



Spring

Criação de Projeto no Spring Initializr

- Acessar o Spring Initializr

Spring Initializr
Initializr generates spring boot project with just what you need to start quickly!
<https://start.spring.io/>



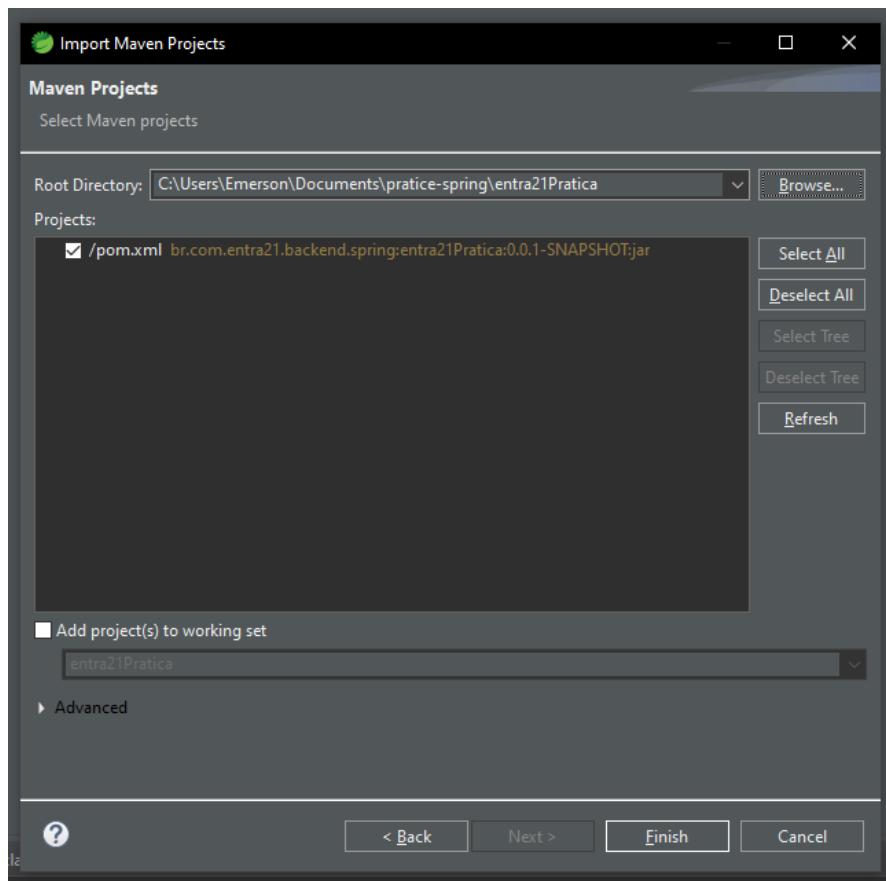
- Configurar e Gerar o Projeto



Project	Language	Dependencies		
<input checked="" type="radio"/> Maven Project	<input checked="" type="radio"/> Java	<input type="radio"/> Kotlin		
<input type="radio"/> Gradle Project	<input type="radio"/> Groovy	ADD ... CTRL + B		
<hr/>				
Spring Boot				
<input type="radio"/> 3.0.0 (SNAPSHOT)	<input checked="" type="radio"/> 3.0.0 (M4)			
<input type="radio"/> 2.7.4 (SNAPSHOT)	<input checked="" type="radio"/> 2.7.3			
<input type="radio"/> 2.6.12 (SNAPSHOT)	<input checked="" type="radio"/> 2.6.11			
<hr/>				
Project Metadata				
Group	br.com.entra21.backend.spring			
Artifact	entra21			
Name	entra21			
Description	Demo project for Spring Boot			
Package name	br.com.entra21.backend.spring.projeto			
Packaging	<input checked="" type="radio"/> Jar	<input type="radio"/> War		
Java	<input type="radio"/> 18	<input checked="" type="radio"/> 17	<input type="radio"/> 11	<input type="radio"/> 8

Importar o Projeto no SpringTools

- Descompactar no workspace e importar no SpringTools
 - Importar como um projeto maven existente
 - Project explorer ou Package explorer > right click > import... > Maven > Localizar o projeto no computador



