

**PROPOSTA**

O **primeiro passo** deve ser a construção de um cadastro de usuários, sendo obrigatório dados como: nome, e-mail, CPF e data de nascimento, onde e-mail e CPF devem ser únicos.

O **segundo passo** é criar um cadastro de endereços, sendo obrigatório dados mínimos para cadastro como: logradouro, número, complemento, bairro, cidade, estado e CEP, associando este endereço ao usuário.

O **terceiro passo** é criar um endpoint que retornará um usuário com a lista de todos seus endereços cadastrados.

**REGRAS DO NEGOCIO**

- Caso os cadastros estejam corretos, **é necessário** voltar o Status 201, caso haja erros de preenchimento de dados, o Status deve ser 400.

- Caso a busca esteja correta, **é necessário** voltar o status 200, caso haja erro na busca, retornar o Status adequado e uma mensagem de erro amigável.

Hoje vamos criar uma API REST utilizando algumas das principais ferramentas disponíveis no mercado. Escolhemos o JAVA com linguagem de programação, além de Spring + Hibernate como Stacks.

Atualmente é possível criar um projeto e subir em pouco minutos e com auxilio de algumas ferramentas podemos tornar este processo ainda mais fluido e simples.

Tecnologias que utilizaremos neste projeto:

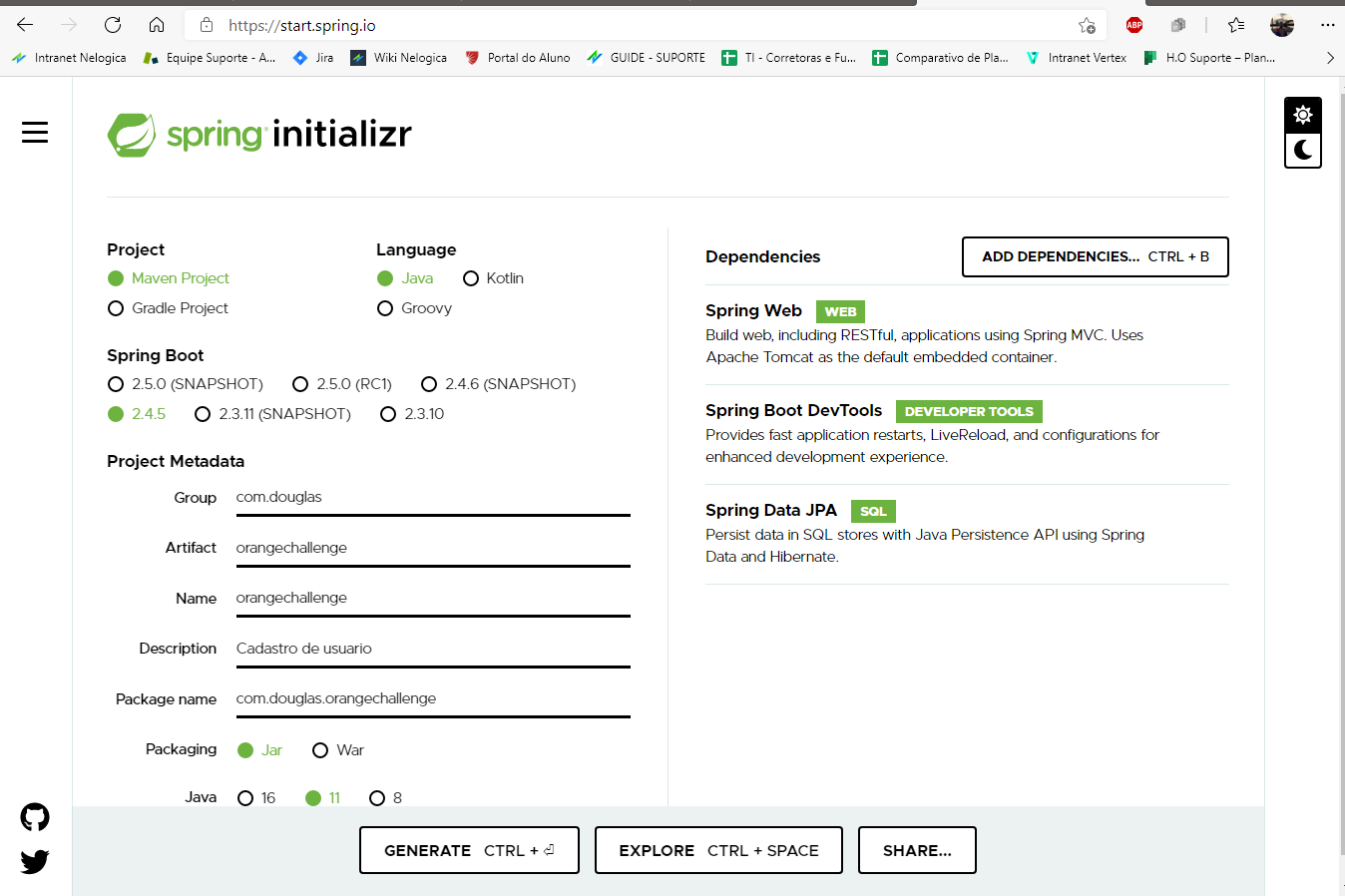
* Spring Web – Vai nos permitir criar um Web Service;
* H2 Database – Banco de dados fornecido pela própria IDE;
* Spring Boot DevTools – Auxiliará durante o processo de desenvolvimento, economizando tempo com o recarregamento do projeto;
* Spring Data JPA – Utilizaremos para implementar recursos do Spring Data e Hibernate;
* Validation – Necessário para implementação de algumas validações.

Ferramentas utilizadas:

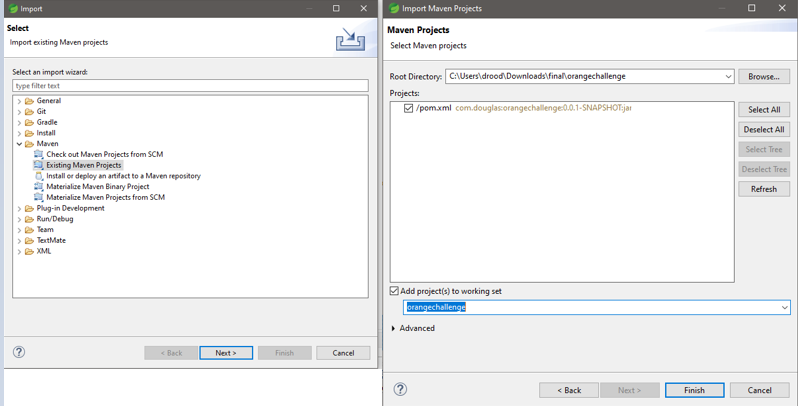
* IDE: Spring Tool Suite 4;
* H2 Database: Incluído na própria IDE;
* Postman;
* Stackoverflow: Todo mundo sabe que esse é fundamental.

Então vamos ao projeto! Iniciaremos a criação e importação das dependências necessárias. Para tanto, podemos criar o projeto diretamente na IDE ou acessando o site do Spring Initialzr: [Spring Initializr](https://start.spring.io/).

Ele vai permitir uma criação rápida de um projeto com todas as dependências necessárias. Basta preencher as informações como Tipo de projeto, Linguagem, Dependências, versão e etc. Conforme imagem abaixo:



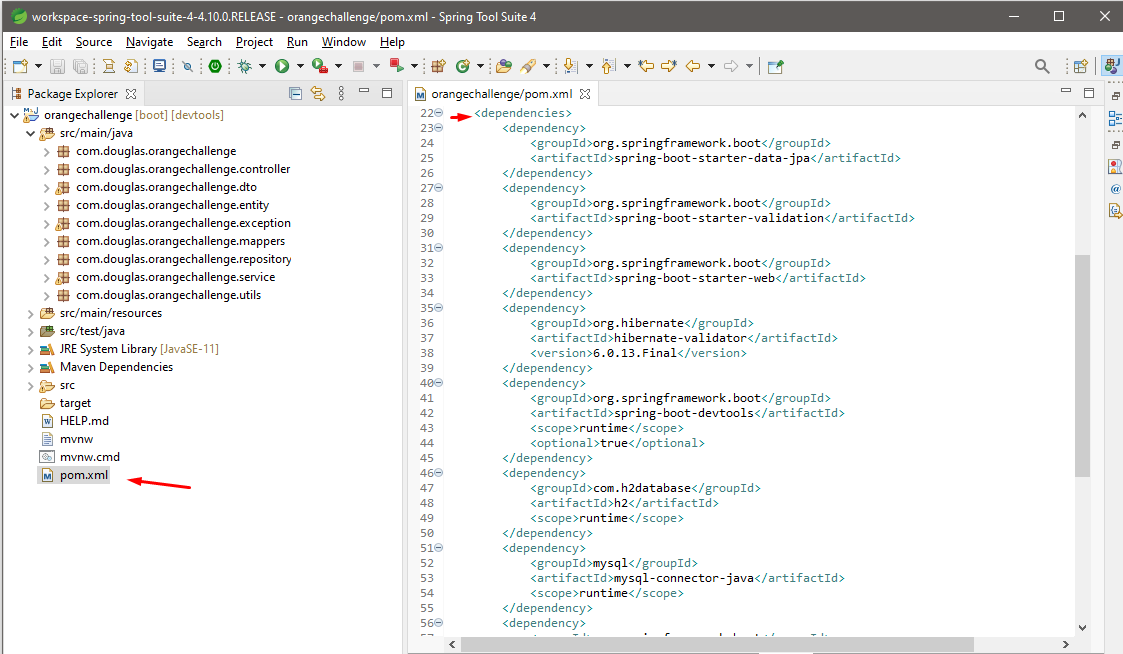
Basta clicar em Generate e será gerado um arquivo Zip do projeto, após extrair podemos importar para a IDE. Acessando File > Import > Maven > Existing Maven Projects, busque pelo projeto exportado e finalize:



