

Documentando a implementação do código Spring

Por onde começar? Pois é, o difícil é definir por onde, por isso começaremos pelo básico, definindo quem serão as nossas tecnologias de trabalho, e avançando aos poucos. Mas antes de tudo, a primeira etapa a ser feita, é criar um repositório do projeto da Zup no GitHub, para salvar o projeto e disponibiliza-lo em um link, ao final deste artigo.

Começo informando que utilizaremos o JAVA 8 e o H2 para o Banco de Dados (banco de dados em memória do JAVA mais voltado para o desenvolvimento local), e apesar da escolha, reconheço que o PostgreSQL também é muito utilizado por empresas para subir aplicações em um ambiente de produção. Além disso, para escrever o nosso código em Java, trabalharemos com uma IDE gratuita e muito utilizada, o Eclipse.

E mãos à obra, minha gente!

As imagens seguintes serão a indicação do passo a passo para a criação do projeto utilizando a tecnologia do Spring Inicializr, segue o endereço: <https://start.spring.io/>.

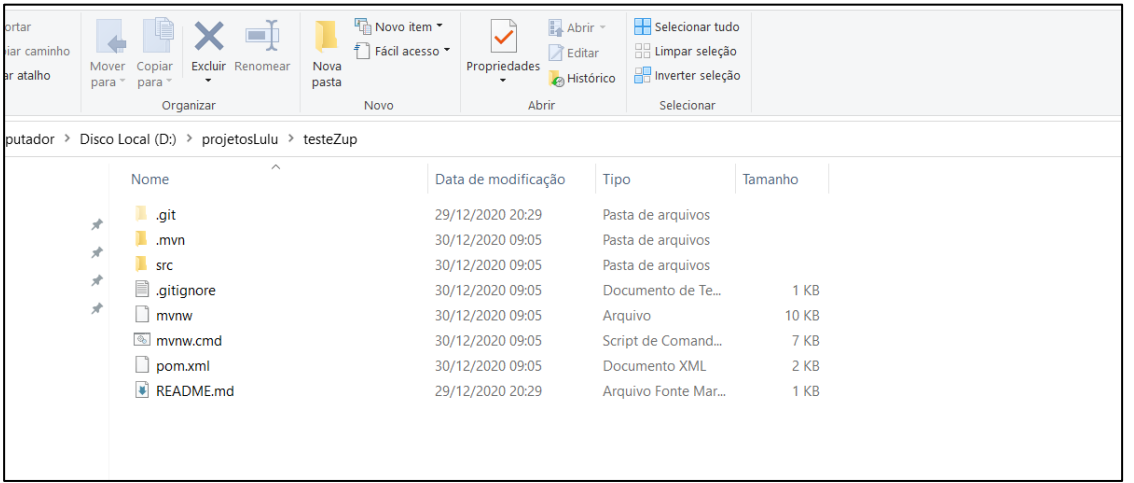
Tecnologias utilizadas:

- **Spring Web:** uma das funções do framework responsável por criar aplicações padrão Web;
- **H2 Database:** é a dependência responsável por utilizar um banco de dados em memória;
- **Spring Data JPA** é uma implementação do Spring para trabalhar com JPA.
- **Validation** validará o conteúdo a ser enviado através da API.

Começaremos pela configuração dos metadados básicos do projeto, e quais as suas dependências (**Web**, o **H2**, **Spring Data**, e o **Validation**), depois disso, é só baixar o Arquivo Zip com o projeto Spring Boot configurado e descompacta-lo onde você deseja, no meu caso, irei fazê-lo utilizando o endereço D: na minha máquina local.

The screenshot displays the Spring Initializr web interface. On the left, under 'Project', 'Maven Project' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', version '2.4.1' is selected. The 'Project Metadata' section includes fields for Group (br.com), Artifact (testZup), Name (testZup), Description (br.com.testeZup), and Package name (br.com.testZup). Under 'Packaging', 'Jar' is selected, and under 'Java', version '8' is selected. On the right, the 'Dependencies' section lists 'Spring Web' (WEB), 'H2 Database' (SQL), 'Spring Data JPA' (SQL), and 'Validation' (I/O). At the bottom, there are buttons for 'GENERATE' (CTRL + G), 'EXPLORE' (CTRL + SPACE), and 'SHARE...'. A settings icon is visible in the top right corner.

Note que o site Spring Inicializr criou automaticamente os arquivos src e o arquivo pom.xml, além de outros, mas estes não serão o foco deste projeto.



