



Spring Boot

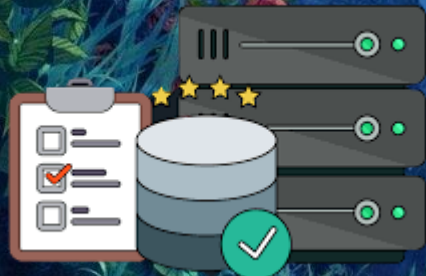
Criando a primeira API Rest em Spring Boot



O spring são pacotes de classes pré definidas em linguagem java que por sua vez facilita o dia de trabalho do desenvolvedor.

Estou criando um blog de um api rest com a ferramenta framework spring boot, sobre a criação de controle de veículos de usuários.

Meu github: <https://github.com/MarioMess/OrangeTalent>



Ferramentas utilizada para essa api rest



IntelliJ Community ferramenta de desenvolvimento do código da api na linguagem java que por sua vez é uma ferramenta de trabalho que traz conforto e clareza na amostra de suas aplicações e interações tanto com o usuário quanto com os mecanismos que ela interage.

H2

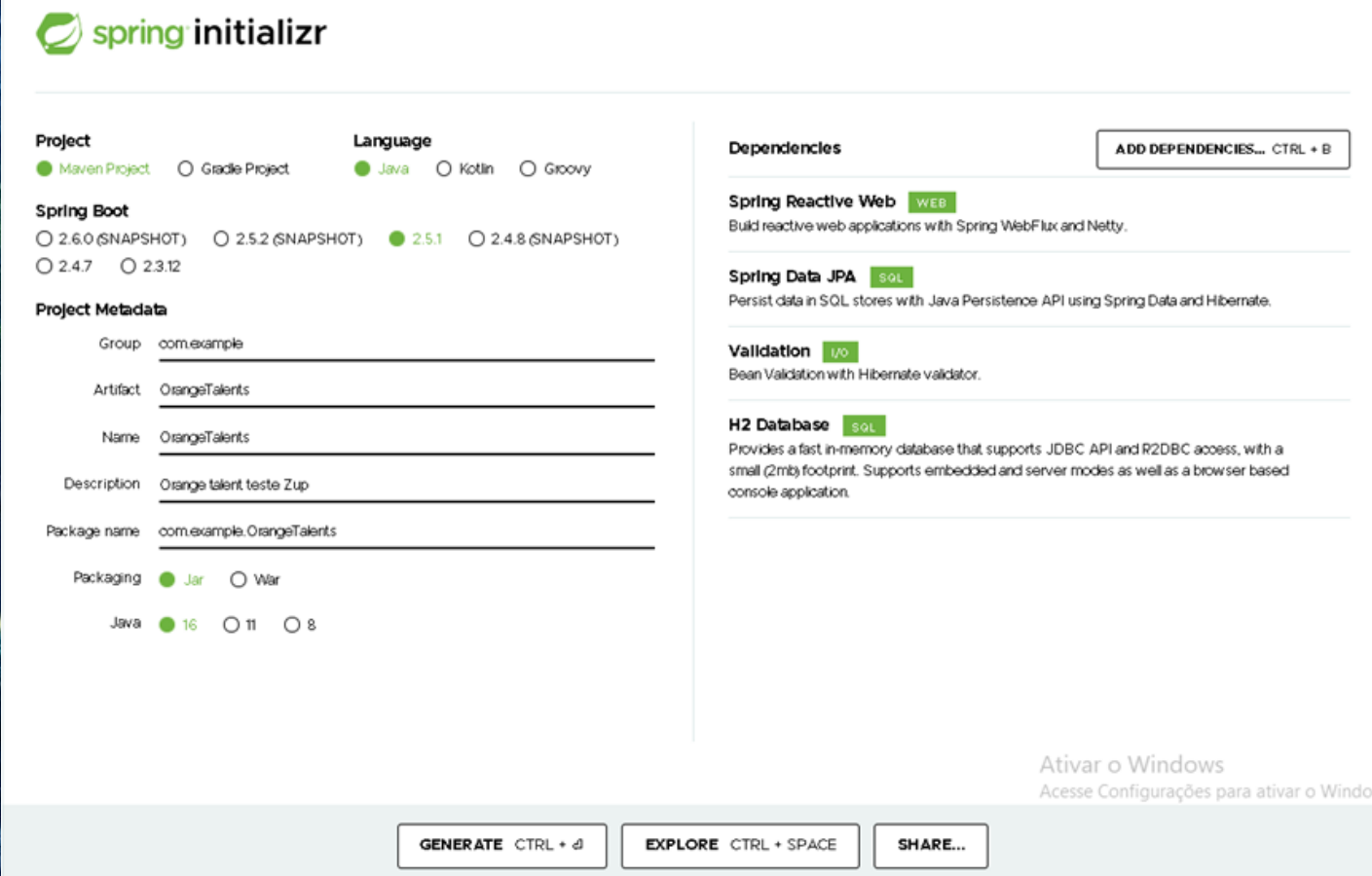
H2 é um banco de dados versátil de fácil abordagem tanto de aprendizado e apresentação e por ser um banco de dados na memória ajuda na velocidade de trabalho.



Postman para visualização do meu projeto em andamento eu utilizei o postman, que é uma plataforma que ajuda no desenvolvimento de api que através dela não preciso criar interface de visualização e nem criar caminhos complicados para que ela me mostre algum resultado, assim detalhando quais são meus lançamentos positivos ou negativos.

Iniciando projeto spring

O primeiro passo para inicializar um projeto em Spring, é ir neste site <https://start.spring.io/>, no site contém as inserções de dependências para a criação de um projeto em spring boot,



The screenshot shows the Spring Initializr web application interface. It features a header with the 'spring initializr' logo. The main content area is divided into several sections: 'Project' with radio buttons for 'Maven Project' (selected) and 'Gradle Project'; 'Language' with radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'; 'Spring Boot' with radio buttons for versions 2.6.0 (SNAPSHOT), 2.5.2 (SNAPSHOT), 2.5.1 (selected), and 2.4.8 (SNAPSHOT), plus 2.4.7 and 2.3.12; 'Project Metadata' with input fields for Group (com.example), Artifact (OrangeTalents), Name (OrangeTalents), Description (Orange talent teste Zup), and Package name (com.example.OrangeTalents), along with 'Packaging' (Jar selected, War) and 'Java' version (16 selected, 11, 8); and 'Dependencies' with a list of selected dependencies: 'Spring Reactive Web' (WEB), 'Spring Data JPA' (SQL), 'Validation' (JPA), and 'H2 Database' (SQL). Each dependency has a brief description. At the bottom right, there is a message 'Ativar o Windows' and a link to 'Acesse Configurações para ativar o Windows'. The footer contains three buttons: 'GENERATE' (CTRL + G), 'EXPLORE' (CTRL + SPACE), and 'SHARE...'.

Project
☒ Maven Project ☐ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 2.6.0 (SNAPSHOT) ☐ 2.5.2 (SNAPSHOT) ☒ 2.5.1 ☐ 2.4.8 (SNAPSHOT)
☐ 2.4.7 ☐ 2.3.12

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☒ 16 ☐ 11 ☐ 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Reactive Web WEB
Build reactive web applications with Spring WebFlux and Netty.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Validation JPA
Bean Validation with Hibernate validator.