Feito isto, apenas aguardamos as dependências serem carregadas, elas podem levar alguns minutos dependendo do computador utilizado.

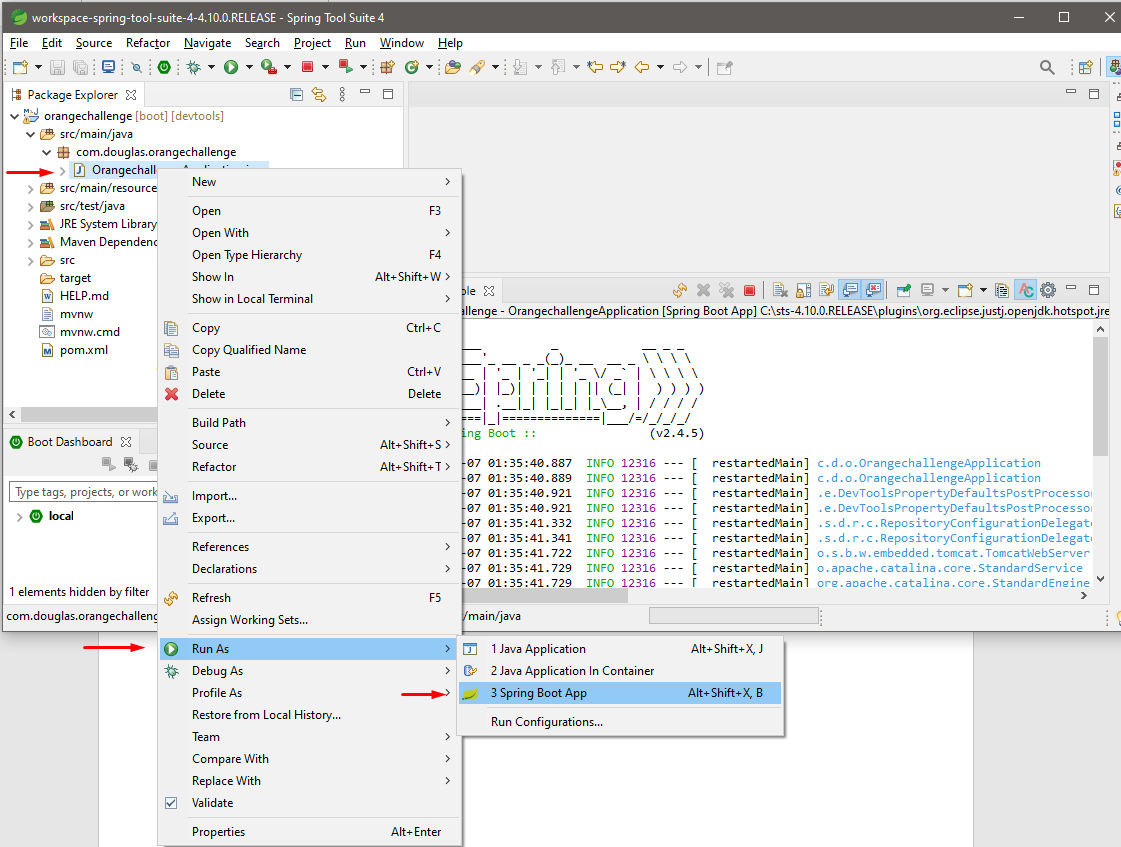


Após finalizar o carregamento, podemos verificar se todas as dependências foram importadas corretamente. Elas ficarão acessíveis no arquivo pom.xml (Project Object Model).



Antes de pôr a mão na massa, vamos rodar o projeto pela primeira vez. Clicando sobre a classe Application com o botão direito do Mouse > Run As > Spring Boot App.

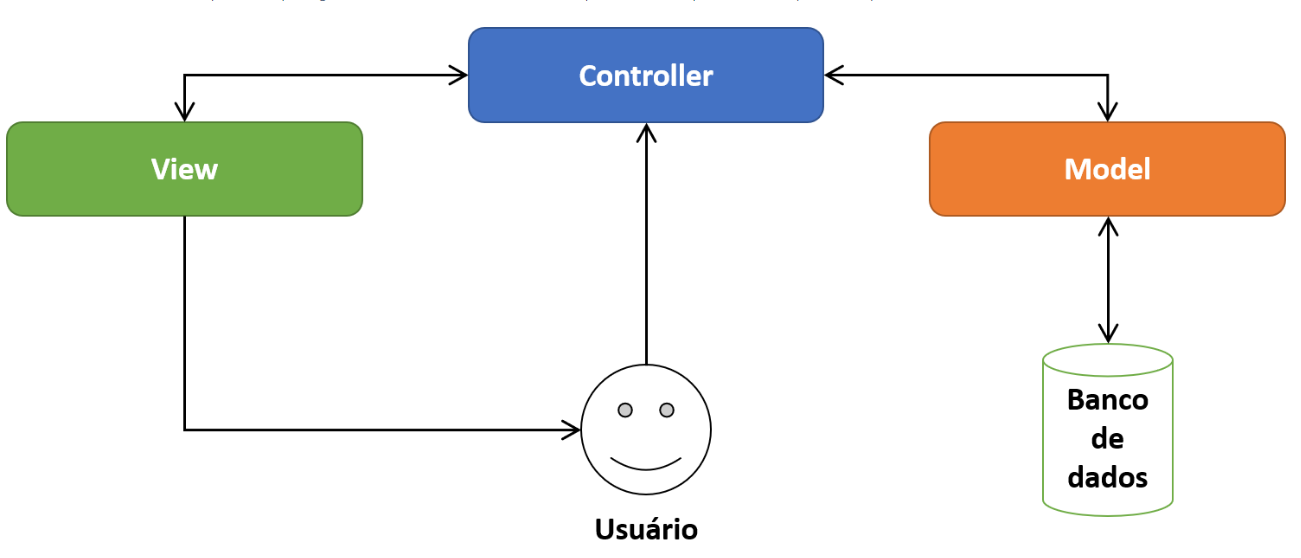
E aqui já podemos entender a vantagem de utilizar o Devtools, após iniciar o projeto pela primeira vez, a cada nova modificação que for salva no projeto, ele reiniciará o serviço, o que acaba otimizando nosso tempo de programação.  
  
Você deve visualizar este resultado no Console:

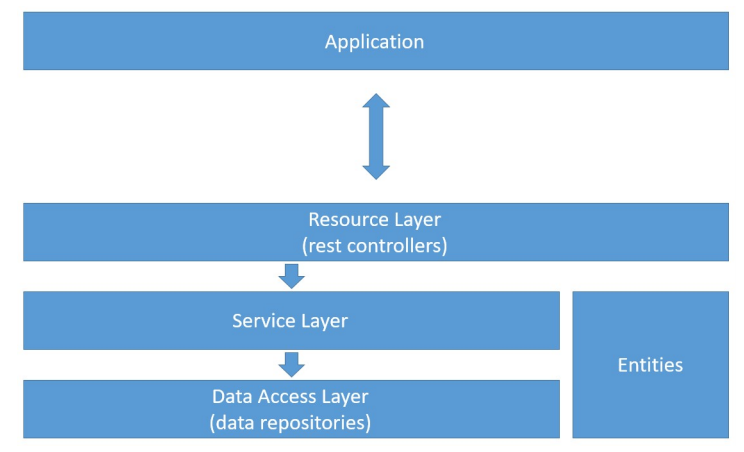


Agora com nosso objetivo em mente, a criação de uma API para cadastrar usuários, endereços e por fim, listar endereços de um usuário específico, vamos estruturar e definir um padrão para iniciar a programar.