Preparar classe Application

- No pacote principal localize a classe Application que contém o método main
 - Nessa classe deve conter antes do nome a anotação @SpringBootApplication, isso define as configurações adicionais para ser uma execução SpringBoot
 - Implemente a interface CommandLineRunner e o método run
 - Pode deixar vazio nesse momento, quando houver necessidade de implementar logicas adicionais assim que o servidor é iniciado, lembre-se que existe esse evento.

```
package br.com.entra21.backend.spring.entra21Pratica;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Entra21PraticaApplication implements CommandLineRunner {

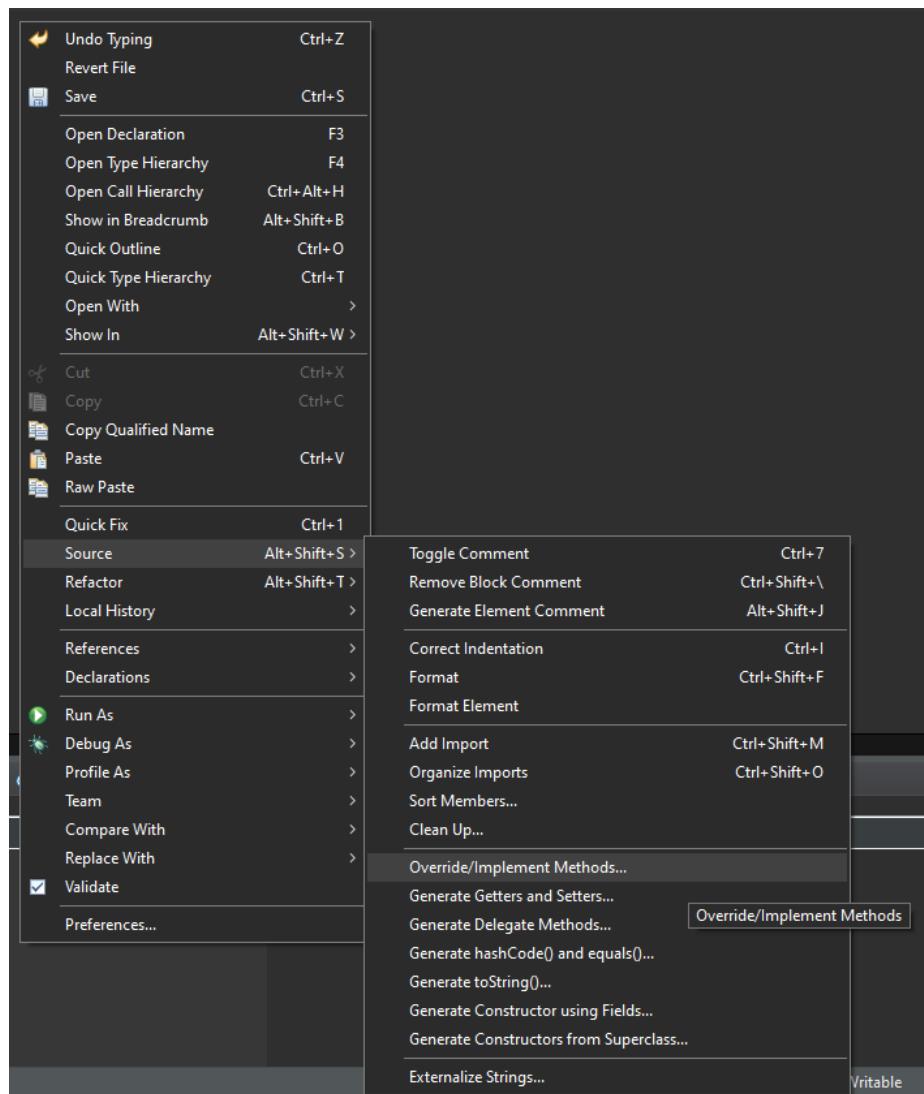
    public static void main(String[] args) {
        SpringApplication.run(Entra21PraticaApplication.class, args);
    }

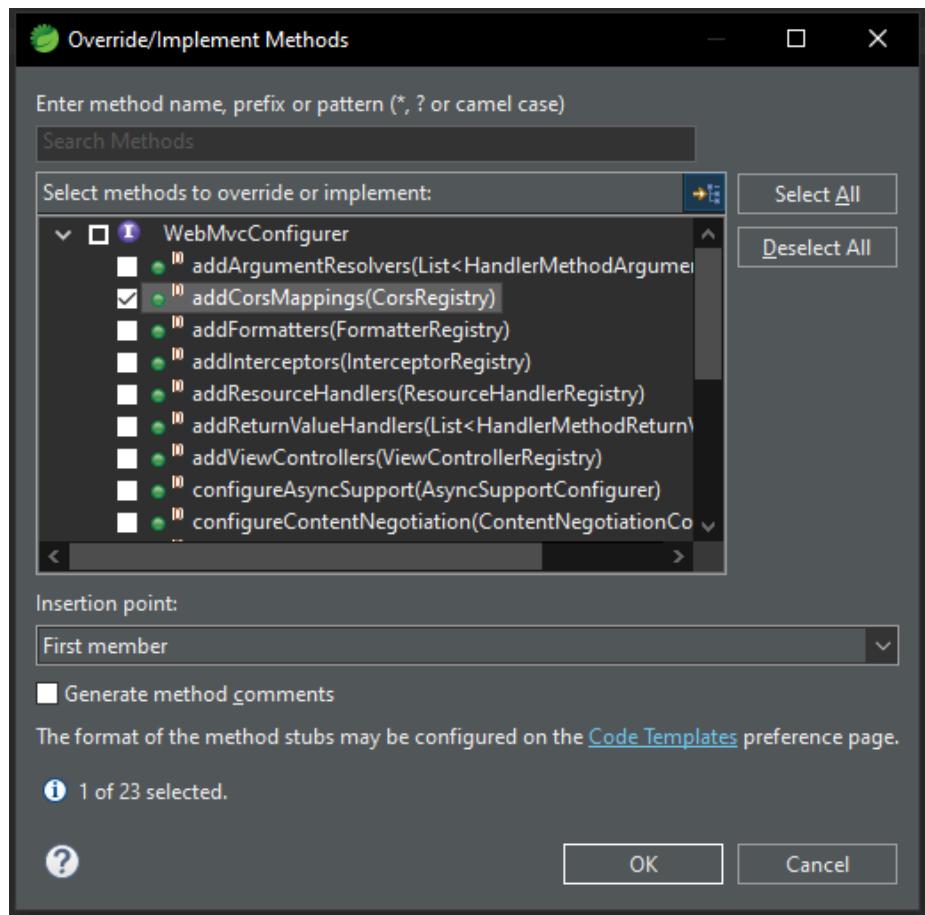
    @Override
    public void run(String... args) throws Exception {
        // TODO Auto-generated method stub
    }
}
```

```
 }  
 }
```