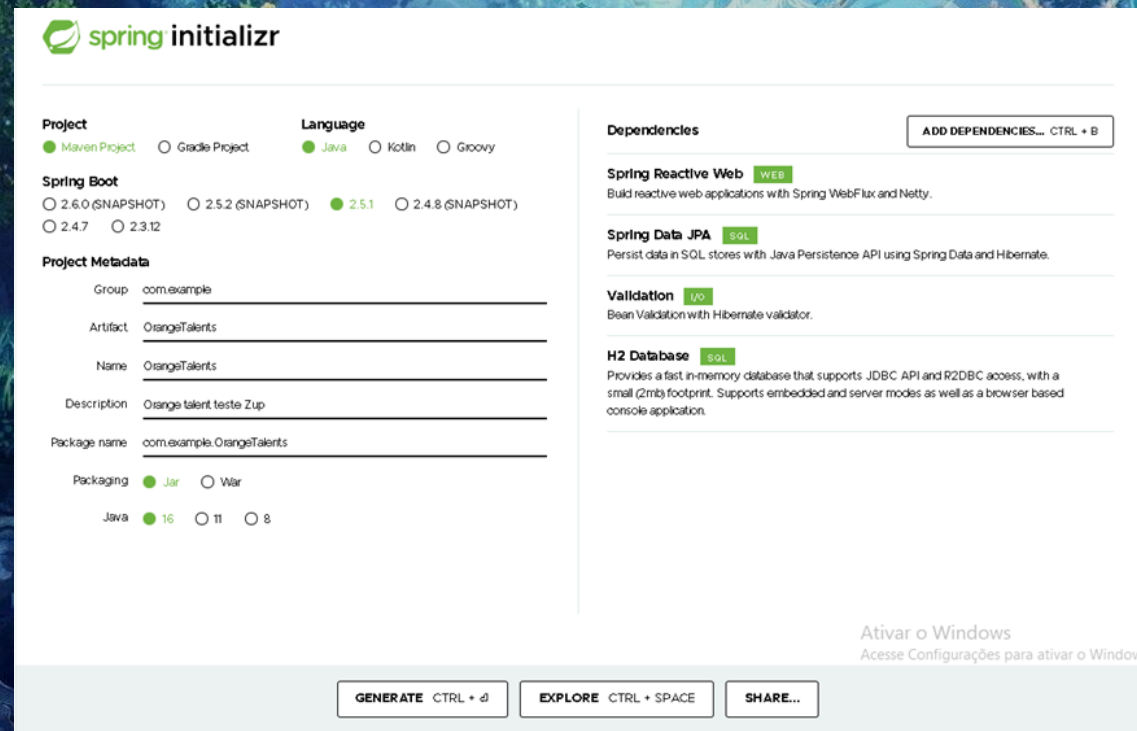
H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Ativar o Windows
Acesse Configurações para ativar o Windows

GENERATE CTRL + G **EXPLORE** CTRL + SPACE **SHARE...**

Spring IO configuração

Insérer o nome do projeto no campo Name, e inserir uma descrição do projeto no campo Description, verificar a versão selecionado do spring, a linguagem que será utilizada será java, o framework utilizado e o Maven, verificar como deseja exportar seu projeto em ponto jar ou em war(utilizei o jar), verificar também a versão da linguagem java que deseja utilizar(utilizei a versão 16), após tudo isso vamos clicar no botão Generate para gerar seu projeto.



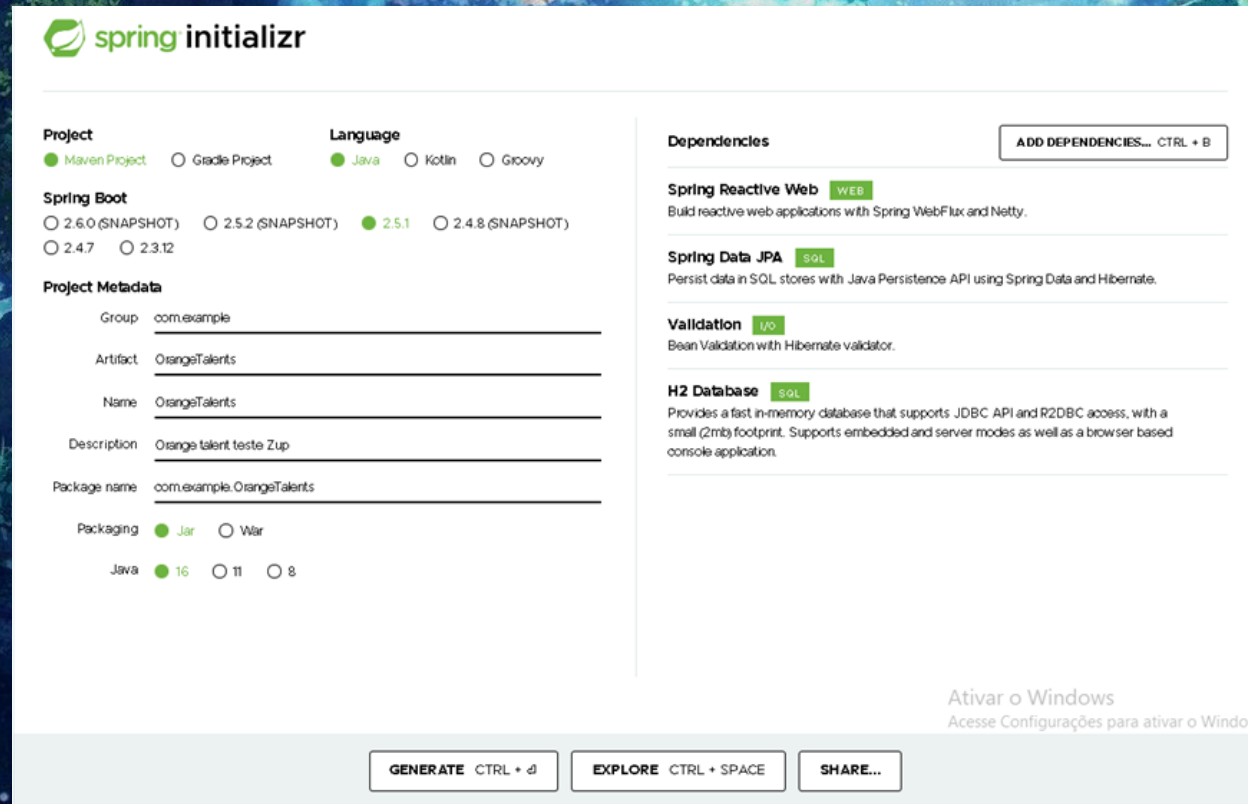
The image shows the Spring Initializr web interface, which is used to generate a Spring project. The interface is divided into several sections:

- Project:** Includes radio buttons for **Maven Project** (selected) and **Gradle Project**.
- Language:** Includes radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Includes radio buttons for versions 2.6.0 (SNAPSHOT), 2.5.2 (SNAPSHOT), **2.5.1** (selected), 2.4.8 (SNAPSHOT), 2.4.7, and 2.3.12.
- Project Metadata:** Includes input fields for **Group** (com.example), **Artifact** (OrangeTalents), **Name** (OrangeTalents), **Description** (Orange talent teste Zip), and **Package name** (com.example.OrangeTalents).
- Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
- Java:** Includes radio buttons for versions **16** (selected), 11, and 8.
- Dependencies:** Includes a button **ADD DEPENDENCIES... CTRL + B** and a list of dependencies:
 - Spring Reactive Web** (WEB): Build reactive web applications with Spring WebFlux and Netty.
 - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
 - Validation** (JOO): Bean Validation with Hibernate validator.
 - H2 Database** (SQL): Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

At the bottom, there are buttons for **GENERATE CTRL + G**, **EXPLORE CTRL + SPACE**, and **SHARE...**. A footer message says "Ativar o Windows" and "Acesse Configurações para ativar o Windows".

Adicionando as dependências

Cada dependência tem sua função para que o projeto compile, a primeira dependência: spring web é para criar uma web service, spring data JPA é a camada da persistência de dados, validation é para validar as informações inseridas no banco de dados, H2 é a dependência relacionada ao banco h2, onde será salva as informações gerado do projeto.

The image shows the Spring Initializr web form, which is used to generate a new Spring project. The form is divided into several sections: Project, Language, Spring Boot, Project Metadata, Dependencies, and a footer. The Project section has radio buttons for Maven Project (selected) and Gradle Project. The Language section has radio buttons for Java (selected), Kotlin, and Groovy. The Spring Boot section has radio buttons for versions 2.6.0 (SNAPSHOT), 2.5.2 (SNAPSHOT), 2.5.1 (selected), and 2.4.8 (SNAPSHOT), as well as 2.4.7 and 2.3.12. The Project Metadata section has input fields for Group (com.example), Artifact (OrangeTalents), Name (OrangeTalents), Description (Orange talent teste Zup), and Package name (com.example.OrangeTalents). The Packaging section has radio buttons for Jar (selected) and War, and a section for Java version with radio buttons for 16 (selected), 11, and 8. The Dependencies section has a button to add dependencies and lists three selected dependencies: Spring Reactive Web (WEB), Spring Data JPA (SQL), and Validation (I/O). The footer has buttons for GENERATE (CTRL + G), EXPLORE (CTRL + SPACE), and SHARE...

spring inicializr

Project
☒ Maven Project ☐ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 2.6.0 (SNAPSHOT) ☐ 2.5.2 (SNAPSHOT) ☒ 2.5.1 ☐ 2.4.8 (SNAPSHOT)
☐ 2.4.7 ☐ 2.3.12

Project Metadata
Group:
Artifact:
Name:
Description:
Package name:
Packaging: ☒ Jar ☐ War
Java: ☒ 16 ☐ 11 ☐ 8

Dependencies [ADD DEPENDENCIES... CTRL + B](#)

Spring Reactive Web **WEB**
Build reactive web applications with Spring WebFlux and Netty.