O próximo passo é *inicializar o projeto no terminal*, primeiramente localizamos o endereço da pasta, e na sequência digitamos o comando **mvn clean install**, e veja que o mvn irá instalar as dependências e verificar se há algo quebrando o código.

```

D:\projetos\lulu\testeZup>mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:testeZup >-----
[INFO] Building testeZup 0.0.1-SNAPSHOT
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ testeZup ---
[INFO] Deleting D:\projetos\lulu\testeZup\target
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ testeZup ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO] The encoding used to copy filtered properties files have not been set. This means that the same encoding will be used to copy filtered properties files as when copying other filtered resources. This might not be what you want! Run your build with --debug to see which files might be affected. Read more at https://maven.apache.org/plugins/maven-resources-plugin/examples/filtering-properties-files.html
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ testeZup ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to D:\projetos\lulu\testeZup\target\classes
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ testeZup ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory D:\projetos\lulu\testeZup\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ testeZup ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to D:\projetos\lulu\testeZup\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ testeZup ---
[INFO]
[INFO] ----- T E S T S -----
[INFO]
[INFO] Running com.example.testeZup.TesteZupApplicationTests
11:35:32.396 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDelegate from class [org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate]
11:35:32.412 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using constructor [public org.springframework.test.context.support.DefaultBootstrapContext(java.lang.Class,org.springframework.test.context.CacheAwareContextLoaderDelegate)]
11:35:32.453 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper for test class [com.example.testeZup.TesteZupApplicationTests] from class [org.springframework.boot.test.context.SpringBootTestContextBootstrapper]
11:35:32.468 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Neither @ContextConfiguration nor @ContextHierarchy found for test class [com.example.testeZup.TesteZupApplicationTests], using SpringBootContextLoader
11:35:32.478 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for test class [com.example.testeZup.TesteZupApplicationTests]: class path resource [com/example/testeZup/TesteZupApplicationTests-context.xml] does not exist
11:35:32.479 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for test class [com.example.testeZup.TesteZupApplicationTests]: class path resource [com/example/testeZup/TesteZupApplicationTests-context.groovy] does not exist
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ testeZup ---
[INFO] Installing D:\projetos\lulu\testeZup\target\testeZup-0.0.1-SNAPSHOT.jar to C:\Users\Ludmila\.m2\repository\com\example\testeZup\0.0.1-SNAPSHOT\testeZup-0.0.1-SNAPSHOT.jar
[INFO] Installing D:\projetos\lulu\testeZup\pom.xml to C:\Users\Ludmila\.m2\repository\com\example\testeZup\0.0.1-SNAPSHOT\testeZup-0.0.1-SNAPSHOT.pom
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 14.192 s
[INFO] Finished at: 2020-12-30T11:35:43-02:00
[INFO]
[INFO] -----
D:\projetos\lulu\testeZup>
```

Note que o Build terminou com **SUCCESS!!**

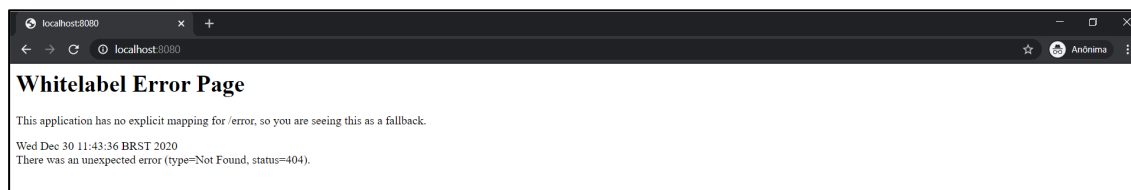
```
Prompt de Comando - mvn spring-boot:run
D:\projetos\lulu\testeZup>mvn spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----[ com.example:testeZup ]-----
[INFO]
[INFO] Building testeZup 0.0.1-SNAPSHOT
[INFO]
[INFO] -----[ jar ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.4.1:run (default-cli) > test-compile @ testeZup >>>
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ testeZup ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO] The encoding used to copy filtered properties files have not been set. This means that the same encoding will be used to copy filtered properties files as when copying other filtered resources. This might
not be what you want! Run your build with --debug to see which files might be affected. Read more at https://maven.apache.org/plugins/maven-resources-plugin/examples/filtering-properties-files.html
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ testeZup ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ testeZup ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory D:\projetos\lulu\testeZup\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ testeZup ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< spring-boot-maven-plugin:2.4.1:run (default-cli) < test-compile @ testeZup <<<
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.1:run (default-cli) @ testeZup ---
[INFO] Attaching agents: []
```