Criar Classe WebConfig

- Ainda no pacote principal crie uma classe com nome WebConfig
 - Essa classe tem o objetivo de remover as restrições cors de segurança que exigem origens e métodos declaradamente permitidos para que execuções externas possam acionar os controllers
 - Implemente a interface WebMvcConfigurer
 - e o método addCorsMapping
 - Botão Direito → Source → Override/Implement Methods → addCorsMapping





- Implementar a liberação cors dentro do método `addCorsMappings()`, pode remover o conteúdo existente.

```
registry.addMapping("/**").allowedOrigins("*").allowedMethods("GET", "POST", "PUT", "DELETE");
```

```
package br.com.entra21.backend.spring.entra21Pratica;

import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

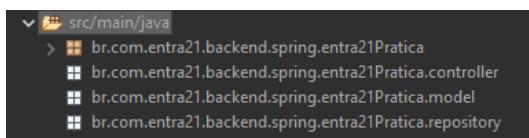
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**").allowedOrigins("*").allowedMethods("GET", "POST", "PUT", "DELETE");
    }
}
```

Criar Sub Pacotes

- Controller

- Responsável por ter classes que representam controllers, os controllers são métodos mapeados para serem executados quando o verbo HTTP + possíveis parâmetros na rota são equivalentes a sua configuração
- Quando o mapeamento aciona o método, a lógica de programação será executada e no final é esperado que seja dado uma resposta para quem solicitou
- Model
 - Representam as abstrações que definem as entidades do sistema
 - Essas entidades também são traduzidas no modelo definido no banco de dados
- Repository
 - Responsável por gerenciar as ações de captura e manutenção no banco de dados como consultas, inserções, atualizações e exclusões
- Outros sub pacotes podem e devem ser criados sempre que houver necessidade para manter o projeto organizado como:
 - interface
 - exceptions
 - util
 - etc



Criar Classe SistemaController

- Criar uma classe com nome SistemaController para fins de verificação do sistema e futuramente outros métodos relevantes a esse contexto
- Antes do nome da classe deve conter as anotações
 - @RestController
 - Define que essa classe é um controller
 - @CrossOrigin(origins = "*")
 - Define que as restrições cors estão permitindo qualquer origin, isso é importante pois permite que seja bloqueado futuramente dependendo da regra de negócio atual
 - @RequestMapping("/sistema")
 - Inclui a necessidade de um prefixo antes de chamar o método do controller
 - Exemplo
 - Para acionar a rota /login seria necessário escrever na url `http://seudominioaqui.com/sistema/login`

```

package br.com.entra21.backend.spring.entra21Pratica.controller;

import org.springframework.web.bind.annotation.CrossOrigin;

```

```

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/sistema")
public class SistemaController {

}

```

- Primeiro método mapeado no controller

```

package br.com.entra21.backend.spring.entra21Pratica.controller;

import org.springframework.ui.Model;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/sistema")
public class SistemaController {

    @GetMapping()
    @ResponseStatus(HttpStatus.OK)
    public String on(Model model) {

        return "Estou ON";
    }
}

```

- Se o sistema for executado é possível acionar esse controller através da url <http://localhost:8080/sistema>
- Localize a janela Boot Dashboard geralmente localizada abaixo da janela Package Explorer e possui um ícone de POWER dentro de um HEXAGONO VERDE
- Selecione o seu projeto na carga local > seuProjeto e execute o (Re)start que tem o ícone de STOP+PLAY
- Teste no navegador ou no POSTMAN com a url <http://localhost:8080/sistema>
- O resultado esperado é o texto retornado no método

The screenshot shows the Postman interface with the following details:

- Request URL:** `http://localhost:8080/sistema`
- Method:** GET
- Headers:** (6 hidden)