Spring Data JPA **SQL**
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

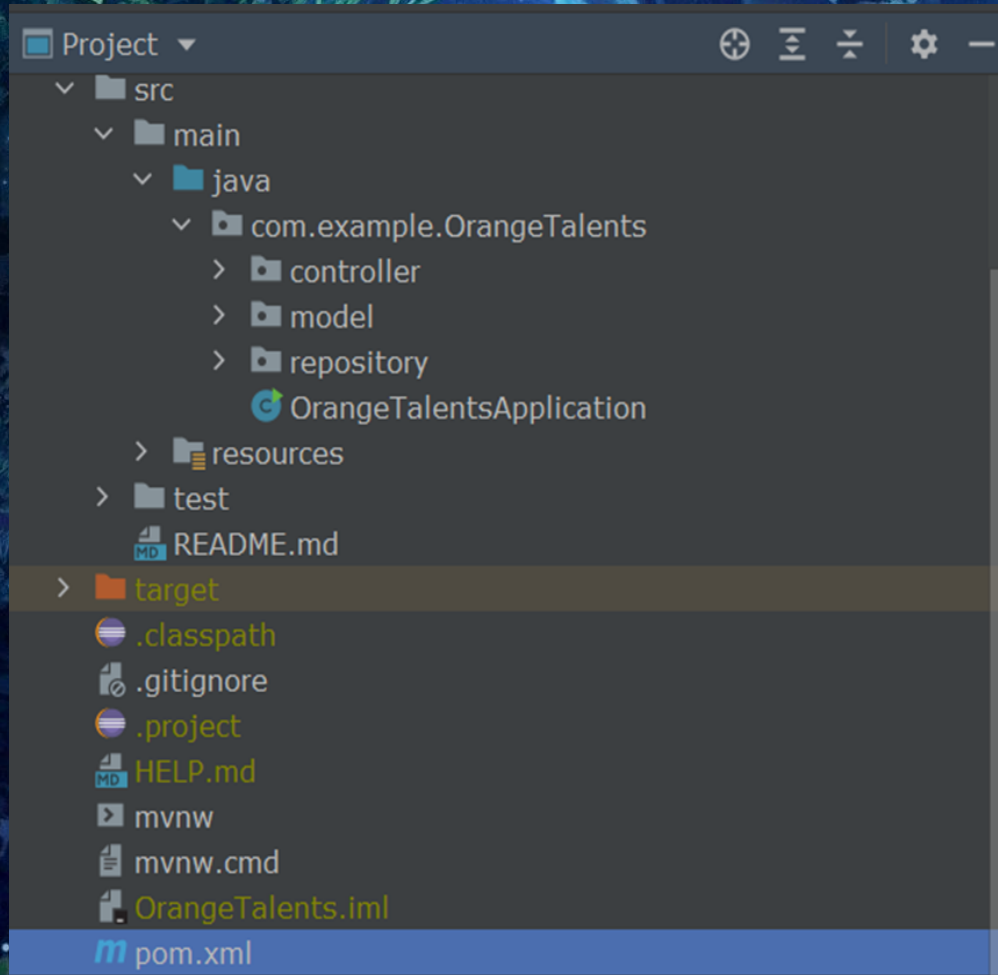
Validation **I/O**
Bean Validation with Hibernate validator.

H2 Database **SQL**
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Ativar o Windows
Acesse Configurações para ativar o Windows

GENERATE CTRL + G **EXPLORE** CTRL + SPACE **SHARE...**

As dependências do projeto ficam no arquivo chamado pom.xml:

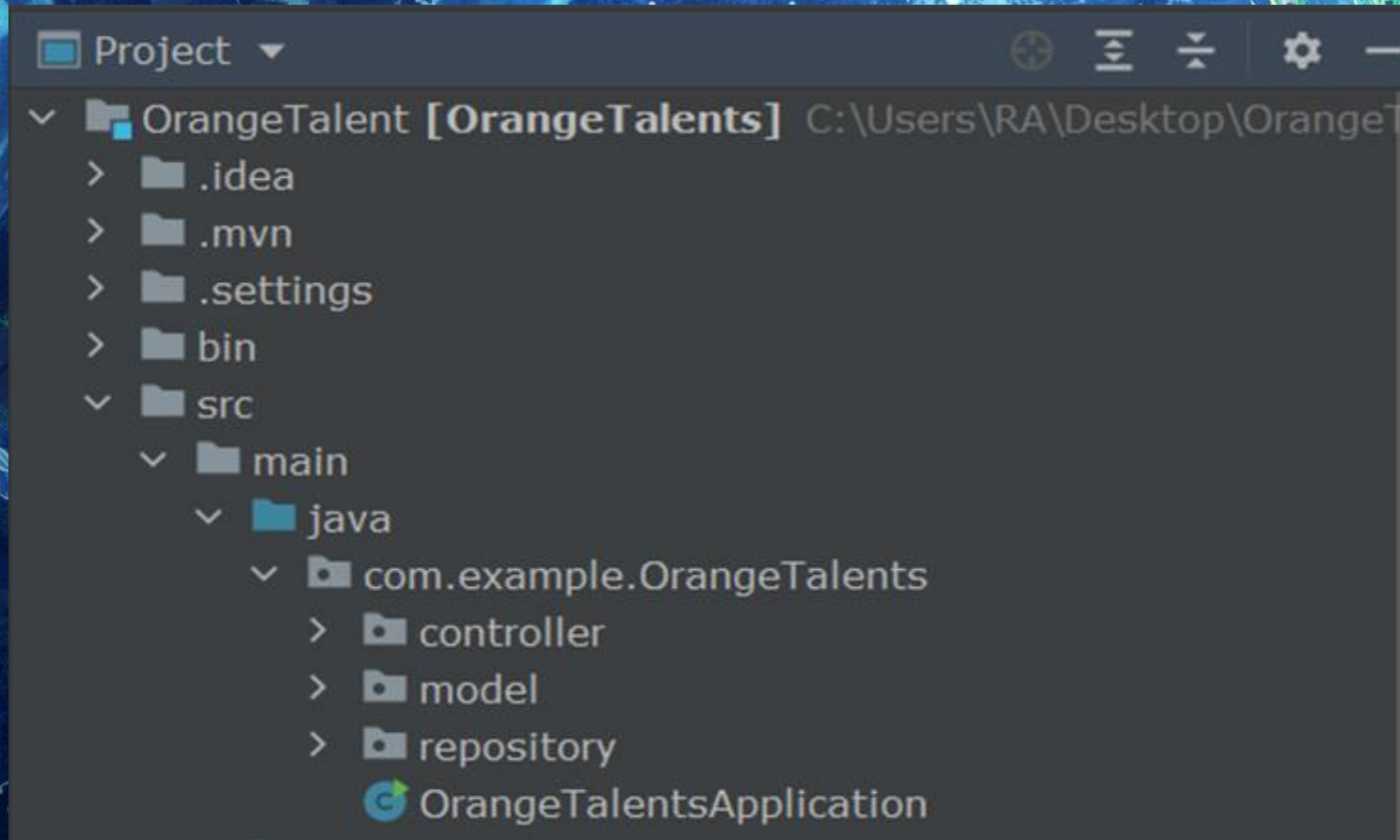


```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.5.1</version>
9         <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.example</groupId>
12    <artifactId>OrangeTalents</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>OrangeTalents</name>
15    <description>Orange talent teste Zup</description>
16    <properties>
17        <java.version>16</java.version>
18    </properties>
19    <dependencies>
20        <dependency>
21            <groupId>org.springframework.boot</groupId>
22            <artifactId>spring-boot-starter-data-jpa</artifactId>
23        </dependency>
24        <dependency>
25            <groupId>com.h2database</groupId>
26            <artifactId>h2</artifactId>
27        </dependency>
28
29        <dependency>
30            <groupId>org.springframework.boot</groupId>
31            <artifactId>spring-boot-starter-validation</artifactId>
32        </dependency>
33        <dependency>
34            <groupId>org.springframework.boot</groupId>
35            <artifactId>spring-boot-starter-web</artifactId>
36        </dependency>
37
38        <dependency>
39            <groupId>org.springframework.boot</groupId>
40            <artifactId>spring-boot-starter-test</artifactId>
41            <scope>test</scope>
42        </dependency>
```


O projeto ao inicializar ele já vem com uma pasta inicial, ao clicar no arquivo OrangeTalentsApplication.java, dentro da pasta será executado o projeto.