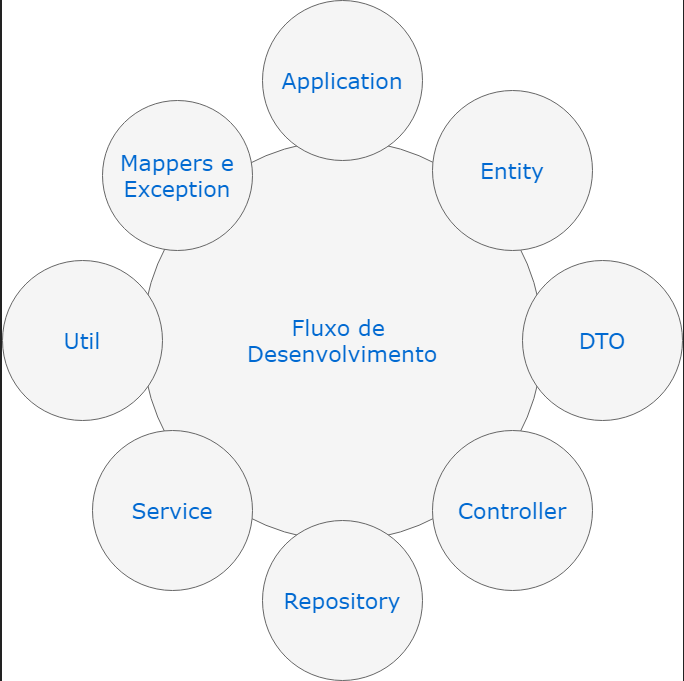
Dada a simplicidade estrutural, vamos adaptar o padrão MVC sem a camada View, pois vamos trabalhar apenas o Back-end neste projeto.

Sendo assim, as interfaces não serão implementadas, apenas nosso Model que será o responsável das regras de negócios, persistência no banco de dados e acesso as classes de entidades, bem como o Controller que encaminhará as requisições para o Model e retornará as respostas.

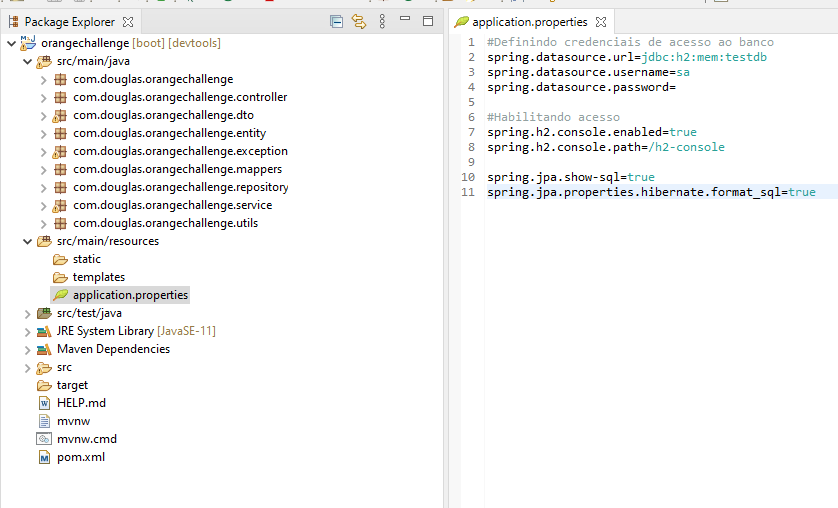




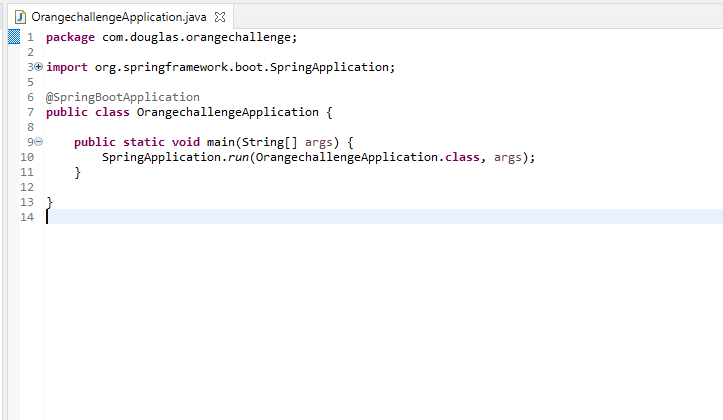
Durante a implementação do projeto teremos diversos pacotes e classes, portanto vamos ver o fluxo de implementação de cada pacote antes de analisar as classes. Está será a ordem que você poderá implementar em seu projeto, pois em seguida vamos apenas ver as classes de maneira isolada. Contudo, isto pode variar de projeto para projeto.



Mas a pergunta que não quer calar. A gente já pode programar? Ainda não! Pois falta falar de uma parte muito importante que devemos configurar em nosso projetinho, estou falando do application.properties. Ele vai ser o responsável por carregar as configurações de acesso ao banco de dados, aqui devemos informar as credenciais de acesso, url e ativar ou desativar o banco H2.

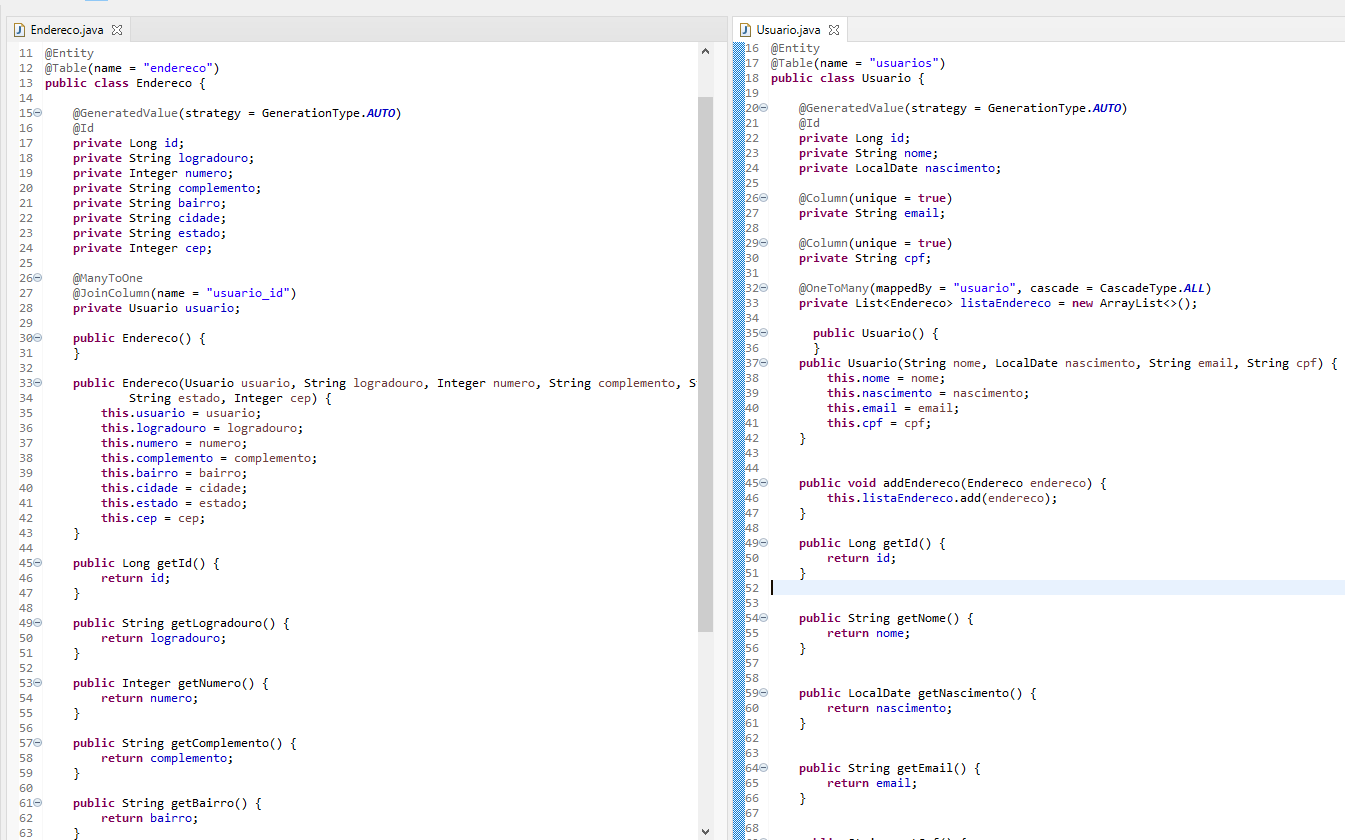


Agora sim! Vamos ao projeto, iniciando pela classe principal que não terá muitas configurações, apenas a Anotation @SpringBootApplication.