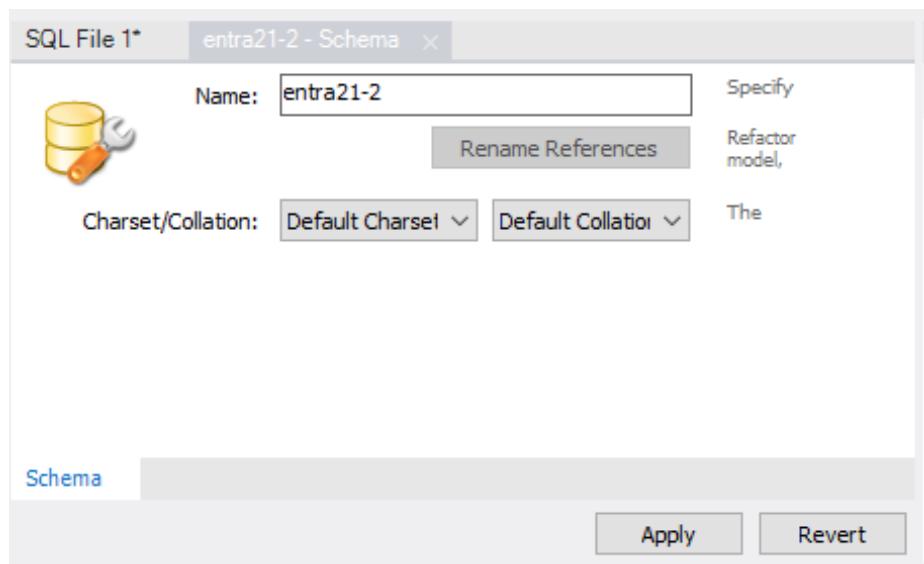
KEY	VALUE	DESCRIPTION	... Bulk Edit	Presets
Key	Value	Description	... Bulk Edit	Presets
- Body:** (Pretty, Raw, Preview, Visualize, Text)

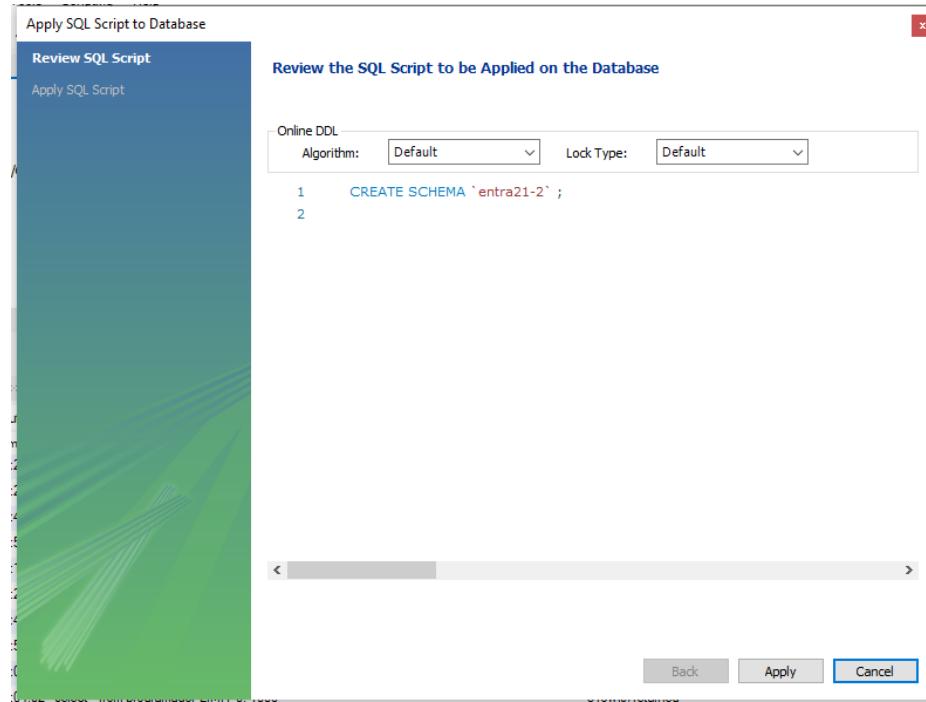

```
1 Estou ON
```
- Response Headers:**