Agora sim, o nosso projeto subiu e está rodando na porta 8080.

```
Prompt de Comando
C:\Users\ludmila>mvn spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----[ com.example:testeZup ]-----
[INFO]
[INFO] Building testeZup 0.0.1-SNAPSHOT
[INFO]
[INFO] -----[ jar ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.4.1:run (default-cli) > test-compile @ testeZup >>>
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ testeZup ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO] The encoding used to copy filtered properties files have not been set. This means that the same encoding will be used to copy filtered properties files as when copying other filtered resources. This might
not be what you want! Run your build with --debug to see which files might be affected. Read more at https://maven.apache.org/plugins/maven-resources-plugin/examples/filtering-properties-files.html
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ testeZup ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ testeZup ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory D:\projetos\lulu\testeZup\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ testeZup ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< spring-boot-maven-plugin:2.4.1:run (default-cli) < test-compile @ testeZup <<<
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.1:run (default-cli) @ testeZup ---
[INFO] Attaching agents: []

2020-12-30 11:40:16.173 INFO 16300 --- [main] c.example.testeZup.TesteZupApplication : Starting TesteZupApplication using Java 1.8.0_211 on DESKTOP-0ALC4PG with PID 16300 (D:\projetos\lulu\testeZup\
target\classes started by Ludmila in D:\projetos\lulu\testeZup)
2020-12-30 11:40:16.176 INFO 16300 --- [main] c.example.testeZup.TesteZupApplication : No active profile set, falling back to default profiles: default
2020-12-30 11:40:16.545 INFO 16300 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2020-12-30 11:40:16.554 INFO 16300 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 3 ms. Found 0 JPA repository interfaces.
2020-12-30 11:40:17.102 INFO 16300 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-12-30 11:40:17.109 INFO 16300 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-12-30 11:40:17.110 INFO 16300 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.41]
2020-12-30 11:40:17.170 INFO 16300 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2020-12-30 11:40:17.178 INFO 16300 --- [main] w.s.c.ServletWebServerApplicationContext : Root webApplicationContext initialization completed in 965 ms
2020-12-30 11:40:17.278 INFO 16300 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2020-12-30 11:40:17.376 INFO 16300 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2020-12-30 11:40:17.408 INFO 16300 --- [main] o.hibernate.jpa.internal.util.LogHelper : HH0000204: Processing PersistenceUnitInfo [name: default]
2020-12-30 11:40:17.438 INFO 16300 --- [main] org.hibernate.Version : HH0000412: Hibernate ORM core version 5.4.25.Final
2020-12-30 11:40:17.544 INFO 16300 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations (5.1.2.Final)
2020-12-30 11:40:17.614 INFO 16300 --- [main] org.hibernate.dialect.Dialect : HH0000400: Using dialect: org.hibernate.dialect.H2Dialect
2020-12-30 11:40:17.707 INFO 16300 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2020-12-30 11:40:17.777 INFO 16300 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2020-12-30 11:40:17.810 WARN 16300 --- [main] jp.haseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view render
ing. Explicitly configure spring.jpa.open-in-view to disable this warning
2020-12-30 11:40:17.881 INFO 16300 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-12-30 11:40:18.017 INFO 16300 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2020-12-30 11:40:18.024 INFO 16300 --- [main] c.example.testeZup.TesteZupApplication : Started TesteZupApplication in 2.094 seconds (DM running for 2.378)
2020-12-30 11:42:31.670 INFO 16300 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-12-30 11:42:31.678 INFO 16300 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2020-12-30 11:42:31.679 INFO 16300 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2020-12-30 12:04:16.672 INFO 16300 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting down ExecutorService 'applicationTaskExecutor'
2020-12-30 12:04:16.672 INFO 16300 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2020-12-30 12:04:16.672 INFO 16300 --- [extShutdownHook] [SchemaDropperImpl$DelayedDropActionImpl] : HH0000477: Starting delayed evictData of schema as part of SessionFactory shut-down'
2020-12-30 12:04:16.674 INFO 16300 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2020-12-30 12:04:16.675 INFO 16300 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 24:02 min
[INFO] Finished at: 2020-12-30T12:04:16-02:00
[INFO]
Desaja finalizar o arquivo em lotes (S/N)? s
D:\projetos\lulu\testeZup>
```

Para confirmar, utilizamos o navegador na URL <http://localhost:8080/>. A indicação do conteúdo *Whitelabel Error Page*, irá desaparecer assim que o projeto for concluído.