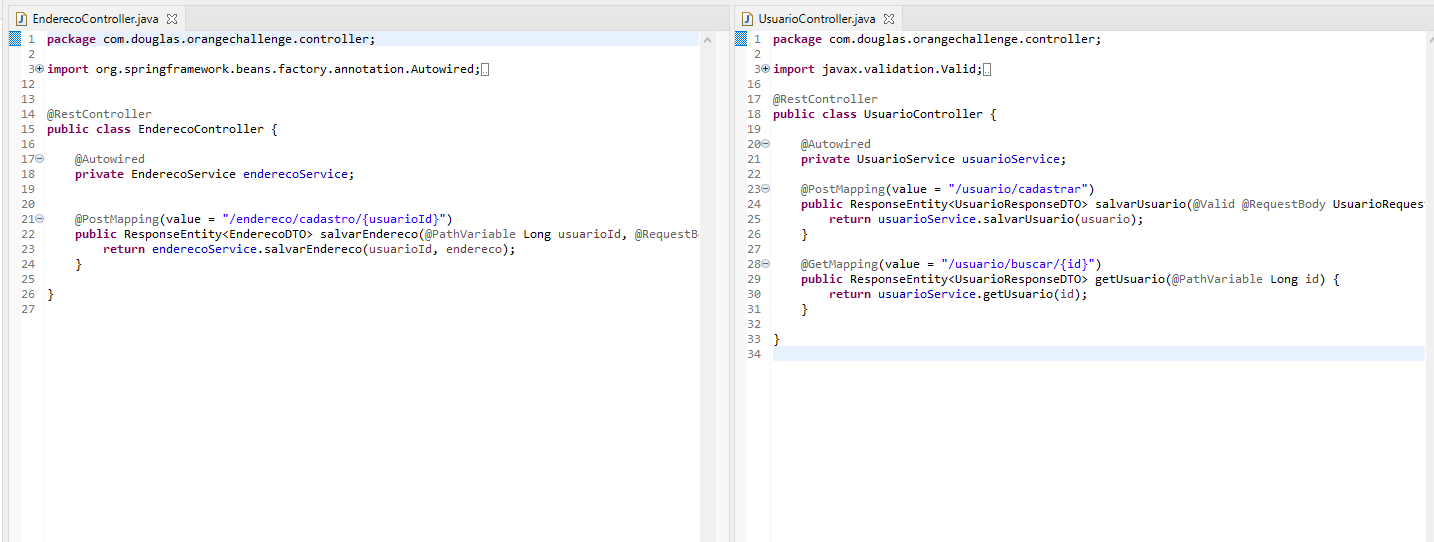


Dentro do pacote Entity, temos as entidades Endereco.java e Usuario.java. Implementaremos apenas os métodos Get. Também faremos algumas validações para que os campos email e cpf sejam únicos no banco de dados, evitando cadastros duplicados.

Além da associação das tabelas de Endereço e Usuário através das anotações @JoimColumn e @ManyToOn/OneToMany responsável por indicar os relacionamentos entre ambas.

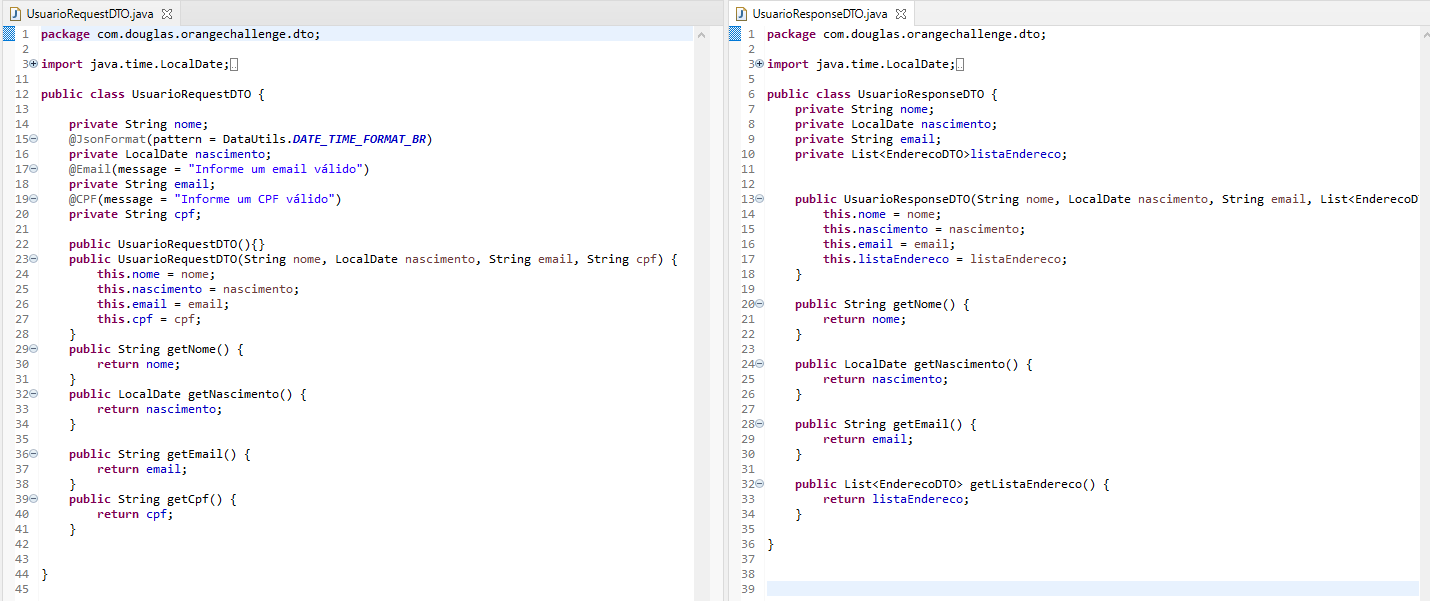
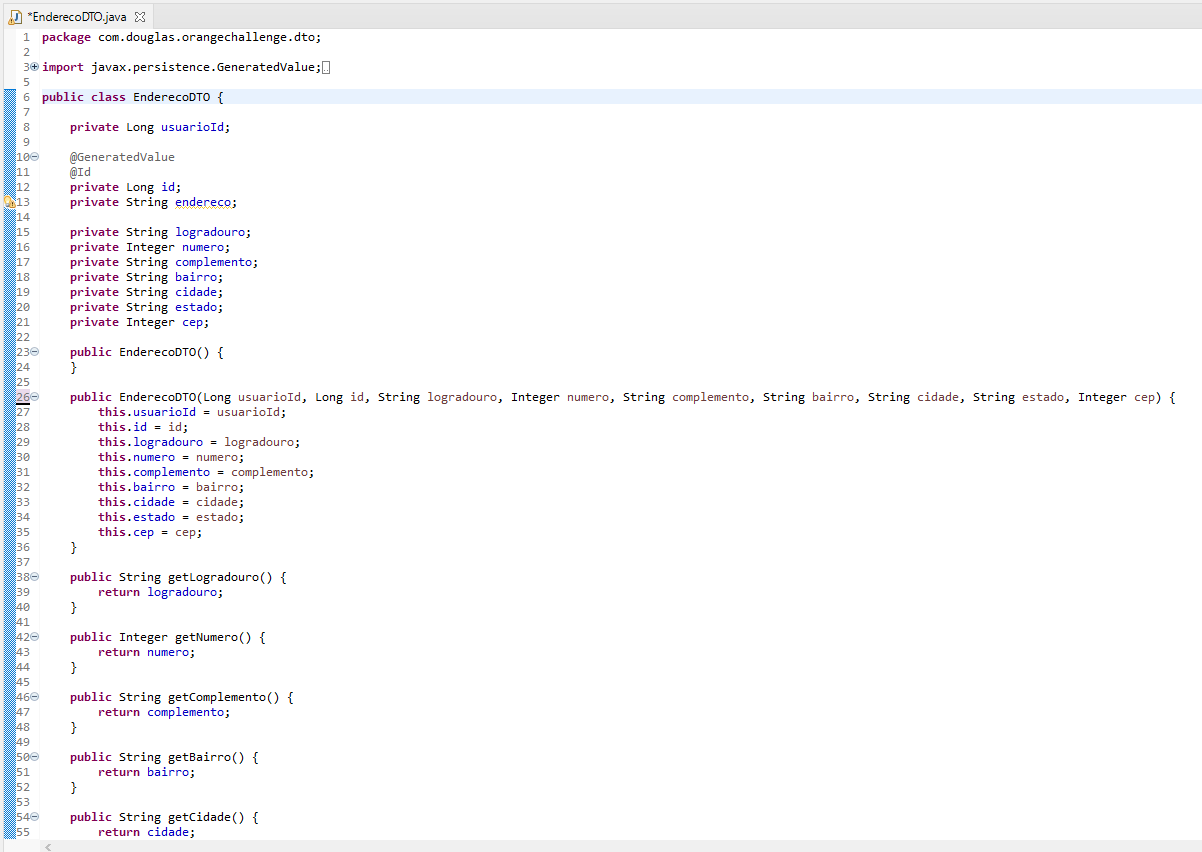


Acessando o pacote Controller, temos as classes EndereçoController e UsuarioController, onde faremos a injeção de dependências indicada pela Anotation @Autowired e o mapeamento de solicitações HTTP POST nos métodos indicados.

