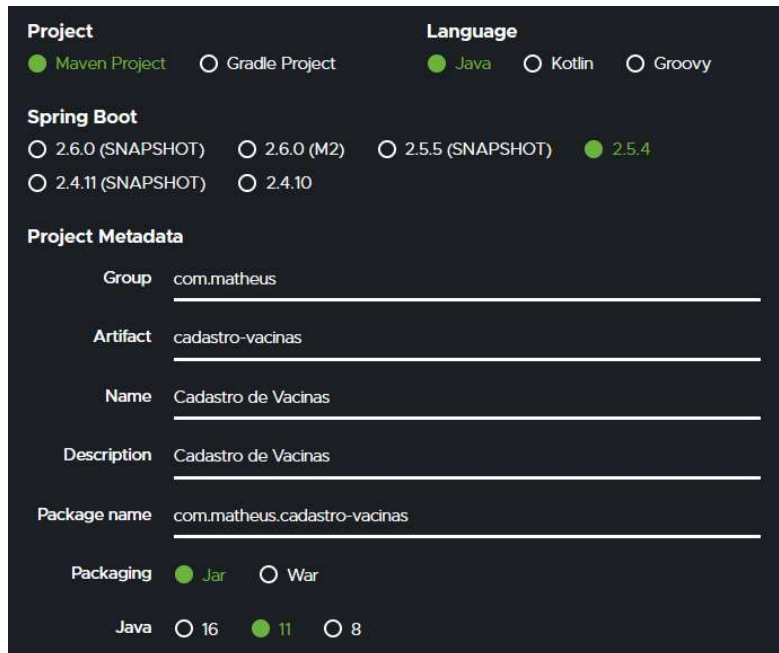


Cadastro de vacinas -> Java, Spring + Hibernate

Matheus Magalhães de Paula Paiva – 19/09/2021

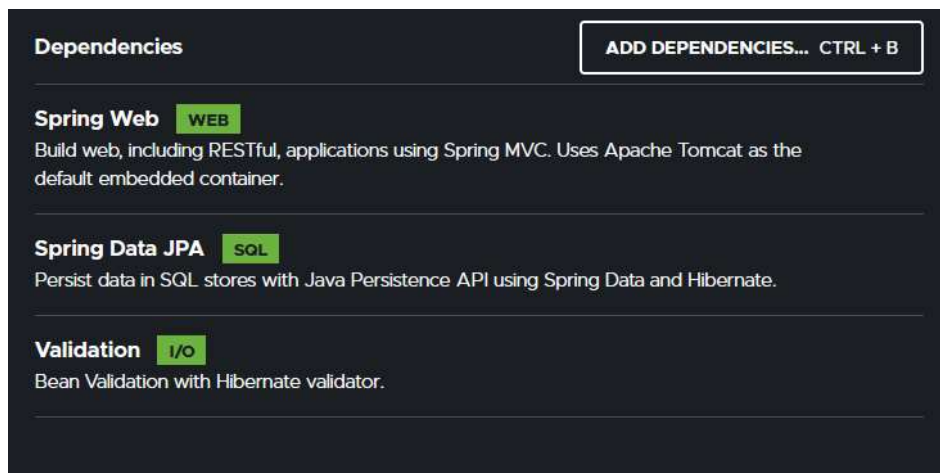
Inicialmente, foi utilizado o Springinitializr, acessando o site <https://start.spring.io/> para gerar o arquivo .zip configurado e com as dependências. A tela foi configurada como se mostra abaixo:



The image shows the Spring Initializr configuration interface. It is divided into several sections:

- Project:** ☒ Maven Project, ☐ Gradle Project
- Language:** ☒ Java, ☐ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 2.6.0 (SNAPSHOT), ☐ 2.6.0 (M2), ☐ 2.5.5 (SNAPSHOT), ☒ 2.5.4, ☐ 2.4.11 (SNAPSHOT), ☐ 2.4.10
- Project Metadata:**
 - Group:** com.matheus
 - Artifact:** cadastro-vacinas
 - Name:** Cadastro de Vacinas
 - Description:** Cadastro de Vacinas
 - Package name:** com.matheus.cadastro-vacinas
- Packaging:** ☒ Jar, ☐ War
- Java:** ☐ 16, ☒ 11, ☐ 8