Criar os pacotes

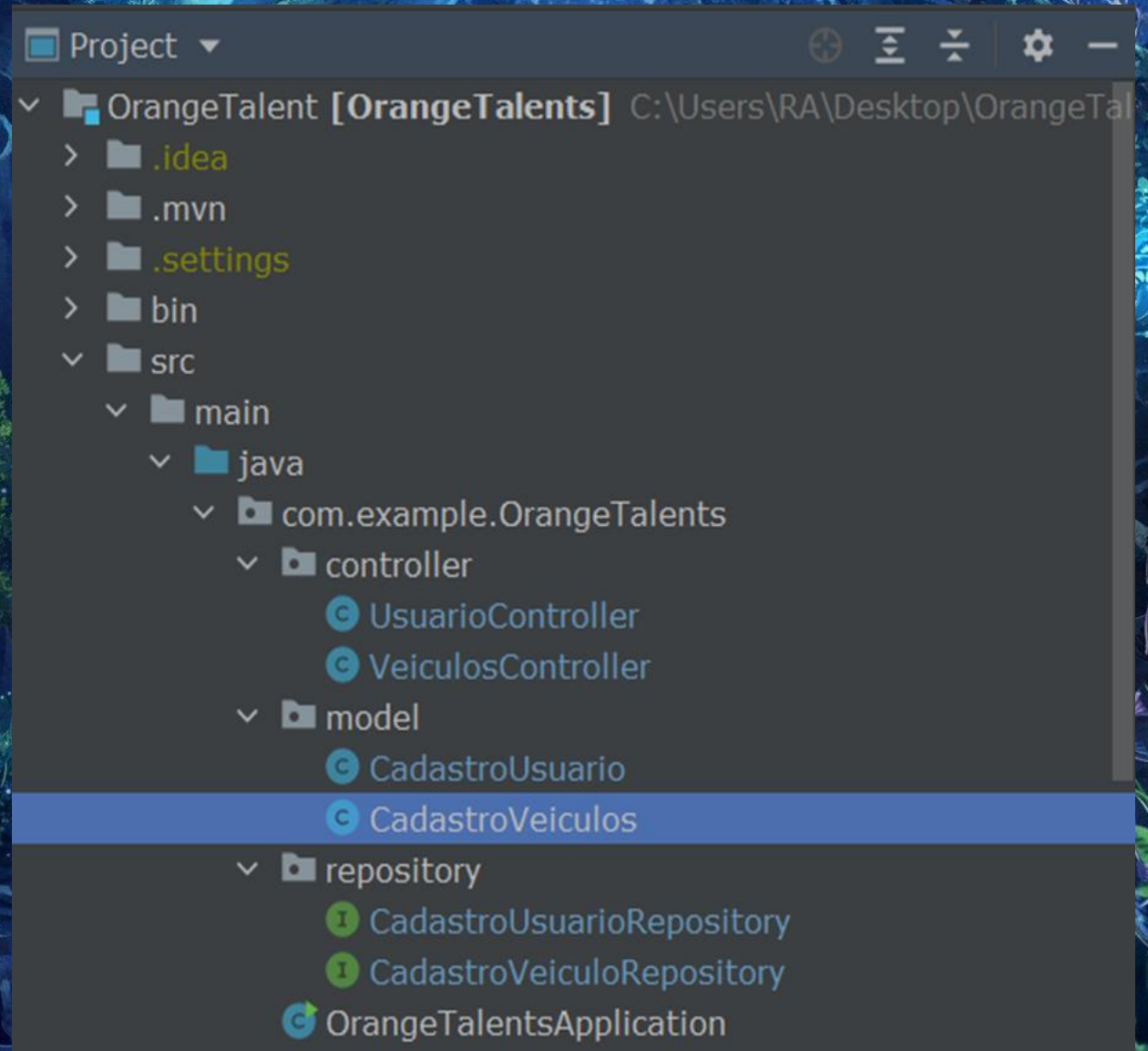
- model
- controller
- repository



A Model

As classes criadas dentro da Model/Entity

- CadastroUsuario
- CadastroVeiculos



As anotações

No pacote model será criado a seguinte classe CadastroUsuario, com os seguintes atributos id nome, data de nascimento, cpf e email, contendo as anotações para validação,

@Id indica que é um id ou seja identificador do usuário.

@GeneratedValue utilizado para que o banco de dados gerencie a entrada do id,

@NotBlank o campo não pode ser nulo e seu valor inicial tem que ser maior que zero.

@NotNull o campo não pode ser nulo.

@CPF o campo segue o padrão brasileiro de 11 dígitos.

@Email segue o padrão email Ex: email@email.com.

@Column (unique=true) usado para referenciar uma coluna específica no banco de dados, onde não deve ser inserida uma informação repetida.

@OneToMany é a ligação de uma tabela a outra onde um usuário contém vários itens.

Na questão da data do usuário foi usado o Date com a importação

```
import java.util.Date;
```

No atributo de ligação **@OneToMany** foi usado uma List onde são vários veículos para um usuário. O List com a importação

```
import java.util.List;
```

```
@Entity(name = "usuario")
public class CadastroUsuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    private String nome;

    @NotBlank(message = "Email inválido")
    @Email(message = "Email válido")
    @Column(unique = true)
    private String email;

    @NotBlank(message = "CPF válido")
    @CPF
    @Column(unique = true)
    private String cpf;

    @NotNull
    private Date dataNascimento;

    @OneToMany(mappedBy = "usuario", cascade = CascadeType.ALL)
    @Column(name = "id")
    private List<CadastroVeiculos> idveiculo;
```




Na classe veículos foi inserido os seguintes atributos

```
@Entity(name = "veiculos")
public class CadastroVeiculos {

    @Id
    @GeneratedValue(strategy =
    GenerationType.IDENTITY)
    private Long id;

    @NotBlank
    private String marca;

    @NotBlank
    private String modelo;

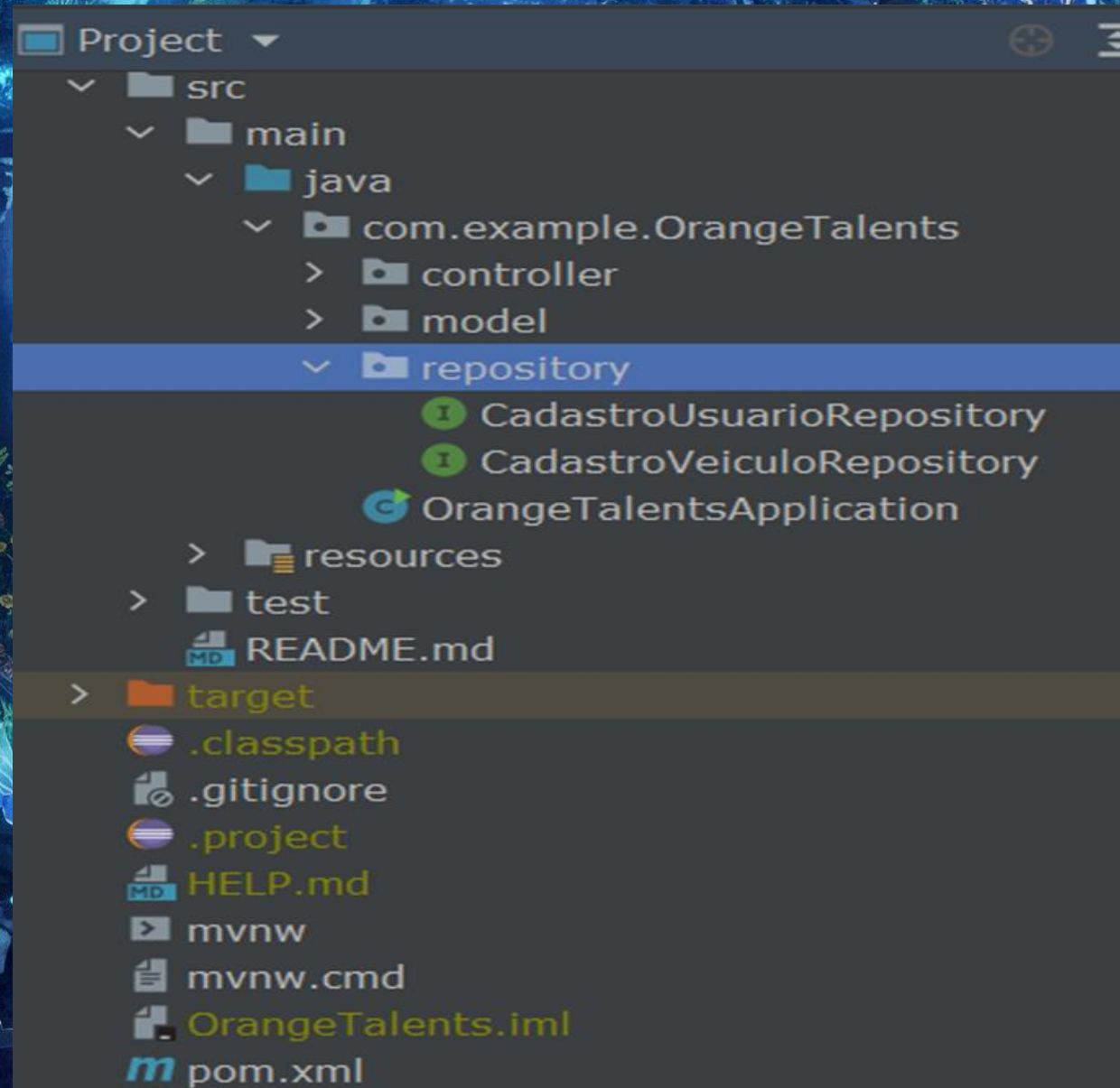
    @NotNull
    private Integer ano;

    @ManyToOne
    private CadastroUsuario usuario;
```


O Repository

É uma interface que faz o controle e a consulta de entrada e saída dos dados entre um endPoint e outro.