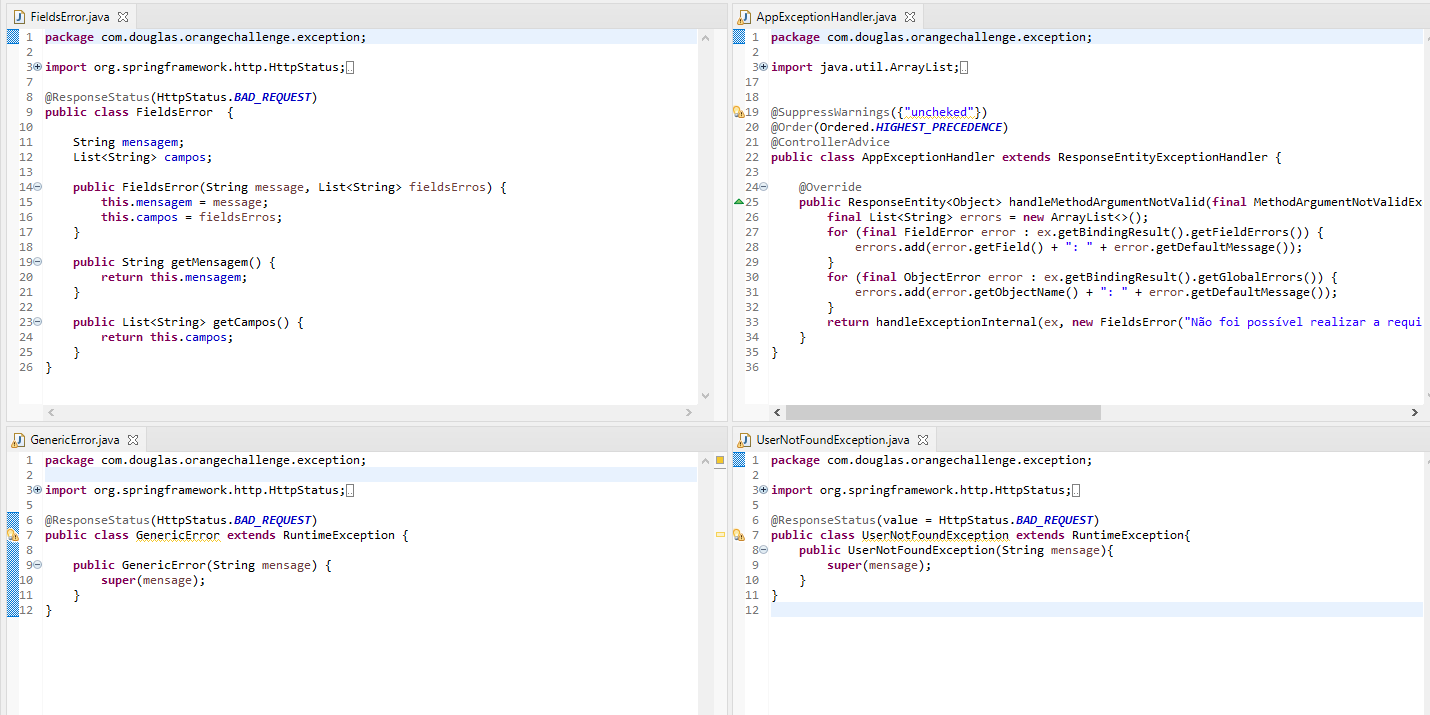


Logos após, seguimos para o pacote DTO (Data Transfer Object), eles serão responsáveis por transferir apenas os dados necessários para cada operação, assim otimizando nosso projeto.

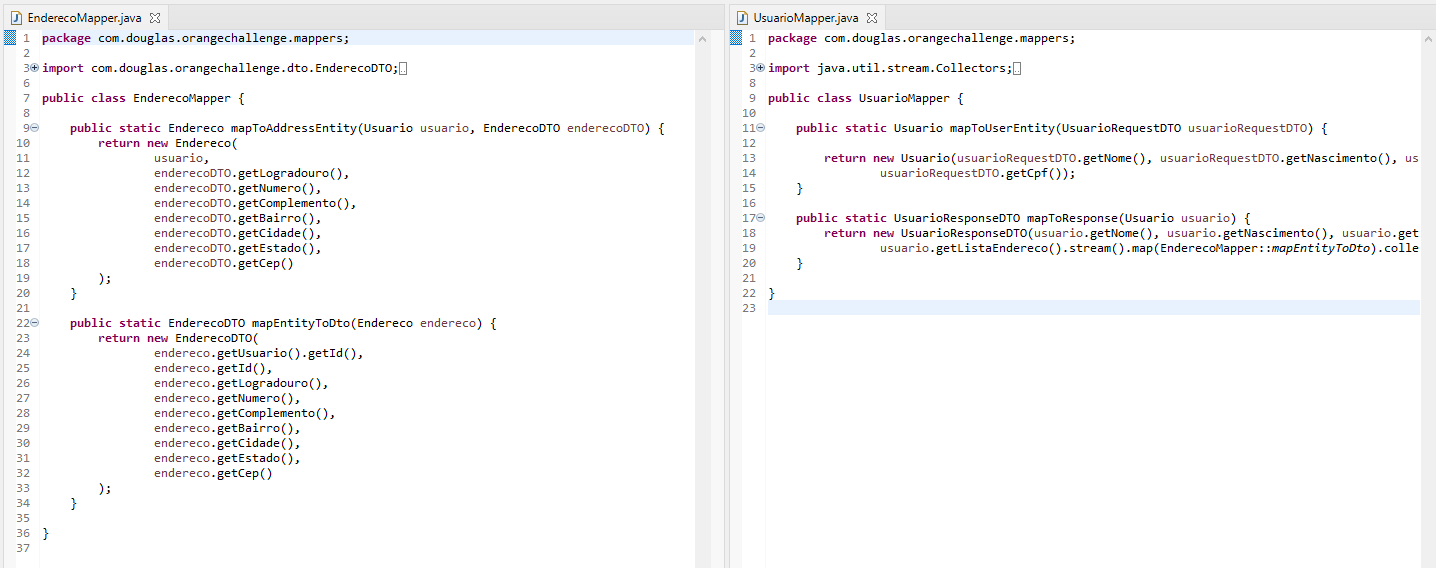
Aqui também iremos efetuar algumas validações como formatação da data para utilizar o padrão local “dd-MM-yyyy”.