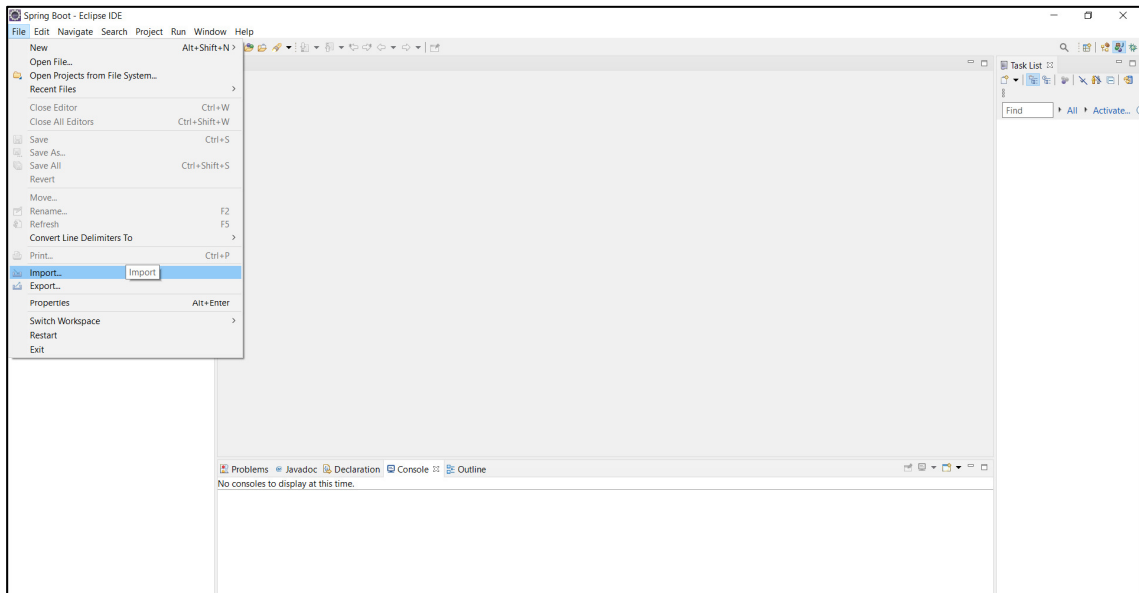


Conteúdo gerado pelo Spring Inicializr:

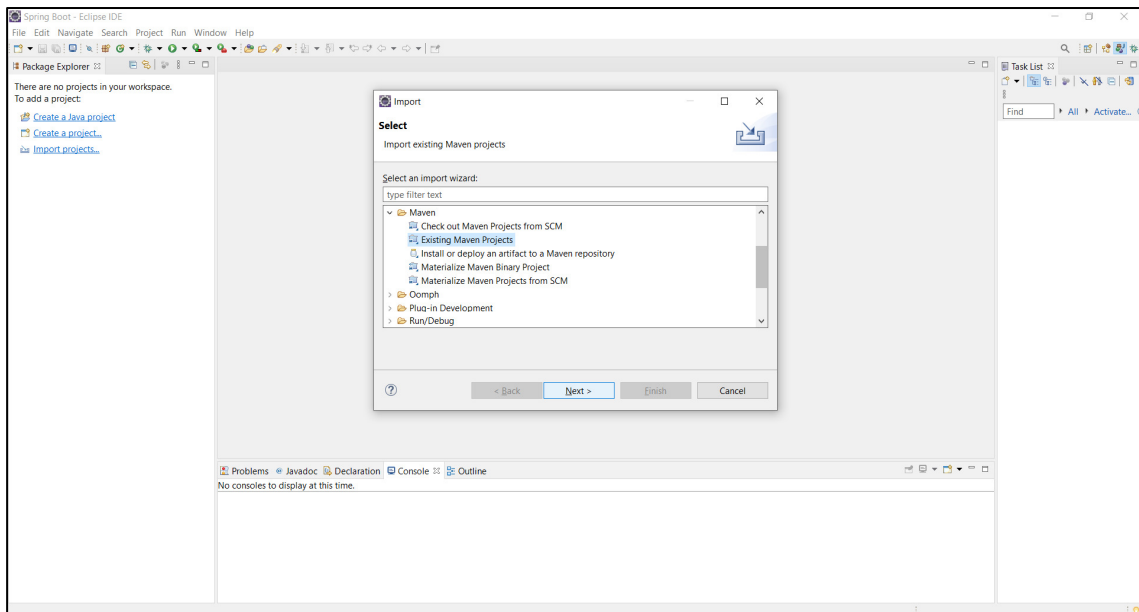
- **SRC:** na pasta src é onde serão gerados os arquivos do projeto;
- **POM.XML:** é onde estão os endereços de todas as dependências e plugins que serão utilizados no projeto, além de definirmos outras informações como versão do JAVA, descrição do projeto, nome do projeto, etc.

Etapas do processo

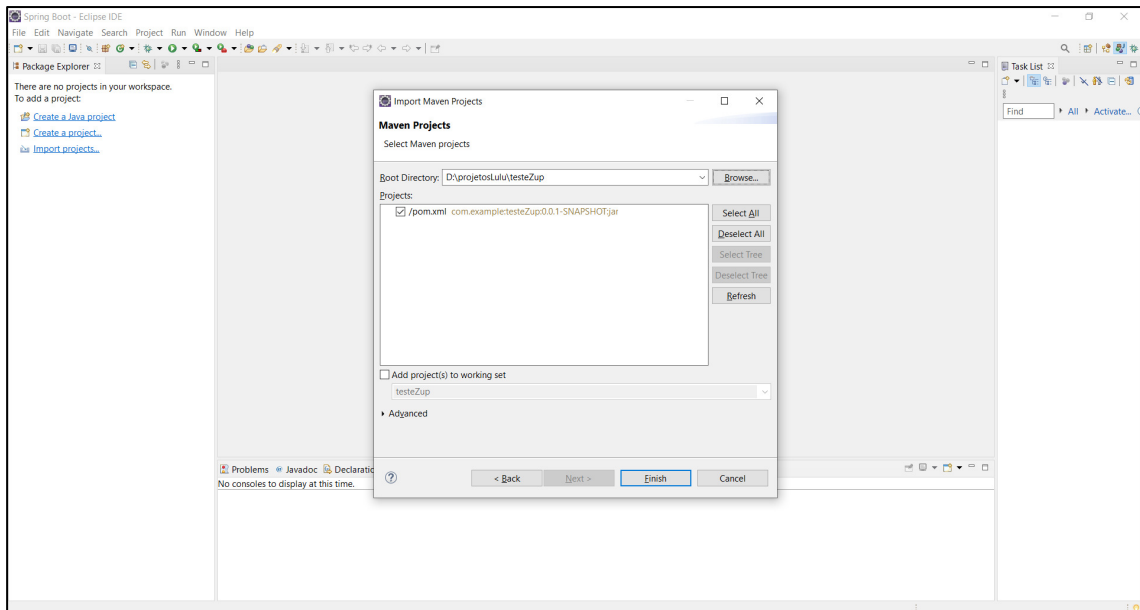
Depois que iniciar o Eclipse, vamos **importar** o projeto pré-definido pela ferramenta Spring Inicializr,