- CadastroUsuarioRepository
- CadastroVeiculoRepository



O Repository

No repository a extensão para a interface JpaRepository onde se faz a consulta e atualização de dados

```
package com.example.OrangeTalents.repository;
```

```
import com.example.OrangeTalents.model.CadastroUsuario;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;
```

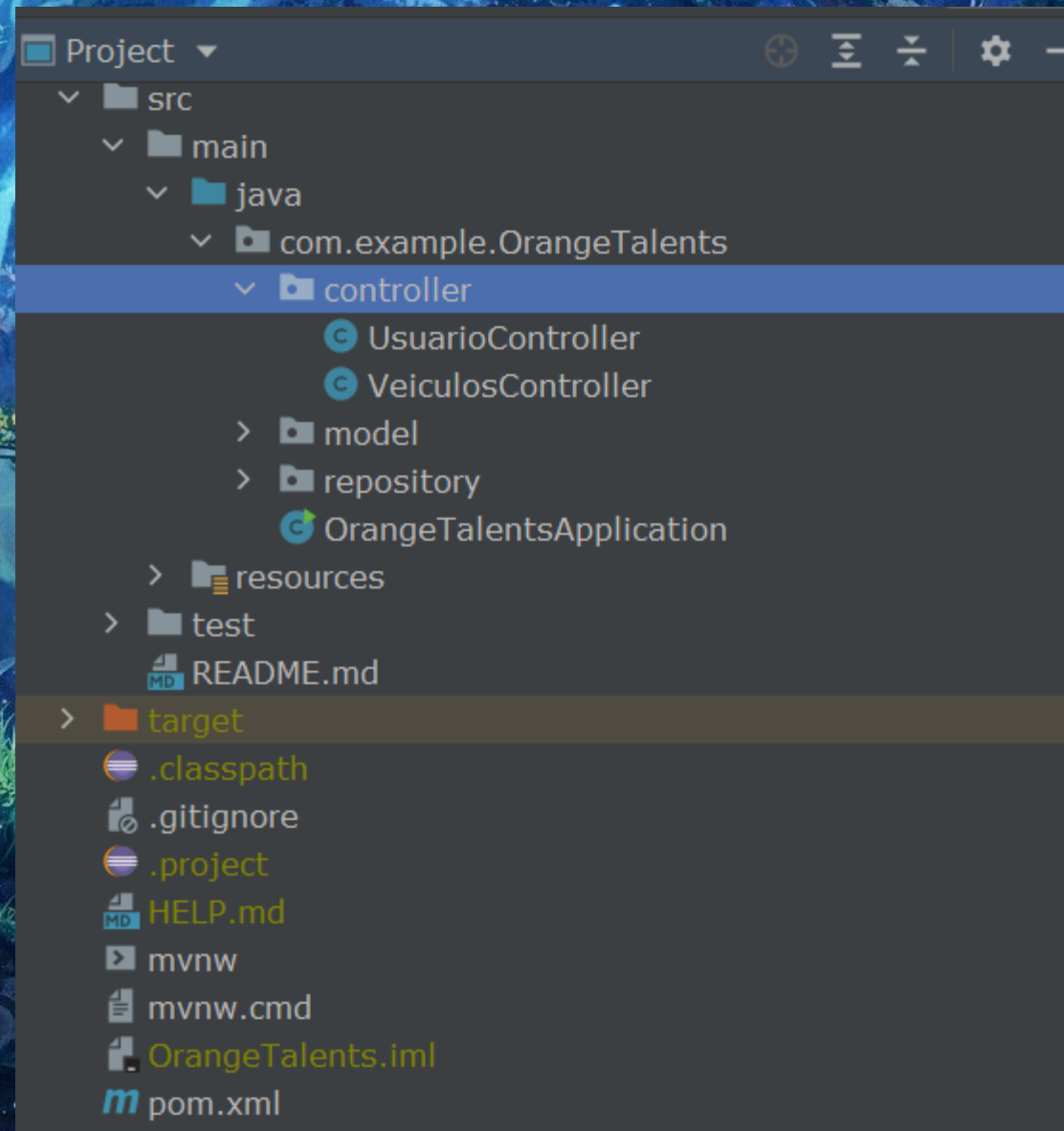
```
@Repository
```

```
public interface CadastroUsuarioRepository extends JpaRepository<CadastroUsuario, Long> {}
```


O Controller

No controller crie as seguintes classe:

- UsuarioController
- VeiculosController



O Controller

O controller é responsável pelo endPoint's/métodos de requisição HTTP GET, POST entre outros(DELETE, PUT, GET, POST).

A anotação **@RestController** informa ao spring que esta é uma classe de controller, controla as requisições de entrada e saída.

A anotação **@CrossOrigin** irá manter a segurança das informações passadas pela URI em seu corpo.

```
@RestController
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RequestMapping("/usuarios")
public class UsuarioController {

    @Autowired
    private CadastroUsuarioRepository usuarioRepository;

    @GetMapping(value = "/usuario")
    public ResponseEntity<List<CadastroUsuario>> GetAll() {
        return ResponseEntity.ok(usuarioRepository.findAll());
    }

    @GetMapping("/{id}")
    public ResponseEntity<CadastroUsuario> buscarusuarios(@PathVariable long id) {
        return usuarioRepository.findById(id).map(resp ->
        ResponseEntity.ok(resp)).orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<CadastroUsuario> cadastrousuarios(@RequestBody CadastroUsuario usuario){
        if (usuario == null){
            return ResponseEntity.badRequest().build();
        }

        usuarioRepository.save(usuario);
        return ResponseEntity.status(HttpStatus.CREATED).body(usuarioRepository.save(usuario));
    }
}
```


O Controller

@RequestMapping é usada para dar nome às URI para ser acessada, sendo que pode ser colocado também o nome da URI no endPoint

ex: **@GetMapping**(value = "/usuario")

A anotação **@Autowired** ele faz a injeção de dependência, passando a responsabilidade ao spring.

A anotação **@GetMapping** está realizando a busca das informações inseridas no banco.