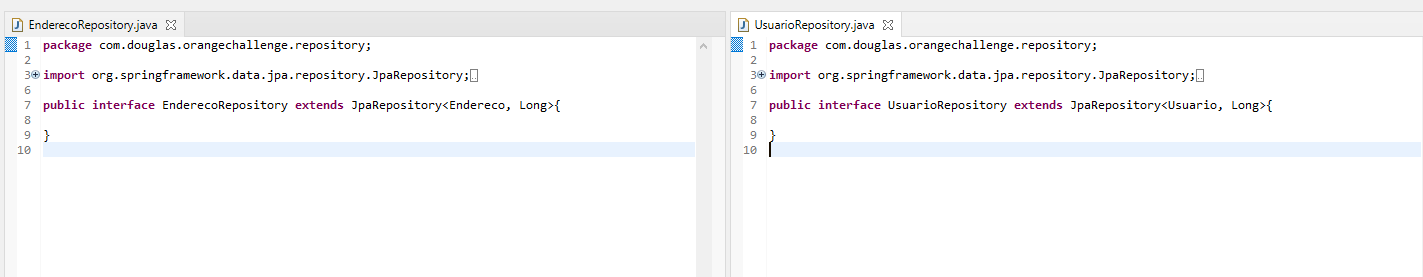
  


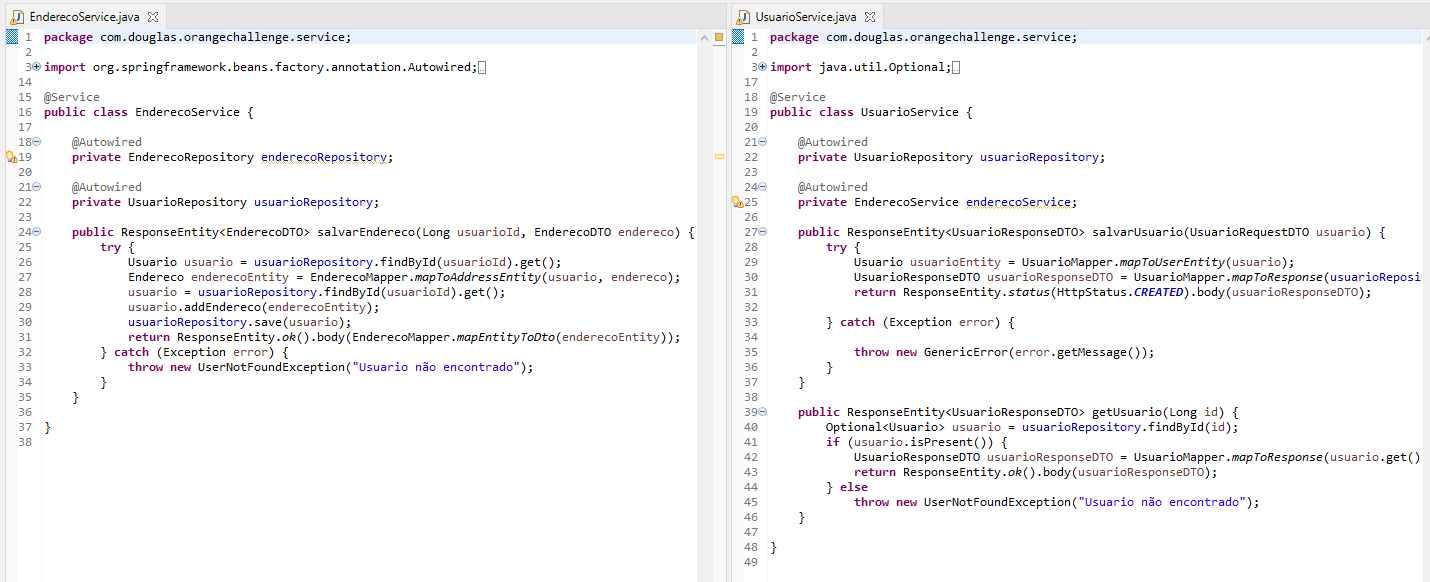
Contudo, a lógica por trás das validações será implementada através das classes de validação. Que retornarão mensagens amigáveis quando uma operação for bem ou mal sucedida.

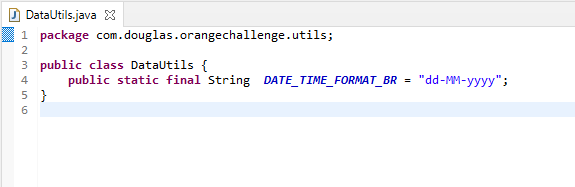


Para facilitar o mapeamento dos métodos, criamos o pacote Mapper, onde foram incluídas das classes de UsuarioMapper e EnderecoMapper.

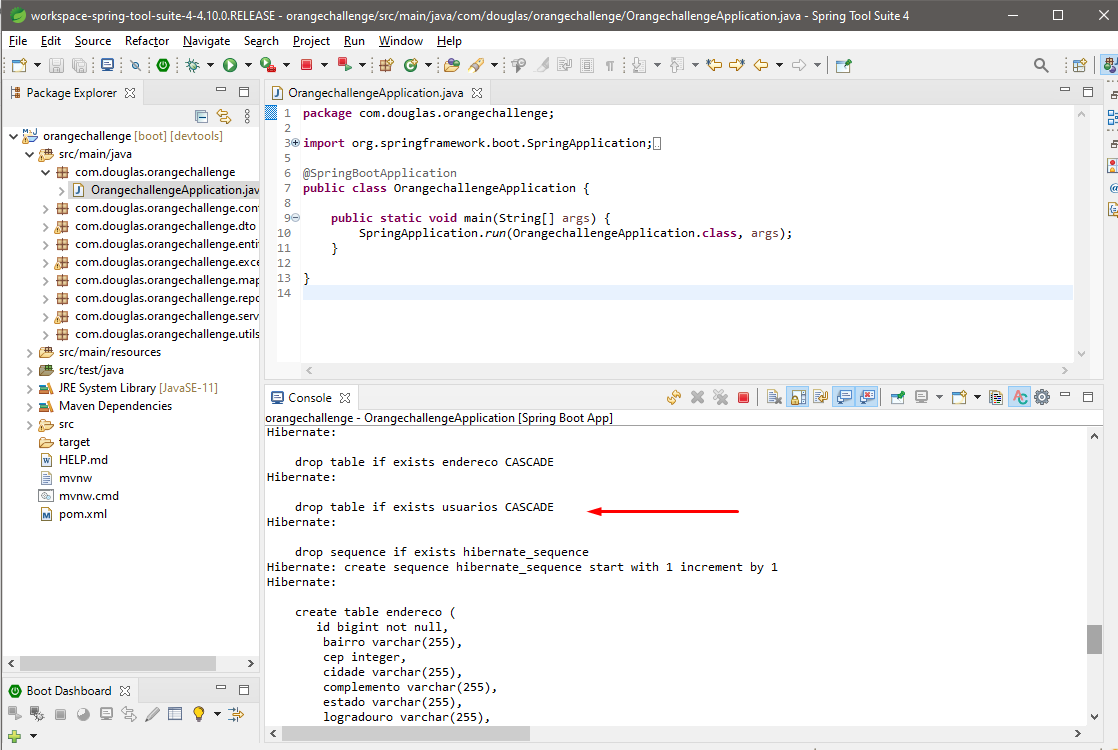


Para implementar os recursos do JPA, foram criadas as Interfaces no pacote repository que implementam o JpaRepository. Ele nos fornece principalmente as funções CRUD.  
  


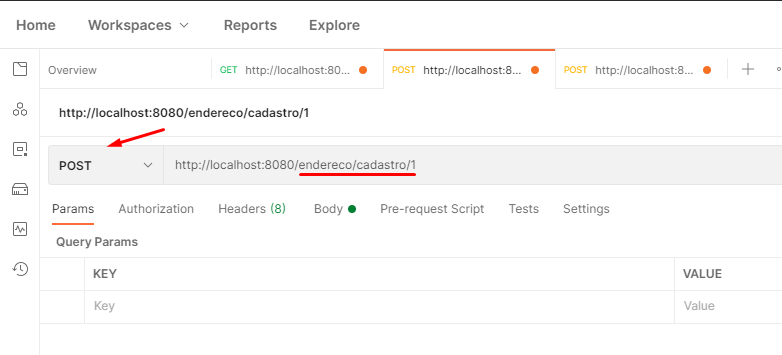
Pacote Service que recebe a Anotation @Service, além da injeção dos DTOs, ela será a principal responsável pelas operações que envolvem esta aplicação, a busca de um usuário.  
  
  
  
Pacote Util, utilizaremos para a criação da classe DataUtils responsável pela formatação de datas para o padrão brasileiro.

