, como nossa estrutura não é muito grande vamos deixar o Hibernate criar o comando DDL e executar de acordo com o nosso mapeamento.



Model ‘Cliente’



Model ‘Conta’

Utilizamos algumas annotations para construir nosso model, são elas:

@AllArgsConstructor: O Lombok, ferramenta que adicionamos na criação do projeto, cria automaticamente o construtor da classe com todos os argumentos em questão, basta inserir a anotação.

@NoArgsConstructor: Muito parecida com a anotação anterior, entretanto cria o construtor de classe vazio.

@Data: Cria automaticamente todos os getters e setters, como também toString;

@Entity: É uma anotação do Hibernate, ela indica que a class é uma entidade;

@Entity: Indica qual tabela nossa entidade vai representar no banco de dados;

@Conlumn: Onde podemos definir algumas propriedades da coluna no banco de dados,name, length, nullable entre outros;

@JoinColumn: Indica uma ligação entre tabelas, no caso, a coluna ‘cliente’ é um campo que faz a ligação com o ‘cliente\_id’ detentor da conta bancária (Foreign Key);

@ManyToOne: Indica qual tipo de relação, no nosso caso vai ser ‘Muitos para uma’,ou seja, existem mais de uma conta para um mesmo cliente;

@OneToMany: Indica qual tipo de relação,ou seja, um cliente pode ter mais de uma conta no banco.

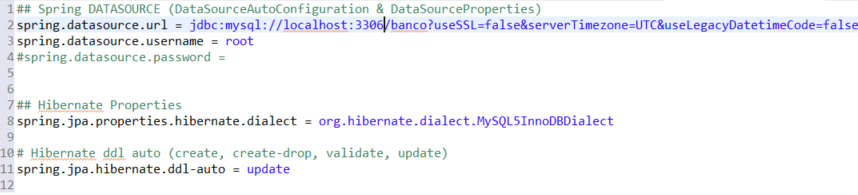
@ID: A anotação ‘@id’ indica qual campo da entidade vai representar a Primary Key da tabela em questão, situada no banco de dados;

@GeneratedValue: Generated Value é onde devemos indicar ao JPA qual tipo de estratégia queremos usar para construir a chave primária automaticamente através do Hibernate, existem 4 estratégias para definir o valor da chave:

* GenerationType.AUTO: Quando não definimos nenhuma estratégia, o banco escolhe como vai ser gerada (Cada banco de dados tem sua estratégia padrão já pré definida), da mesma forma acontece quando definimos como “AUTO”, não alteramos o modo em que a chave vai ser construída (fica na responsabilidade do banco definir).
* GenerationType.IDENTITY: Quando definimos que a estratégia a ser utilizada seja IDENTITY, a partir do AUTO\_INCREMENT o Hibernate vai gerar as chaves (Obs.: utilizando MySQL).
* GenerationType.SEQUENCE: O Hibernate vai usufruir das sequences do próprio banco de dados.
* GenerationType.TABLE: O Hibernate vai montar as chaves através de um tabela que armazena as chaves a serem utilizadas, então podemos usar essa mesma tabela para entidades ou uma entidade específica.

Assim temos um leque de opções para implementar, no nosso caso vamos utilizar a IDENTITY, caso queira utilizar outra não se incomode, elas devem ser escolhidas de acordo com o que achar mais adequado para a situação.

Após criar o model, devemos passar as informações de acesso ao banco para nossa api, percebesse que no pacote ‘src/main/resources’ temos um arquivo ‘application.properties’, vamos então inserir algumas informações necessária:



O JDC,username e password são informados no worchebend do MySQL. Em localhost você deve inserir o seu ip como também o usuário e senha configurados no MySQL.

“DDL-auto” é propriedade que podemos configurar, quando iniciamos o banco de dados ela determina qual procedimento realizar em toda base de dados mapeada (update,create,validate...). Exemplo: Se configurarmos para “Create”, todas vez que iniciarmos o serviço, todas as tabelas e campos mapeados serão recriados, assim perdemos todos os dados já gravados no banco.

No caso, iremos utilizar o “update”, que apenas verifica se houve alguma mudança no mapeamento e aplica ao iniciarmos.