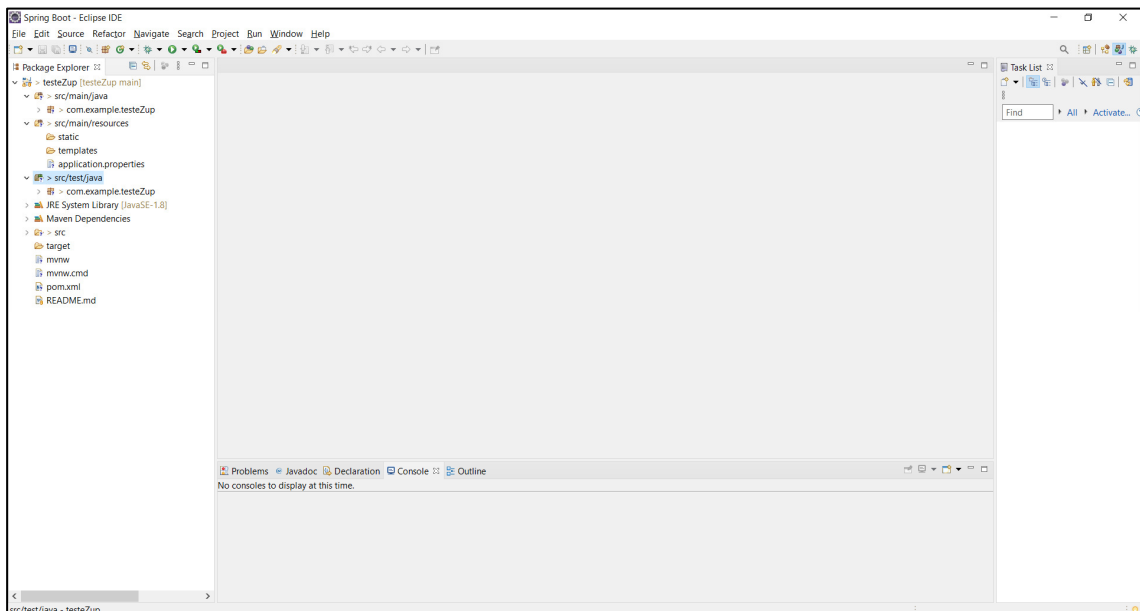
Depois selecione a pasta Maven/Existing Maven Projects, e clique em Next;



Agora clicamos em Browse e selecionamos o endereço onde o nosso projeto inicial foi salvo, e depois clicamos em Finish.



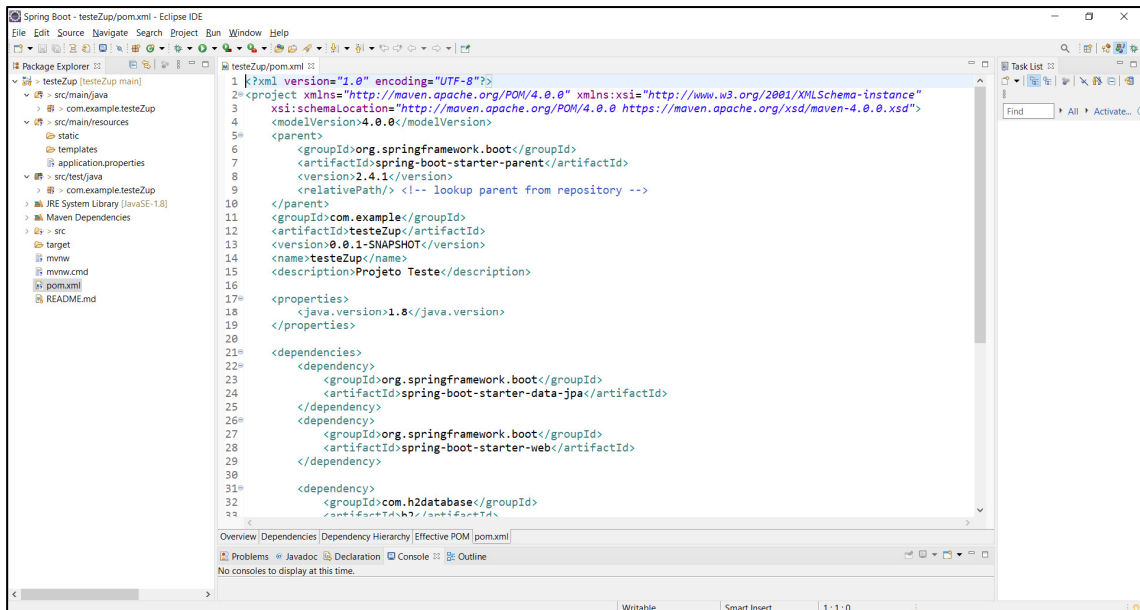
Observe que o projeto será aberto na pasta Package Explorer, na lateral esquerda da página;