Header	Value
Status	200 OK
Time	245 ms
Size	260 B
- Response Body:** `Estou ON`

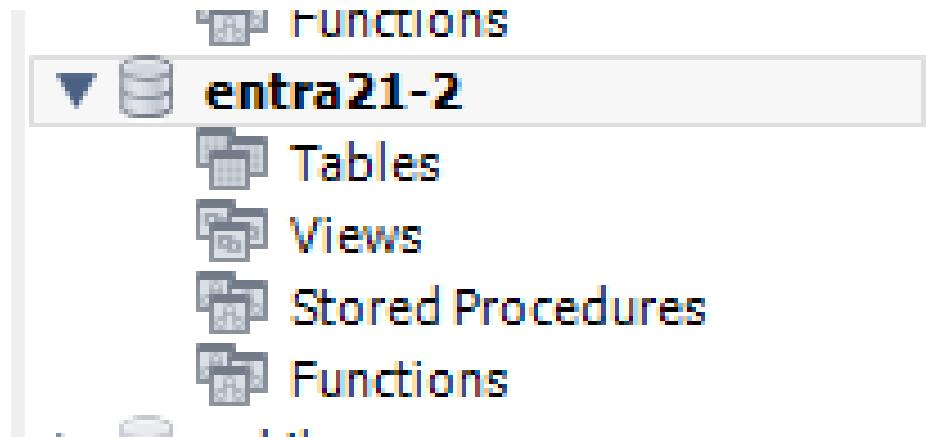
Configurando banco de dados

- Inicie o MySql Workbench e caso não possua, acompanhe a utilização.
 - Na tela inicial deve existir uma instancia MySQL80
 - Caso não exista é possível criar
- É muito importante não esquecer as credenciais dessa instancia
 - Sugestão
 - user
 - root
 - password
 - Mysql123@
 - Anote esse dados pois serão solicitados no projeto Spring
- Inicie a instancia
- Localize os ícones no menu superior
 - Create a new schema in the connected server
 - informe o nome
 - entra21
 - Nos buttons inferiores desse forme selecione
 - apply

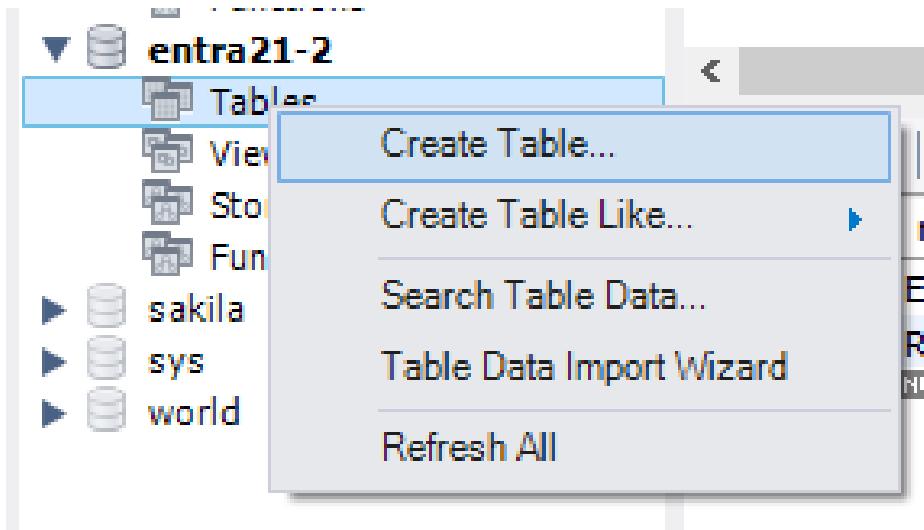




- Na interface principal na lateral esquerda, deve conter uma carga de schemas
 - localize o entra21 e expanda a carga



- Localize a sub carga tables e click com o botão direito do mouse
 - Selecione a opção create table
 - A interface para criação de tabelas muda pouco de ferramenta para ferramenta



- Crie uma tabela com a seguinte estrutura
 - Nome da tabela
 - programador
 - Campos
 - id
 - INT
 - primary key (PK - checkbox)
 - not null (NN - checkbox)
 - autoincrement(AI - checkbox)
 - nome
 - VARCHAR(45)
 - not null (NN - checkbox)
 - qtd_linguagem
 - INT
 - Nos buttons inferiores desse forme selecione
 - apply

The screenshot shows the MySQL Workbench interface for creating a new table. The table name is set to 'programador'. The schema is 'entra21-2'. The engine is set to InnoDB. The table has three columns: 'id' (INT, Primary Key, Auto Increment), 'nome' (VARCHAR(45)), and 'qtd_linguagem' (INT). The 'Columns' tab is selected, and the 'Apply' button is visible at the bottom right.