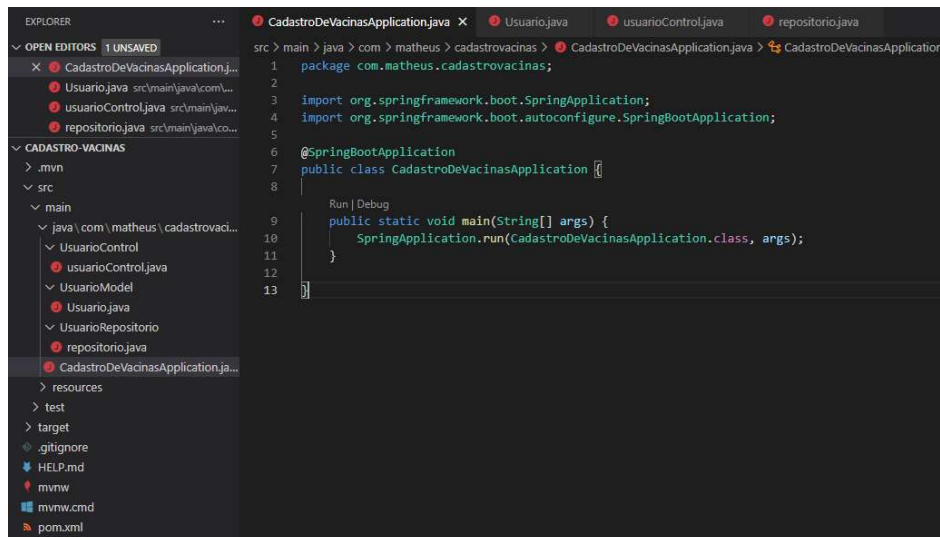
Para as dependências, foram adicionadas o Spring Web, o Spring Data JPA e o Validation, conforme imagem a seguir:



The image shows the Dependencies section of the Spring Initializr. It includes a button "ADD DEPENDENCIES... CTRL + B" and three selected dependencies:

- Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- Validation** (I/O): Bean Validation with Hibernate validator.

Após a geração do arquivo .zip, este foi adicionado ao Visual Studio Code para poder trabalhar com o desafio proposto. No arquivo principal, foi feita uma anotação marcando que o arquivo trabalhado é uma aplicação de Springboot, utilizando `@SpringBootApplication` e dentro da função principal, foi inicializada a aplicação, através da chamada `SpringApplication.run(CadastroDeVacinasApplication.class, args);`. A seguir se encontra a imagem mostrando a função principal do código:



Para a criação do cadastro de usuários, foi criada uma classe chamada Usuário e nela colocada todas as variáveis que seriam utilizadas para adicionar os dados, sendo eles: Nome, CPF, Email e data de nascimento. Houve a anotação @Entity sobre o início da classe, com o nome da tabela de usuário. Na declaração das variáveis, cada uma foi anotada como Column, para o banco de dados, onde foram configuradas para que todos os campos sejam preenchidos, com um limite de tamanho e as entradas de CPF e Email marcadas como unique.

```
@Entity(name = "Usuario")
public class Usuario {

    @Column(nullable = false, length = 45)
    private String nome;

    @Column(nullable = false, length = 14, unique = true)
    private String cpf;

    @Column(nullable = false, length = 14, unique = true)
    private String email;

    @Column(nullable = false, length = 10)
    private String dataNasc;
```

Também foram criados métodos Setters, para o preenchimento das variáveis, declaradas como private.

Para a continuação do código, foram criadas uma interface repositório, configurando o banco de dados com o JPA do Springboot e mais uma classe de controle de usuário, fazendo a inserção dos valores no BD.

```
@Repository
public interface repositorio extends JpaRepository <Usuario, Long> {

}
```

Como o Spring Data JPA já fornece os principais métodos, não foi necessário criar novos métodos dentro da interface, foi utilizada em uma chamada dentro da classe de controle do usuário, conforme abaixo:

```
@RestController
@RequestMapping("/usuario")
public class usuarioControl {

    @Autowired
    private repositorio usuarioRepositorio;

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Usuario adicionar (@RequestBody Usuario usuario)
    {
        return usuarioRepositorio.save(usuario);
    }
}
```

A propriedade repositório usuarioRepositorio foi anotada com o @AutoWired para que seja colocada uma instância de repositório em usuarioRepositorio. Para finalizar o código elaborado, na criação do método, é utilizado a anotação @RequestBody, para que o JSON utilizado na inserção do dado seja convertido para o tipo usuário. A anotação @ResponseStatuts foi colocada com a finalidade de mostrar o status 201, caso seja possível adicionar o cadastro da pessoa no banco de dados. Para fazer o mapeamento dos dados no método, foi utilizado o @PostMapping.