A anotação **@PostMapping** está inserindo as informações do usuário no banco de dados

```
@RestController
@CrossOrigin(origins = "*", allowedHeaders = "*")
@RequestMapping("/usuarios")
public class UsuarioController {

    @Autowired
    private CadastroUsuarioRepository usuarioRepository;

    @GetMapping(value = "/usuario")
    public ResponseEntity<List<CadastroUsuario>> GetAll() {
        return ResponseEntity.ok(usuarioRepository.findAll());
    }

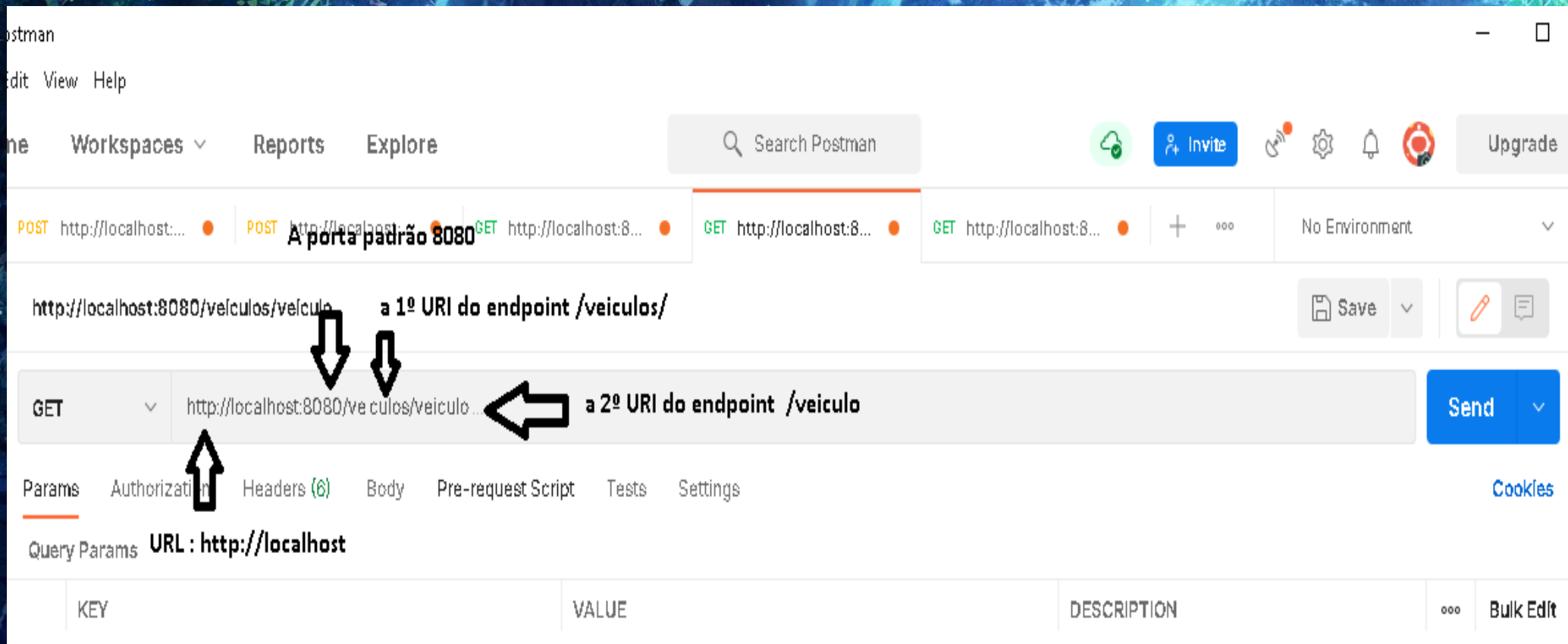
    @GetMapping("/{id}")
    public ResponseEntity<CadastroUsuario> buscarusuarios(@PathVariable long id) {
        return usuarioRepository.findById(id).map(resp ->
        ResponseEntity.ok(resp)).orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<CadastroUsuario> cadastrousuarios(@RequestBody CadastroUsuario usuario){
        if (usuario == null){
            return ResponseEntity.badRequest().build();
        }

        usuarioRepository.save(usuario);
        return ResponseEntity.status(HttpStatus.CREATED).body(usuarioRepository.save(usuario));
    }
}
```


Postman

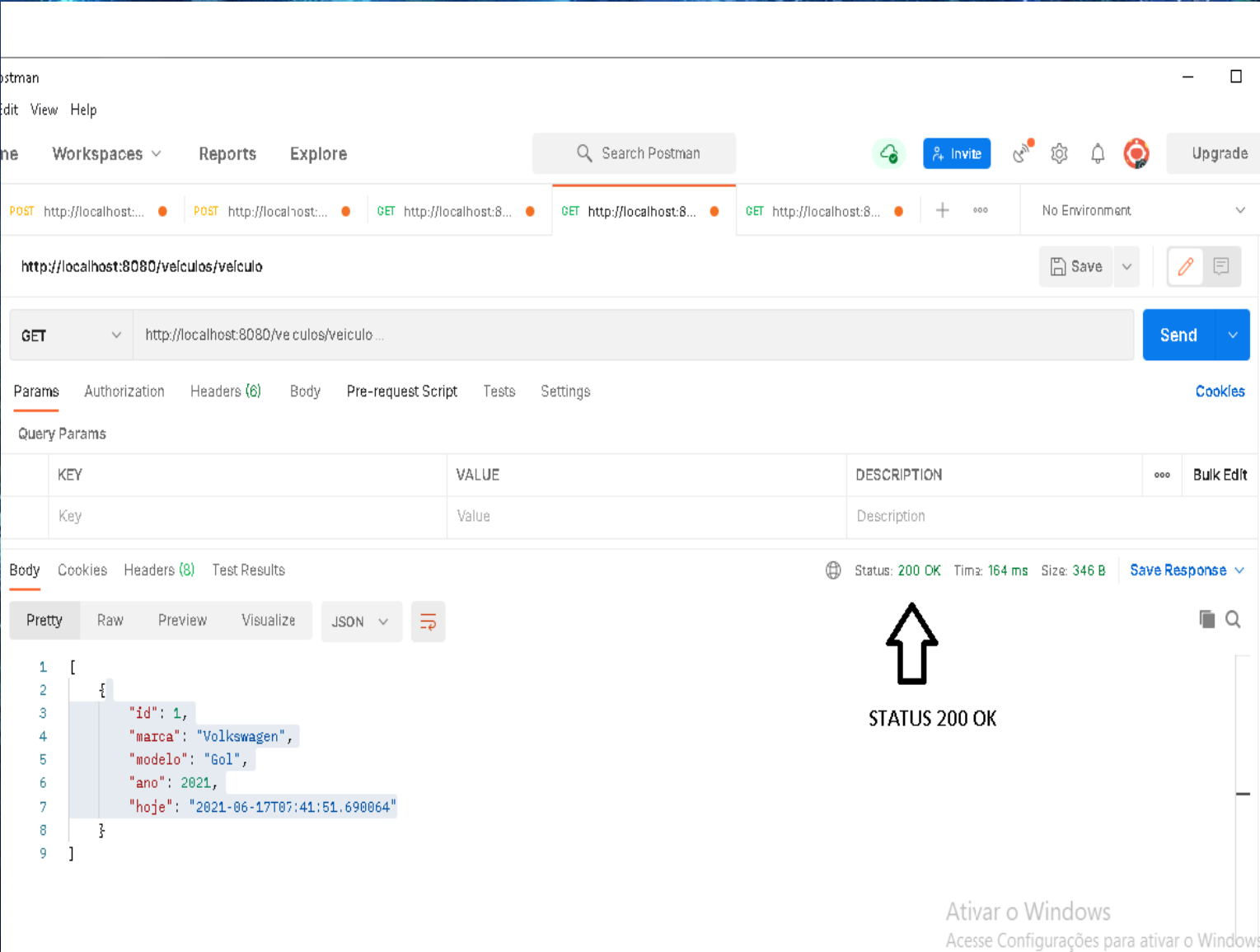
Para a consulta dos endPoint's pelo postman ou na web para passar a URL para o acesso <http://localhost:8080/veiculos/veiculo>



Postman

O postman foi utilizado para a consulta dos endPoint's Post e Get.

GET: Está trazendo os dados do banco, retornado o STATUS 200 onde houve sucesso ao trazer as informações.



The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Workspaces', 'Reports', and 'Explore' tabs. Below that, a search bar and several utility buttons like 'Invite', 'Upgrade', and 'No Environment' are visible. The main area displays a GET request to 'http://localhost:8080/veiculos/veiculo'. The 'Send' button is highlighted. Below the request, the 'Query Params' section is empty. The 'Body' tab is selected, showing a JSON response. The response status is '200 OK' with a time of '164 ms' and a size of '346 B'. A large black arrow points to the 'Status: 200 OK' text.

KEY	VALUE	DESCRIPTION
Key	Value	Description

```
[
  {
    "id": 1,
    "marca": "Volkswagen",
    "modelo": "Gol",
    "ano": 2021,
    "hoje": "2021-06-17T05:41:51.690064"
  }
]
```