The screenshot shows the 'Review SQL Script' dialog in MySQL Workbench. It displays the SQL code for creating the 'programador' table:

```

CREATE TABLE `entra21-2`.`programador` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `qtd_linguagem` INT NULL,
  PRIMARY KEY (`id`)
);

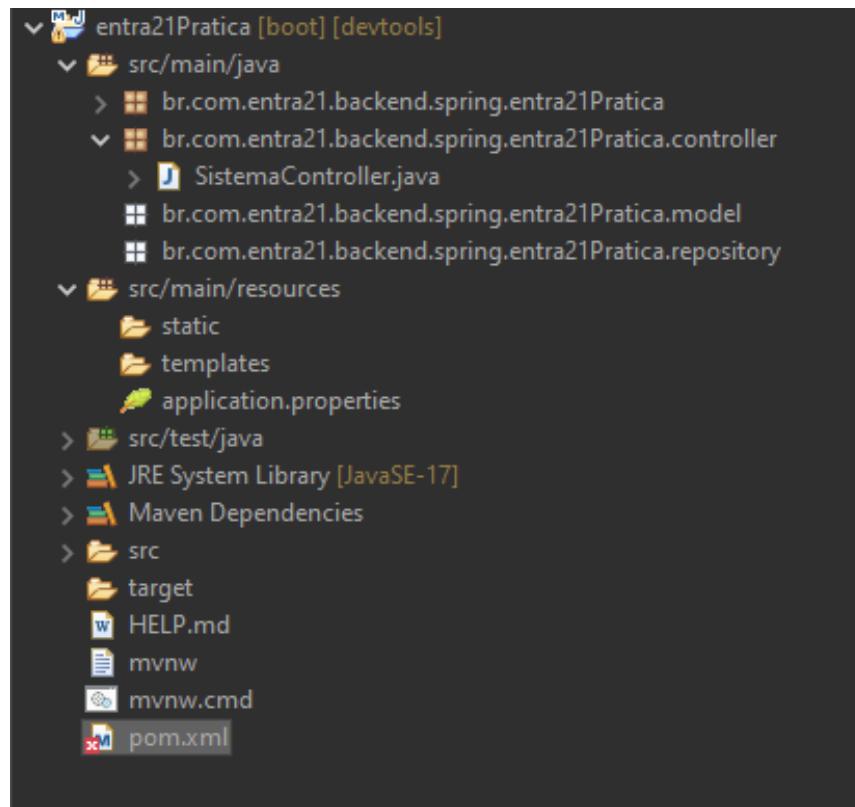
```

The 'Apply' button is highlighted at the bottom right of the dialog.

- A tabela ja deve existir na carga de tables do schema entra21

Adicionar Dependencias para Integração do Banco de Dados

- Localize o arquivo pom.xml na raiz do projeto



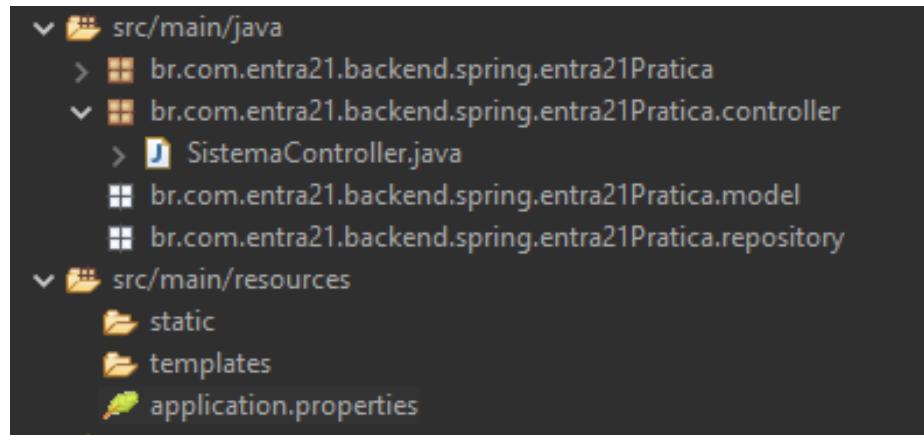
- Garanta que as seguintes dependências estejam no arquivo, pois serão necessárias:

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.3</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

Integrando banco de dados ao SpringBoot

- No projeto spring localize a pasta src/main/resources



- Edite o arquivo application.properties
- Inclua a instruções abaixo

```
spring.datasource.url=jdbc:mysql://localhost:3306/entra21-2      //Endereço e nome do banco
spring.datasource.username=root                                //Usuario do banco
spring.datasource.password=Mysql123@                            //Senha do banco

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57InnoDBDialect
```

Utilizando o evento de inicialização do projeto para fazer um select na tabela

- No pacote principal localize a classe Application
- Crie um novo atributo nessa classe

```
@Autowired
private JdbcTemplate jdbc;
```

- @Autowired significa que o objeto com nome jdbc da Classe JdbcTemplate será instanciado juntamente com essa classe Application
- Localize o metodo run gerado ao implementar a interface CommandLineRunner
- Inclua co código

```
String sql = "INSERT INTO programador (nome, qtd_linguagem) VALUES (?, ?)";
int result = jdbc.update(sql, "Emerson", 4);
```

```

package br.com.entra21.backend.spring.entra21Pratica;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jdbc.core.JdbcTemplate;

@SpringBootApplication
public class Entra21PraticaApplication implements CommandLineRunner {

    @Autowired
    private JdbcTemplate jdbct;

    public static void main(String[] args) {
        SpringApplication.run(Entra21PraticaApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {

        String sql = "INSERT INTO programador (nome, qtd_linguagem) VALUES (?, ?)";
        int result = jdbct.update(sql, "Emerson", 4);

    }
}

```

CRUD de programadores

- Criar no package de model as classes
 - Classe Programador
 - Colocar no topo da classe as anotations `@Entity` e `@Table(name="programador")`
 - Herdando de `MaturidadeNivel3Richardson`
 - Com os atributos definidos na tabela do banco de dados
 - `id`
 - Recebe Anotation `@Id` e `@GeneratedValue(strategy=GenerationType.IDENTITY)`
 - `nome`
 - `qtd_linguagem`

```

package br.com.entra21.backend.spring.entra21Pratica.model;

import java.util.ArrayList;

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

@Entity
@Table(name="programador")
public class Programador extends MaturidadeNivel3Richardson {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String nome;
    private Integer qtd_linguagem;
    public Programador() {
        super();
        // TODO Auto-generated constructor stub
    }
}