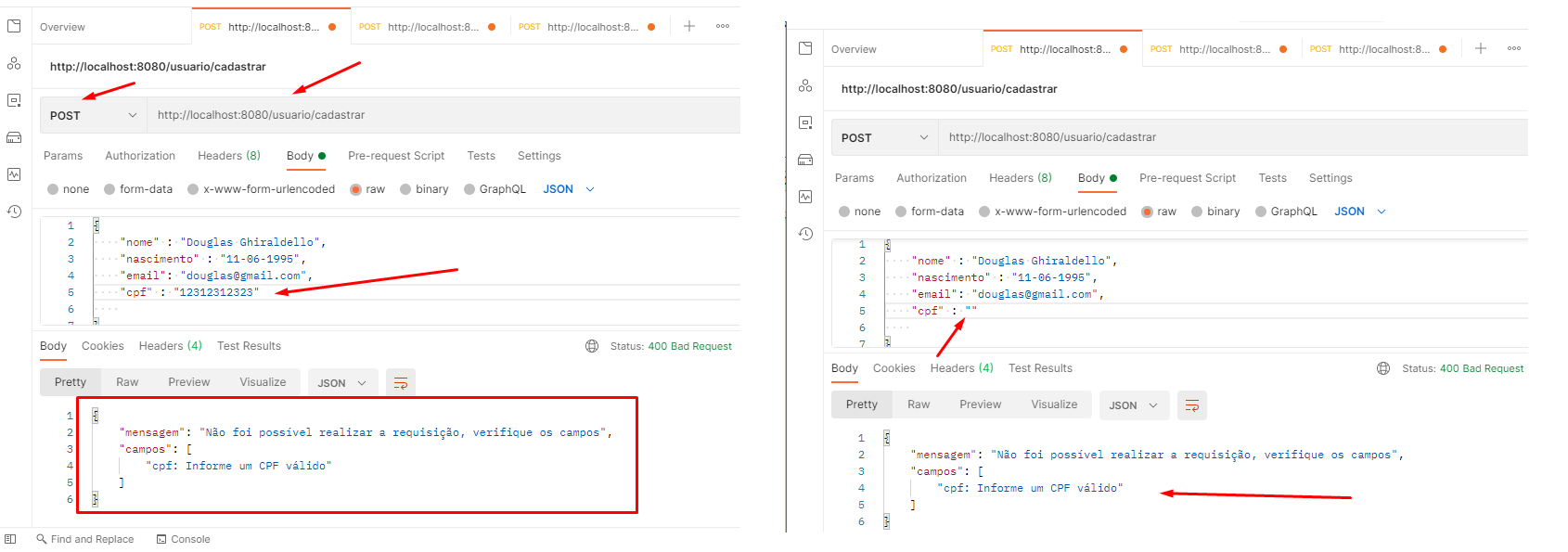
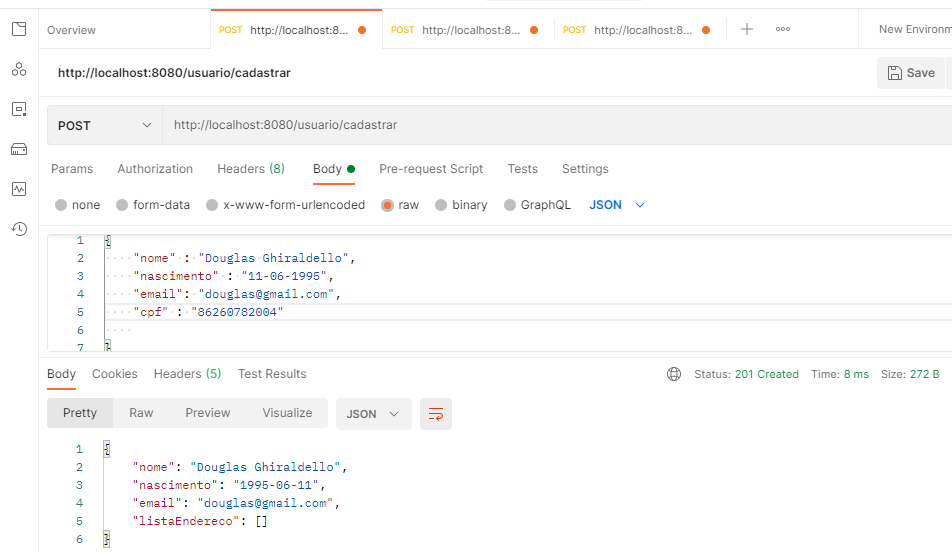


Agora que já implementamos a lógica, vamos colocar em funcionamento. Nosso banco já foi configurado, além de todas as dependências necessárias. Portanto ao reiniciar o projeto, devemos ver a criação das tabelas de maneira automática.

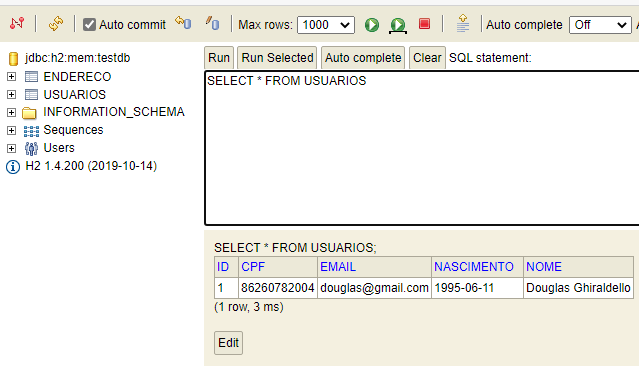


Em nossos testes utilizaremos a ferramenta Postman, que será a responsável por efetuar as requisições necessárias. Ao iniciarmos o Postman, utilizaremos o método POST e passar a url de acesso configurada na classe Controller, para cadastro de usuário e endereço

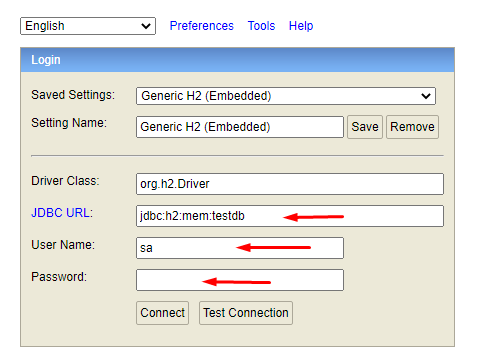
  
Dentro do postman vamos ao primeiro teste, informando dados inválidos ou vazio para um campo necessário. Conforme imagem abaixo, podemos notar que ao tentar efetuar este procedimento, teremos o código 400 como resposta, indicando que algo está errado.

Conforme validação implementada, vemos a mensagem que indica que o cpf informado está inválido:  
  
  
  
Segundo teste, vamos digitar corretamente todos os campos. Conforme imagem abaixo, é possível verificar que o código 201 é retornado indicando que os dados foram inseridos corretamente:  
  


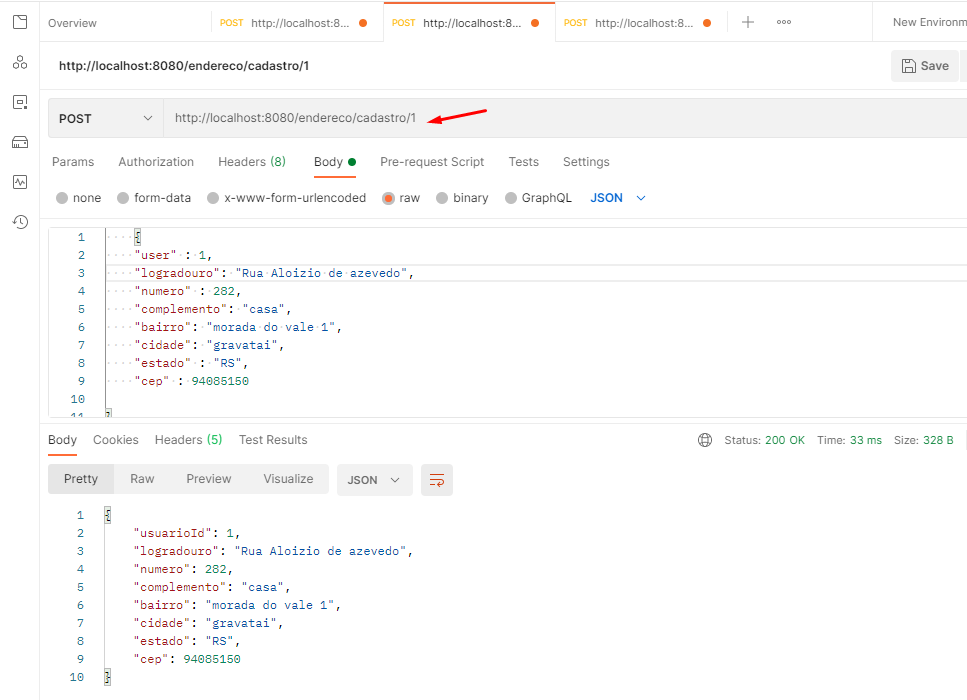
Podemos acessar o banco de dados e verificar se os dados foram inseridos de maneira correta.