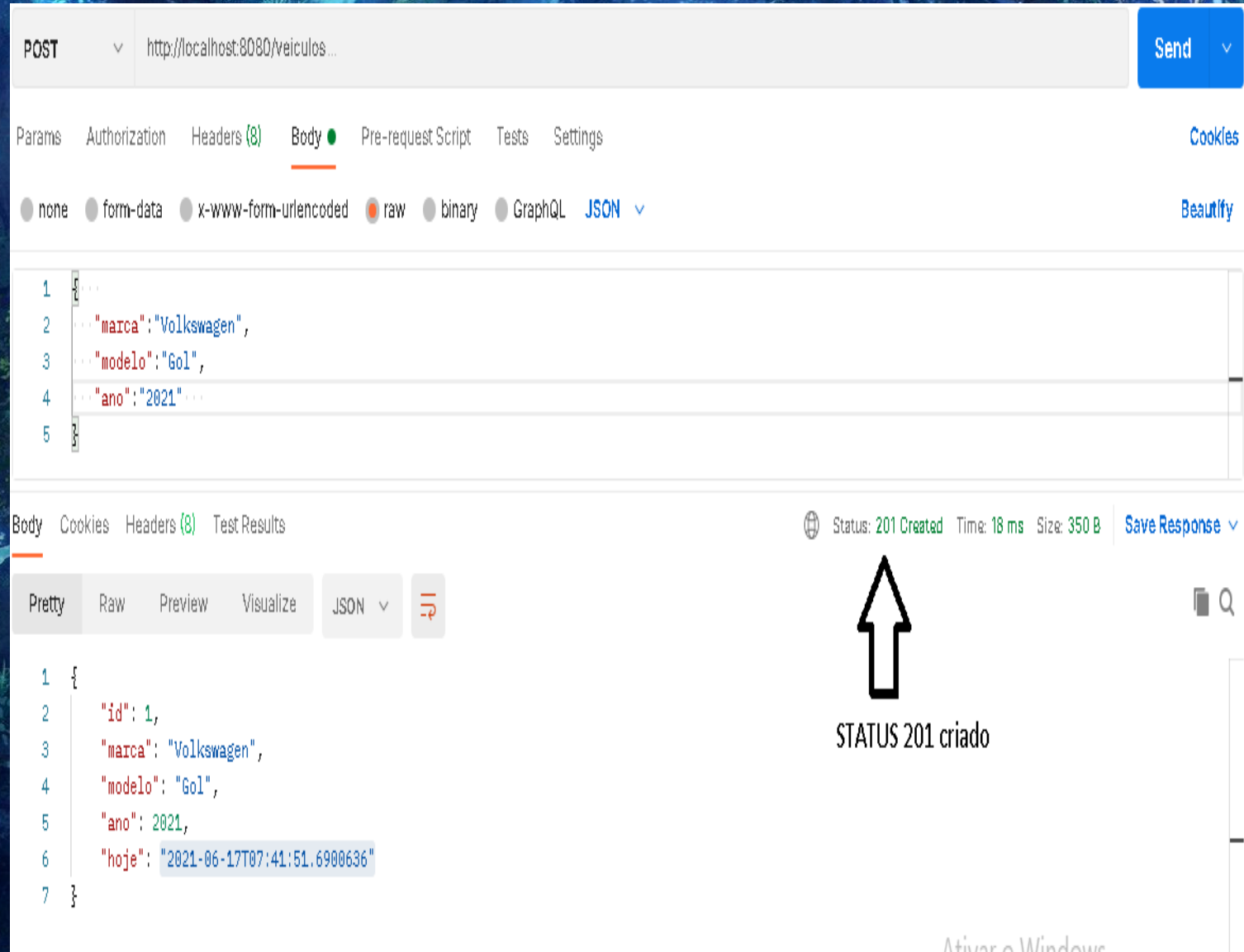
STATUS 200 OK

Ativar o Windows
Acesse Configurações para ativar o Windows

Postman

O postman foi utilizado para a consulta dos endPoint's Post e Get.

POST: Salva as informações do usuário no banco retornado status 201.



Postman

O postman foi utilizado para a consulta dos endPoint's Post e Get;

POST erro no cadastro das informações status 400.

POST `http://localhost:8080/usuarios...` [Send](#)

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings [Cookies](#) [Beautify](#)

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL [JSON](#)

```
1 {
2   "nome": "José",
3   "email": "maxxrcol@mail.com",
4   "cpf": "88439565070",
5   "dataNascimento":
6
7 }
```

Nota que não foi inserido a data por isto o erro 400.

Body Cookies Headers (7) Test Results [Status: 400 Bad Request](#) Time: 48 ms Size: 332 B [Save Response](#)

Pretty Raw Preview Visualize [JSON](#)

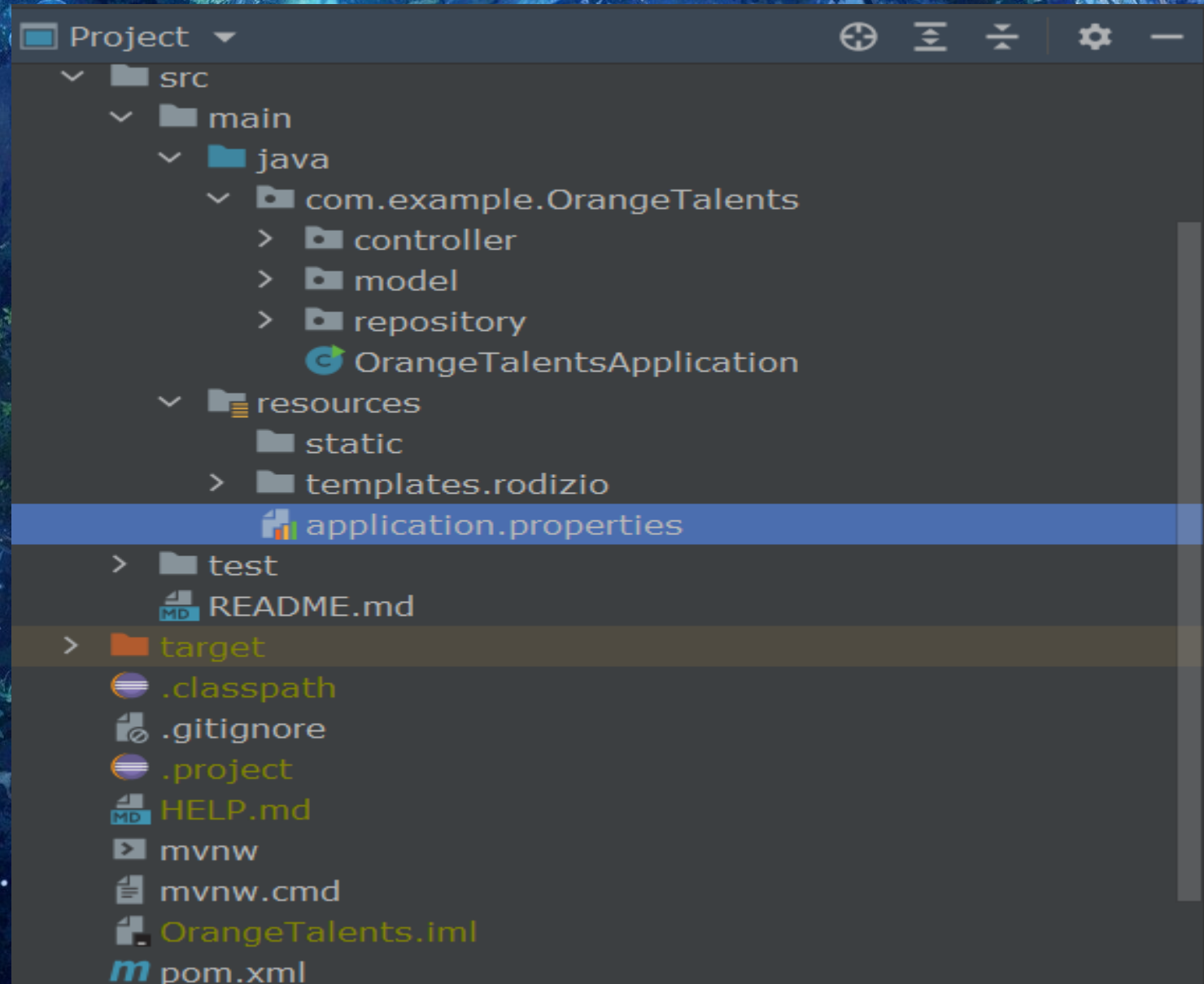
```
1 {
2   "timestamp": "2021-06-17T10:42:36.217+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "path": "/usuarios"
6 }
```

STATUS 400 ERRO NO CADASTRO

Ativar o Windows
Acesse Configurações para ativar o Windows

Application properties

Para salvar as informações no banco de dados precisa ser feito as configurações no application.properties



Application properties

Para salvar as informações no banco de dados precisa ser feito as configurações no `application.properties`, esta configurações são referente ao banco de dados para ter acesso.

datasource

```
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.url=jdbc:h2:mem:veiculo  
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
```


Banco de dados H2 configuração e acesso

Para acessar o h2 no navegador usar este link :
<http://localhost:8080/h2>

Para acessar o painel do h2 inserir a seguinte informação,

`spring.datasource.url=JDBC:H2:MEM:VEICULO`,
esta informação deve ser colocada no campo
JDBC URL no painel h2.

O password e o username referencial são estes:

`spring.datasource.username=sa`

`spring.datasource.password=`

Clicar em CONNECT para acessar.