```

```

    }
    public Programador(ArrayList<ItemNivel3> links) {
        super(links);
        // TODO Auto-generated constructor stub
    }
    public Programador(Integer id, String nome, Integer qtd_linguagem) {
        super();
        this.id = id;
        this.nome = nome;
        this.qtd_linguagem = qtd_linguagem;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public Integer getQtd_linguagem() {
        return qtd_linguagem;
    }
    public void setQtd_linguagem(Integer qtd_linguagem) {
        this.qtd_linguagem = qtd_linguagem;
    }
}

```

- Classe **ItemNivel3**

- criar os atributos ref, url, headers e body com encapsulamento, seus construtores vazios e argumentos

```

package br.com.entra21.backend.spring.projeto.model;

import java.util.ArrayList;

public class ItemNivel3 {

    private String ref;
    private String url;
    private ArrayList<String> headers;
    private String body;

    public ItemNivel3() {
    }

    public ItemNivel3(String ref, String url, ArrayList<String> headers, String body) {
        super();
        this.ref = ref;
        this.url = url;
        this.headers = headers;
        this.body = body;
    }

    public String getRef() {
        return ref;
    }

    public void setRef(String ref) {
        this.ref = ref;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public ArrayList<String> getHeaders() {
        return headers;
    }
}

```

```

    public void setHeaders(ArrayList<String> headers) {
        this.headers = headers;
    }
    public String getBody() {
        return body;
    }
    public void setBody(String body) {
        this.body = body;
    }
}

```

- Classe MaturidadeNivel3Richardson

- criar os atributos ArrayList<ItemNivel3> links com encapsulamento, seu construtor vazios e argumentos

```

package br.com.entra21.backend.spring.entra21Pratica.model;

import java.util.ArrayList;

public class MaturidadeNivel3Richardson {

    ArrayList<ItemNivel3> links;

    public MaturidadeNivel3Richardson() {
        this.links = new ArrayList<>();
    }

    public MaturidadeNivel3Richardson(ArrayList<ItemNivel3> links) {
        super();
        this.links = links;
    }

    public ArrayList<ItemNivel3> getLinks() {
        return links;
    }

    public void setLinks(ArrayList<ItemNivel3> links) {
        this.links = links;
    }

}

```

- Criar um sub pacote repository

- Criar uma INTERFACE com nome IProgramadorRepository

```

package br.com.entra21.backend.spring.entra21Pratica.repository;

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

import br.com.entra21.backend.spring.entra21Pratica.model.Programador;

public interface IProgramadorRepository extends JpaRepository<Programador, Integer> {
}

```

- Nessa interface herdar extends JpaRepository<Programador, Integer>

- Essa interface garante vários métodos para acesso ao banco de dados e que entendem a classe Programador e a tabela programador

- Criar no sub pacote de controllers a classe ProgramadorController
- Antes da classe

```
@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/programadores")
```

- Dentro da classe

```
private final String PATH = "localhost:8080/programadores";

@Autowired
private IProgramadorRepository programadorRepository;
```

```
package br.com.entra21.backend.spring.entra21Pratica.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.entra21.backend.spring.entra21Pratica.repository.IProgramadorRepository;

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/programadores")
public class ProgramadorController {

    private final String PATH = "localhost:8080/programadores";

    @Autowired
    private IProgramadorRepository programadorRepository;

}
```

Inicializando o Servidor

- Quando inicializado o servidor, ele deve ter criado na tabela um item com os dados informados.

	id	nome	qtd_linguagem
▶	1	Emerson	4
*	NULL	NULL	NULL

- Após o teste comentar o código dentro do método run

```

package br.com.entra21.backend.spring.entra21Pratica;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jdbc.core.JdbcTemplate;

@SpringBootApplication
public class Entrada21PraticaApplication implements CommandLineRunner {

    @Autowired
    private JdbcTemplate jdbc;

    public static void main(String[] args) {
        SpringApplication.run(Entrada21PraticaApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {

        String sql = "INSERT INTO programador (nome, qtd_linguagem) VALUES (?, ?)";
        int result = jdbc.update(sql, "Emerson", 4);

    }
}

```

Criando Métodos para CRUD Programadores

- Listar Todos os Programadores

```

@GetMapping()
@ResponseStatus(HttpStatus.OK)
public List<Programador> listar() {
    return programadorRepository.findAll();
}

```

http://localhost:8080/programadores

GET http://localhost:8080/programadores

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "links": [],
4     "id": 3,
5     "name": "Ruben",
6     "qtd_linguagem": 300
7   },
8   {
9     "links": [],
10    "id": 4,
11    "name": "Emerson",
12    "qtd_linguagem": 4
13  }
14 ]

```

Status: 200 OK Time: 279 ms Size: 364 B Save Response

- Listar Um Programador por ID