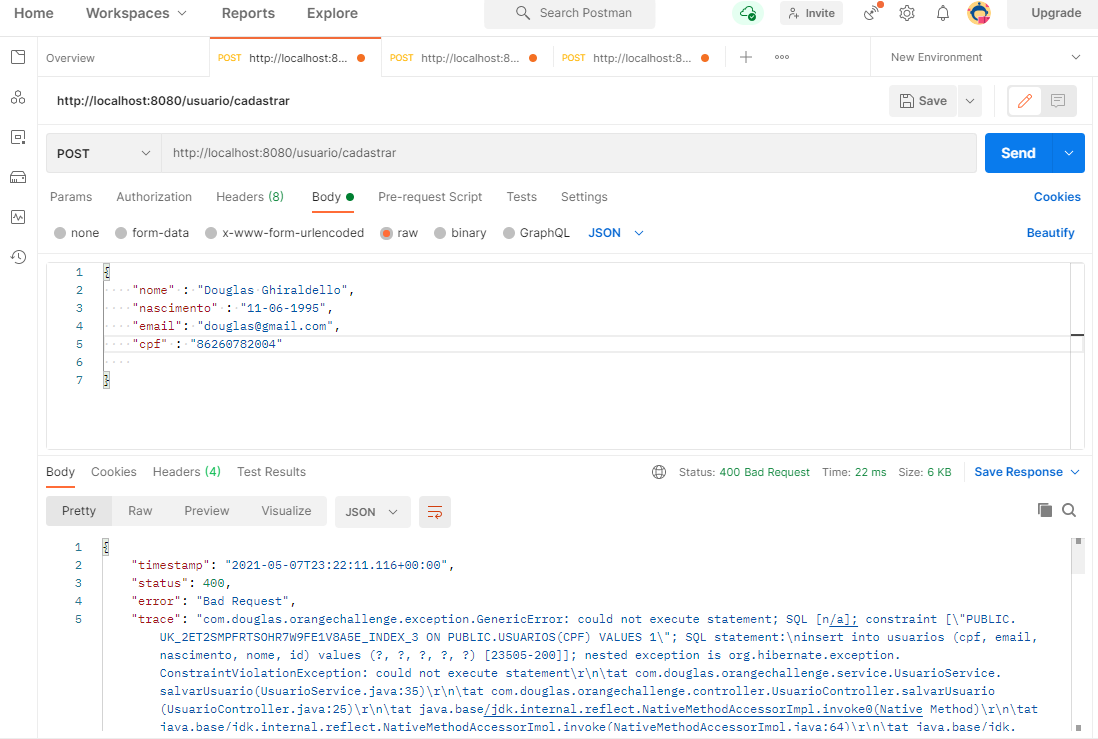
Obs: O banco de dados H2 está acessível pelo navegador através do link: <http://localhost:8080/h2-console/>. As credenciais de acesso são definidas no application.properties.



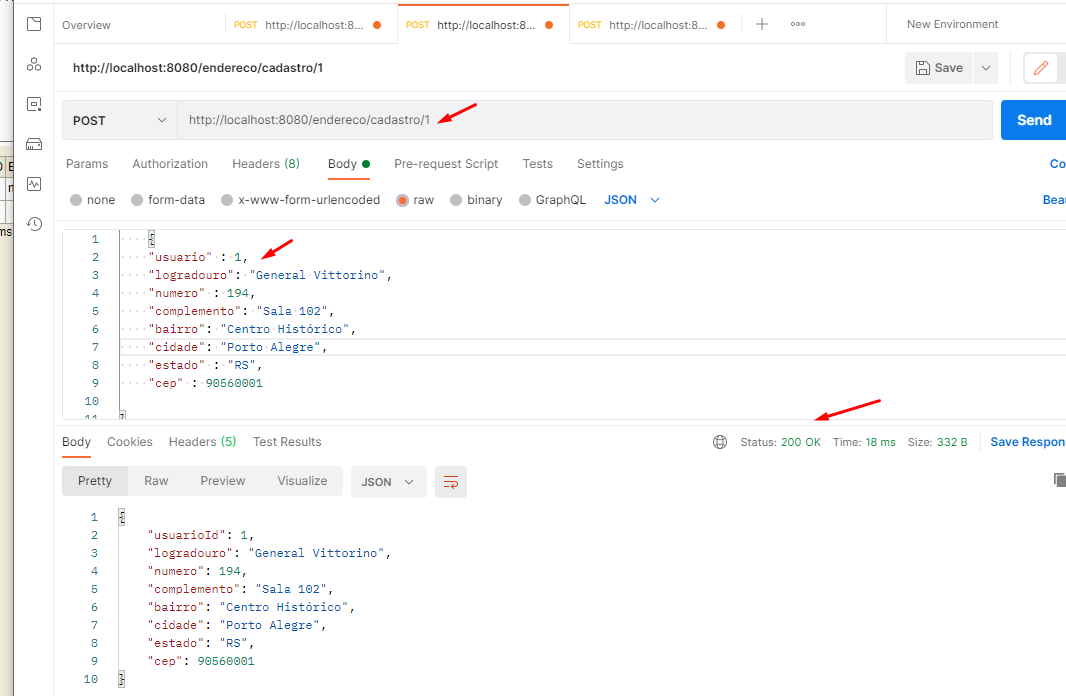
Seguindo com os testes, vamos cadastrar um endereço para um usuário específico. Enviada a requisição utilizando o método POST, temos o código 200 retornado, indicando que o endereço foi vinculado corretamente ao usuário id 1:



Testando o comportamento da API ao tentar incluir um usuário que já está cadastrado:



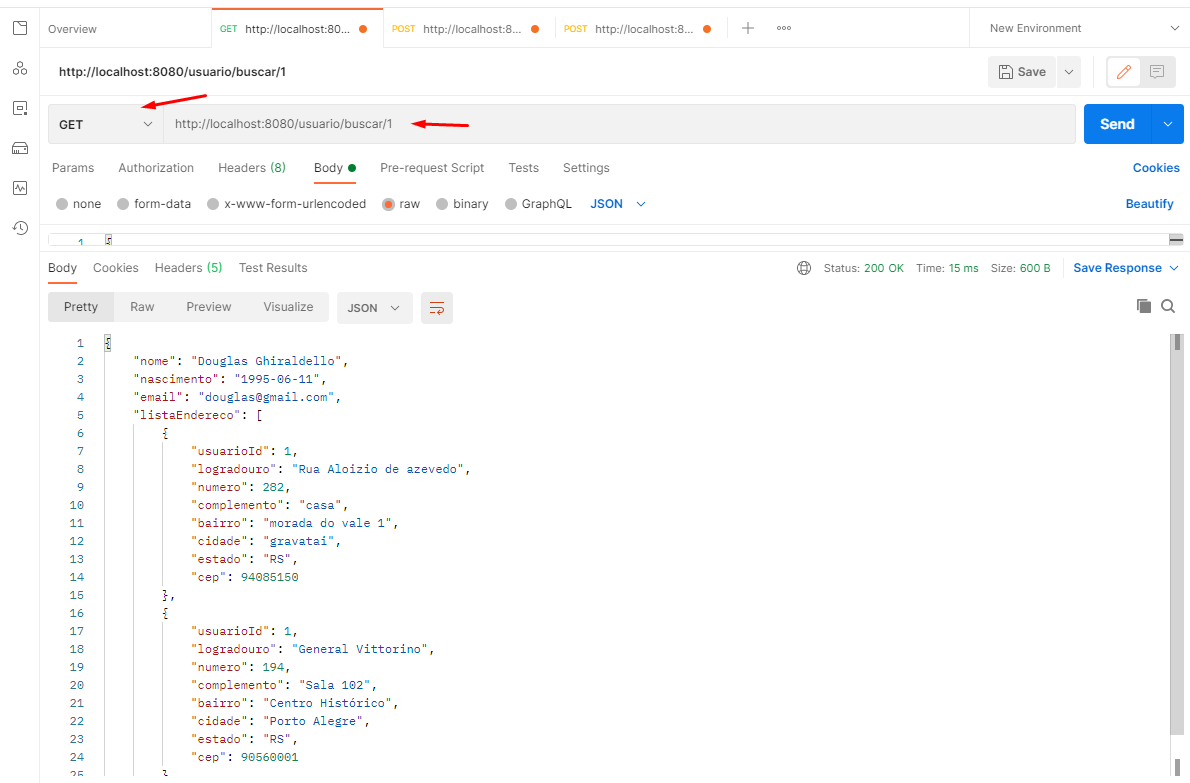
Cadastrando um segundo endereço e vinculando ao usuário 1:



Verificando no banco se os endereços foram atribuídos corretamente ao usuário:



Por fim, vamos efetuar uma busca com método GET, filtrando todos os endereços do usuário id 1:



Temos novamente como resultado o código 200, e todos os endereços atribuídos ao usuário foram listados.

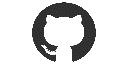
Por fim, se você chegou até aqui e seguiu corretamente os passos anteriormente citados, você conseguiu implementar as funcionalidades propostas.

Se você ficou com alguma dúvida ou quer dar uma espiadinha no código, poderá acessá-lo no Github através dos links anexados.

Nos vemos em breve!



[Esta Foto](https://commons.wikimedia.org/wiki/File:LinkedIn_logo_In-Black.svg) de Autor Desconhecido está licenciado em [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)



[Esta Foto](http://informationtransfereconomics.blogspot.com/2017/02/information-equilibrium-code.html) de Autor Desconhecido está licenciado em [CC BY](https://creativecommons.org/licenses/by/3.0/)