Navegando entre as pastas, notamos que vários arquivos padrão foram criados quando utilizamos o Spring Inicializr;

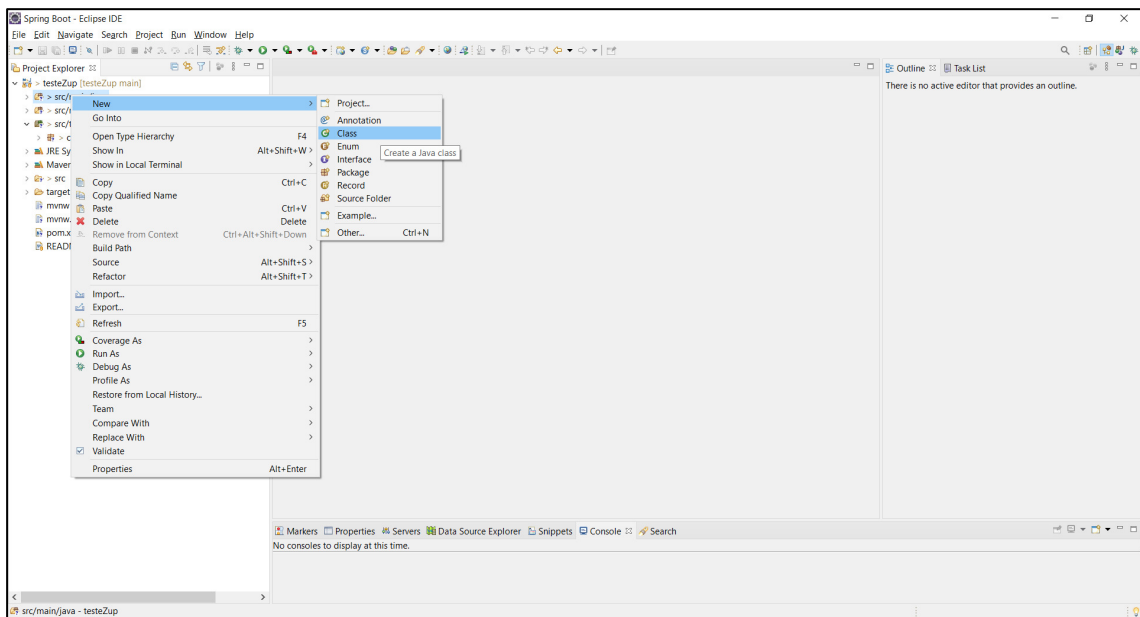
Abrindo o arquivo POM.XML a seguir, encontramos os dados da identidade do nosso projeto, além das dependências e plugins que serão utilizados.

É muito importante lembrar que havendo necessidade para a inclusão de mais dependências ou plugins, é aqui que acrescentaremos as mesmas. Ora dentro da guia: `<dependencies></dependencies>` ora dentro da guia `<plugins></plugins>`.