English

Preferences Tools Help

Login

Saved Settings:

Generic H2 (Embedded)

Setting Name:

Generic H2 (Embedded)

Save

Remove

Driver Class:

org.h2.Driver

JDBC URL:

jdbc:h2:mem:veiculo

User Name:

sa

Password:

Connect

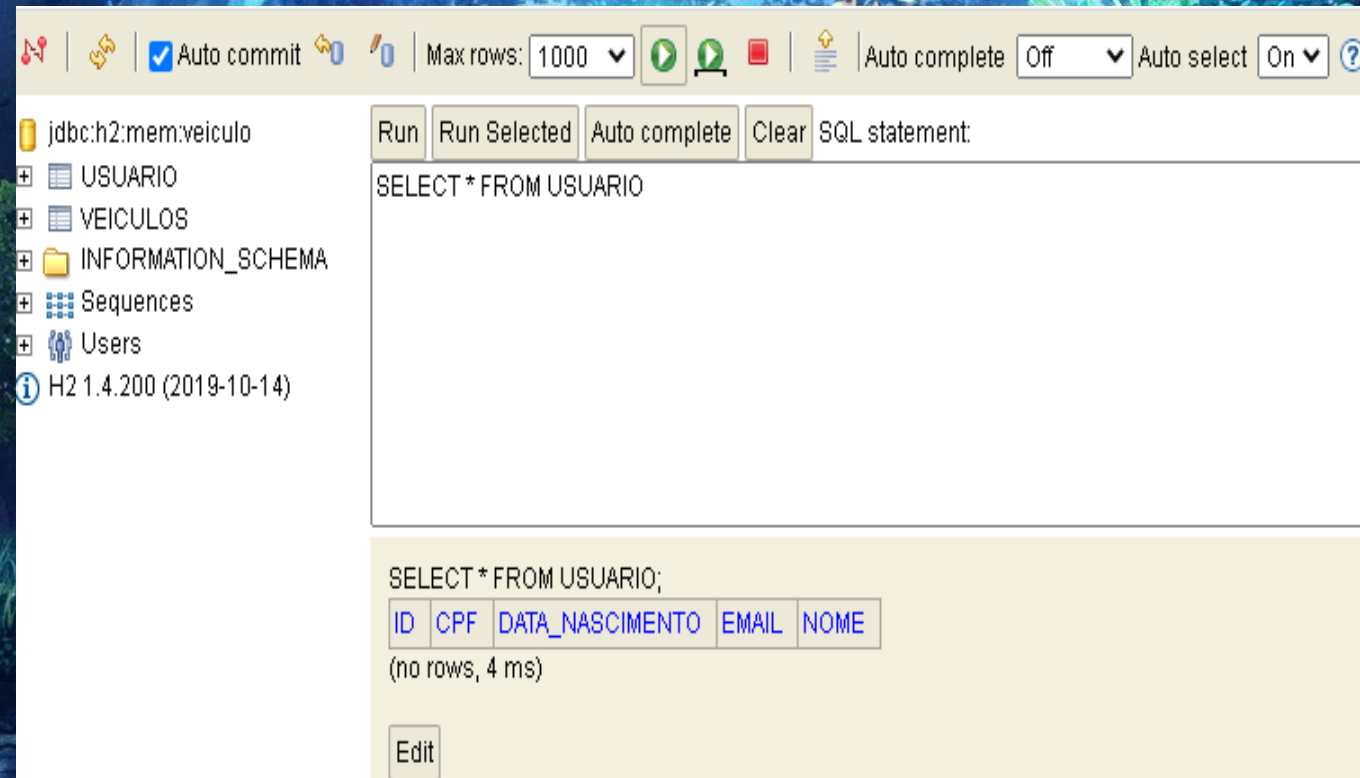
Test Connection

Banco de dados H2 configuração e acesso

Dentro do banco H2 contém as tabelas

- USUARIO
- VEICULOS

Verifica-se que não tem informações no banco.



Banco de dados H2 configuração e acesso

Dentro do banco H2 contém as tabelas

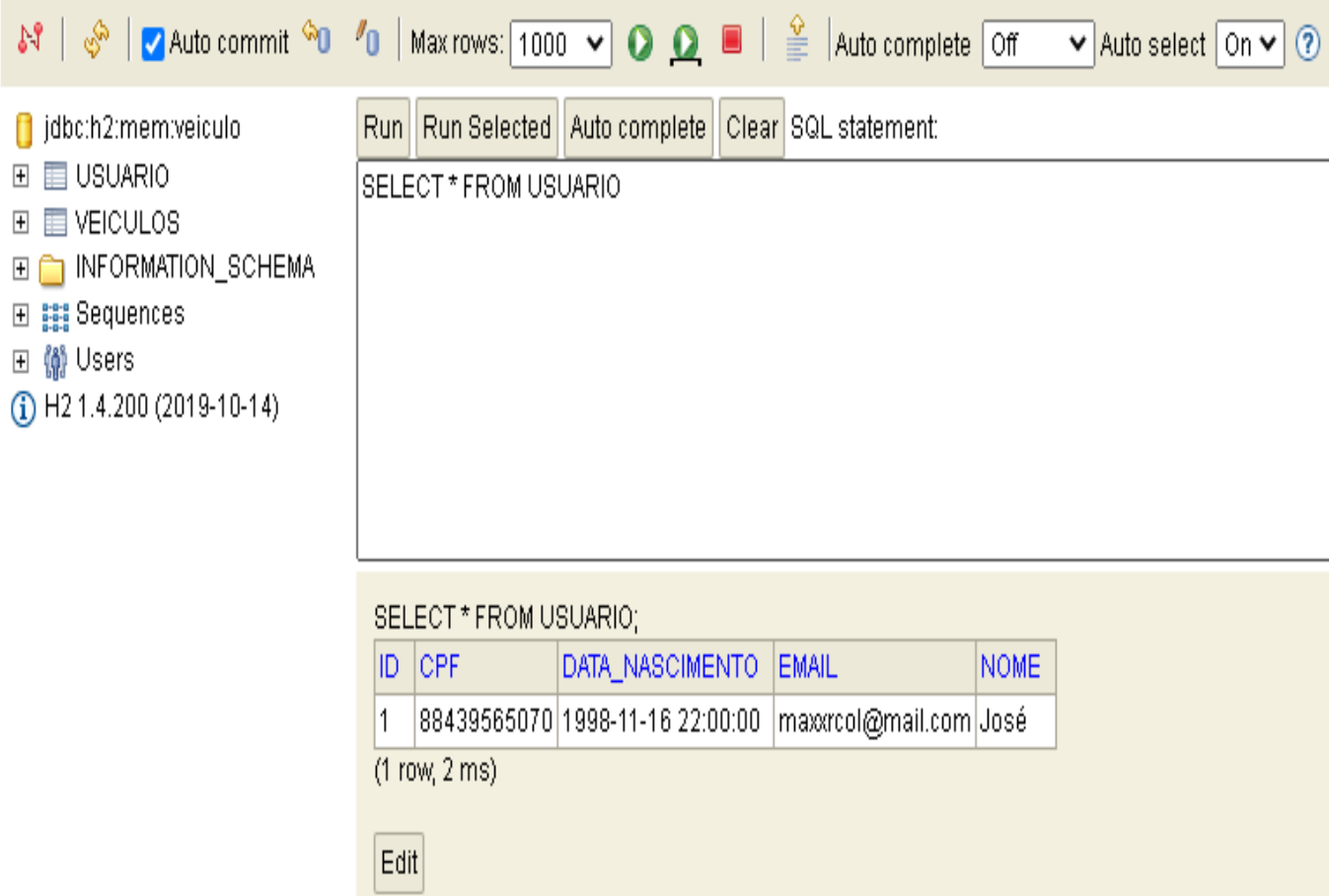
- USUARIO
- VEICULOS

Para verificar se tem informações inseridas digite o seguinte comando SQL

```
SELECT * FROM USUARIO;
```

Após clicar no BOTÃO VERDE,

Verifica-se que foi inserido as informações do usuario no banco.



The screenshot shows the H2 Database console interface. The left sidebar displays the database structure for 'jdbc:h2:mem:veiculo', including tables 'USUARIO' and 'VEICULOS', an 'INFORMATION_SCHEMA' folder, 'Sequences', and 'Users'. The main area shows the SQL statement 'SELECT * FROM USUARIO;' entered in the 'SQL statement:' field. Below the statement, the results of the query are displayed in a table format. The table has five columns: 'ID', 'CPF', 'DATA_NASCIMENTO', 'EMAIL', and 'NOME'. A single row of data is shown, representing a user with ID 1, CPF 88439565070, birth date 1998-11-16 22:00:00, email maxxrcol@mail.com, and name José. The status '(1 row, 2 ms)' is shown below the table. An 'Edit' button is located at the bottom left of the results area.

Auto commit ☒ Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:mem:veiculo

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM USUARIO
```

```
SELECT * FROM USUARIO;
```

ID	CPF	DATA_NASCIMENTO	EMAIL	NOME
1	88439565070	1998-11-16 22:00:00	maxxrcol@mail.com	José

(1 row, 2 ms)

Edit