```

@GetMapping("/{id}")
@ResponseStatus(HttpStatus.OK)
public List<Programador> buscar(@PathVariable("id") int param) {
}

```

```

List<Programador> response = programadorRepository.findById(param).stream().toList();

return response;
}

```

http://localhost:8080/programadores/4

GET http://localhost:8080/programadores/4

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Send Save

1

Status: 200 OK Time: 24 ms Size: 309 B Save Response

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "links": [],
4     "id": 4,
5     "nome": "Emerson",
6     "qtd_linguagem": 4
7   }
8 ]

```

- Criar Um Programador

```

@PostMapping()
@ResponseBody(HttpStatus.CREATED)
public @ResponseBody Programador adicionar(@RequestBody Programador novoProgramador) {

    return programadorRepository.save(novoProgramador);
}

```

http://localhost:8080/programadores/

POST http://localhost:8080/programadores/

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Send Save

1
2 ...
3 ...
4

Status: 201 Created Time: 33 ms Size: 312 B Save Response

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "links": [],
4     "id": 4,
5     "nome": "Emerson",
6     "qtd_linguagem": 4
7   }
8 ]

```

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```
1 select * from programador;
```

The result grid displays the following data:

	id	nome	qtd_linguagem
▶	1	Emerson	4
▶	2	Rubem	300
*	NULL	NULL	NULL

- Editar Um Programador

```
@PutMapping("/{id}")
@ResponseStatus(HttpStatus.OK)
public @ResponseBody Optional<Programador> atualizar(@PathVariable("id") int param, @RequestBody Programador programadorNovosDados) {
    Programador atual = programadorRepository.findById(param).get();
    atual.setNome(programadorNovosDados.getNome());
    atual.setQtd_linguagem(programadorNovosDados.getQtd_linguagem());
    programadorRepository.save(atual);

    return programadorRepository.findById(param);
}
```

The screenshot shows a Postman request to <http://localhost:8080/programadores/4> using the PUT method.

Body tab (JSON format):

```
1 {
2   ...
3   "nome": "Emerson",
4   ...
5   "qtd_linguagem": 6
6 }
```

Response tab (Pretty JSON format):

```
1 {
2   ...
3   "links": [],
4   "id": 4,
5   "nome": "Emerson",
6   "qtd_linguagem": 6
7 }
```

Query 1

```

1  select * from programador;
2
3 • delete from programador where id =2;

```

Result Grid

	id	nome	qtd_linguagem
▶	3	Rubem	300
4	Emerson	6	
*	NULL	NULL	NULL

- Deletar Um Programador

```

@DeleteMapping("/{id}")
@ResponseStatus(HttpStatus.OK)
public @ResponseBody boolean deletar(@PathVariable("id") int id) {
    programadorRepository.deleteById(id);

    return !programadorRepository.existsById(id);
}

```

http://localhost:8080/programadores/3

DELETE http://localhost:8080/programadores/3

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

This request does not have a body

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

1 true

Status: 200 OK Time: 92 ms Size: 257 B Save Response

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a query editor window titled "Query 1" containing the SQL command:

```

1  select * from programador;
2

```

Below the query editor is a results grid window. It has a header row with columns labeled "id", "nome", and "qtd_linguagem". There is one data row below the header, which contains the values 4, Emerson, and 6 respectively. The "id" column has a tooltip "4" above it.

- Classe ProgramadorController com CRUD

```

package br.com.entra21.backend.spring.projeto.controller;

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import br.com.entra21.backend.spring.projeto.model.Programador;
import br.com.entra21.backend.spring.projeto.repository.IProgramadorRepository;

@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/programadores")
public class ProgramadorController {

    @Autowired
    private IProgramadorRepository programadorRepository;

    @GetMapping()
    @ResponseStatus(HttpStatus.OK)
    public List<Programador> listar() {
        return programadorRepository.findAll();
    }

    @GetMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    public List<Programador> buscar(@PathVariable("id") int param) {

        List<Programador> response = programadorRepository.findById(param).stream().toList();

        return response;
    }

    @PostMapping()
    @ResponseStatus(HttpStatus.CREATED)
    public @ResponseBody Programador adicionar(@RequestBody Programador novoProgramador) {

```

```

        return programadorRepository.save(novoProgramador);
    }

    @PutMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody Optional<Programador> atualizar(@PathVariable("id") int param, @RequestBody Programador programadorNovosDados) {

        Programador atual = programadorRepository.findById(param).get();
        atual.setNome(programadorNovosDados.getNome());
        atual.setQtd_linguagem(programadorNovosDados.getQtd_linguagem());
        programadorRepository.save(atual);

        return programadorRepository.findById(param);
    }

    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody boolean deletar(@PathVariable("id") int id) {
        programadorRepository.deleteById(id);

        return !programadorRepository.existsById(id);
    }
}

```

Projeto

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/bc0878c0-b689-49b7-9491-1351d1bcda88/entra21Pratica.zip>