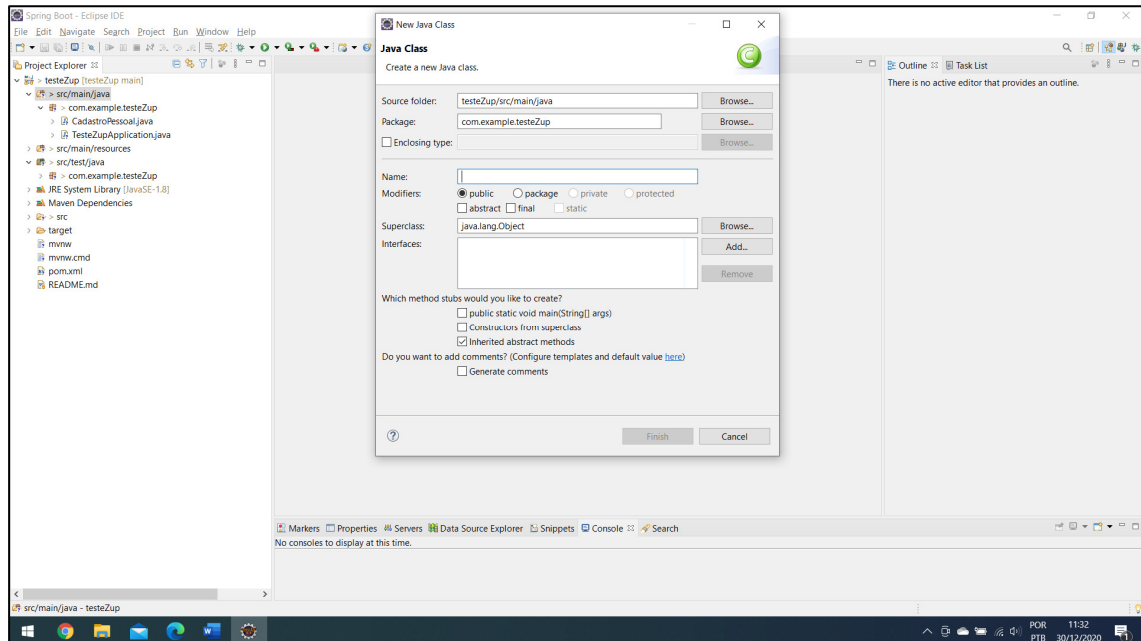


Antes de mais nada, informarei que utilizaremos a versão do *Eclipse IDE for JAVA for Enterprise JAVA Developers 2020-06*. Agora sim, vamos iniciar o nosso código.

Para criar uma classe, utilizaremos o caminho a seguir:



Selecione no **Browse** onde queremos salvar a nossa classe, e depois o nome que escolhemos.



A nova classe aparecerá no Package Explorer, ou Project Explorer (Depende de qual workspace estará sendo utilizada no momento);

Criando a Classe CadastroPessoa

```
package br.com.testeZup.modelo;
```

```
import java.sql.Date;
```

```
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;
```

```
@Entity
```

```
public class CadastroPessoa {
```

```
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    private String nome;
```

```
    private String email;
```

```
    private String cpf;
```

```
    private Date dataDeNascimento;
```

```
    public CadastroPessoa() {  
    }
```

```
    public CadastroPessoa(Long id, String nome, String email, String cpf,  
Date dataDeNascimento) {  
        this.id = id;  
        this.nome = nome;  
        this.email = email;
```

```

        this.cpf = cpf;
        this.dataDeNascimento = dataDeNascimento;
    }

    public CadastroPessoa(String nome, String email, String cpf, Date
dataDeNascimento) {
        this.nome = nome;
        this.email = email;
        this.cpf = cpf;
        this.dataDeNascimento = dataDeNascimento;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    public Date getDataDeNascimento() {
        return dataDeNascimento;
    }

    public void setDataDeNascimento(Date dataDeNascimento) {
        this.dataDeNascimento = dataDeNascimento;
    }
}
}
}

```


Explicando um pouquinho sobre o código escrito...

Primeiramente comecei escrevendo todos os atributos da **classe CadastroPessoa** (id, nome, e-mail, cpf e dataDeNascimento), em seguida criei o **construtor da classe** (para a construção do objeto quando for chamado), e por último, criei os **getters e setters**, já que no caso do exemplo os atributos serão privados.

- `@Entity` – é a entidade responsável pelo mapeamento entre a tabela do banco de dados e a minha classe(CadastroPessoa);
- `@Id` - é o identificador da tabela;
- `@GeneratedValue` – geração automática de identificadores.

Criando a Classe CadastroPessoaController

```
package br.com.testeZup.controller;

import java.util.List;
import java.util.Optional;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.testeZup.controller.form.CadastroPessoaForm;
import br.com.testeZup.modelo.CadastroPessoa;
import br.com.testeZup.repository.CadastroPessoaRepository;

@RestController
@RequestMapping("/cadastro")
public class CadastroPessoaController{

    @Autowired
    CadastroPessoaRepository cadastroRepository;

    @PostMapping
    public ResponseEntity abreConta(@RequestBody @Valid CadastroPessoaForm
form) {

        CadastroPessoa cadastroConversao = form.converter();
        try {
            Optional<CadastroPessoa> temCpf =
cadastroRepository.findByCpf(form.getCpf());
            Optional<CadastroPessoa> temEmail =
cadastroRepository.findByEmail(form.getEmail());
```

```

        if(temCpf.isPresent()) {
            return ResponseEntity.ok("Cpf já existe");
        }

        if(temEmail.isPresent()) {
            return ResponseEntity.ok("Email já existe");
        }

        cadastroRepository.save(cadastroConversao);

        return ResponseEntity.ok(cadastroConversao);
    } catch (Exception e) {

        return ResponseEntity
            .status(HttpStatus.FORBIDDEN)
            .body("Error Message " + e.getMessage());
    }
}

```

Explicando um pouquinho sobre o código escrito...

Na classe Controller começo pela **criação da classe CadastroPessoaController**, contaremos com as seguintes etapas na nossa classe:

Começaremos injetando a classe **repositório CadastroPessoaRepository**;

Em seguida utilizaremos o método PostMapping, que falaremos mais em *anotações utilizadas* e por último criaremos um Try cath para tratar se o cpf e o e-mail já existem.

Anotações e referências utilizadas

- **@RestController** = notifica o SpringBoot que a classe Controller receberá requisições Rest (Post, Get, Put, Delete, Path, etc);
- **@RequestMapping** = é o mapeamento da rota onde a controller receberá as requisições ("/cadastro");
- **@Autowired** = responsável por injetar a dependência do CadastroRepository;
- **@PostMapping** = é a anotação que indica para a Controller que receberemos requisições do tipo Post.
- **ResponseEntity** = envelope a resposta em forma de objeto;
- **@RequestBody** = captura a informação do corpo do formulário de CadastroPessoaForm;
- **Try cath** = o nosso tratamento contará com a seguinte lógica, a ferramenta findByCpf e findByEmail começará verificando na tabela do banco de dados se a informação já existe, comparando-a com o formulário da classe CadastroPessoaForm, e caso já exista, o próximo passo será retornar uma mensagem cadastrada no if, e se houver qualquer erro além daqueles tratados na classe ErroDeValidação, exemplo; falha na conexão do banco de dados, ele pegará essa informação no Cath.

Criando a ClassePessoaForm

```
package br.com.testeZup.controller.form;

import java.sql.Date;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.Length;

import br.com.testeZup.modelo.CadastroPessoa;

public class CadastroPessoaForm {

    @NotNull @NotEmpty
    private String nome;
    @NotNull @NotEmpty @Email
    private String email;
    @NotNull @NotEmpty @Length(min = 11)
    private String cpf;
    @NotNull
    private Date dataDeNascimento;

    public CadastroPessoa converter() {
        return new CadastroPessoa(this.nome, this.email, this.cpf,
this.dataDeNascimento);
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getCpf() {
        return cpf;
    }
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
    public Date getDataDeNascimento() {
        return dataDeNascimento;
    }
    public void setDataDeNascimento(Date dataDeNascimento) {
        this.dataDeNascimento = dataDeNascimento;
    }
}
```

Explicando um pouquinho sobre o código escrito...

O que fizemos nesta classe foi escrever os argumentos do projeto, indicar o **tipo de validação** (optou-se pelo `@NotNull @NotEmpty @Length @Email`) e acrescentar os getters e setters, já que são argumentos privados, além do método converter, que transfere os dados de CadastroPessoa para que após ser preenchido persista no Banco de Dados;

Criando a Classe CadastroPessoaRepository

```
package br.com.testeZup.repository;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import br.com.testeZup.modelo.CadastroPessoa;

public interface CadastroPessoaRepository extends
    JpaRepository<CadastroPessoa, Long>{

    Optional<CadastroPessoa> findByCpf(String cpf);

    Optional<CadastroPessoa> findByEmail(String email);

}
```

Explicando um pouquinho sobre o código escrito...

Criaremos uma interface chamada CadastroPessoaRepository que herdará da JpaRepository, todas as funções da Jpa que precisaremos para salvar, deletar, atualizar, listar, entre outras do banco de dados. E porque uma interface e não uma classe? Porque esta última será um contrato entre o nosso código e a JPA.

Implementações utilizadas da convenção do SpringData;

- `findByCpf (String cpf)` – procura na tabela do banco de dados, CadastroPessoa, o campo cpf, e retorna o registro igual;
- `findByEmail (String email)` – procura na tabela do banco de dados, CadastroPessoa, o campo Email, e retorna o registro igual;

O nosso TesteZupAplicattion terá a anotação `@SpringBootApplication`, que será a classe que utilizaremos para rodar a nossa aplicação dentro do main, onde há um Tomcat embutido e roda para subir a nossa aplicação, logo não precisamos adicionar o servidor separado.

Configurando o arquivo application.properties

É um arquivo onde constará as configurações do nosso projeto Spring, adicionaremos algumas linhas para configurar o H2, o JPA e o DataSource.

```
# data source
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:testeZup
spring.datasource.username=sa
spring.datasource.password=

# jpa
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

# h2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Sobre a Package Validação

O pacote **br.com.testeZup.config.validacao** será responsável por capturar as nossas exceções e trata-las, confesso que ainda não dominei o que acontece na captura da exceção, mesmo assim, considereei utilizar a implementação, inclusive, retirei a implementação do código do curso Spring Boot da Alura realizado recentemente.

Observação

Todos os testes foram realizados no **Postman**.

Link do projeto

<https://github.com/ludmilaaraujo/testeZup>