

## Criar projeto base

**Passo 1:** Acesse o site <https://start.spring.io/>

**Passo 2:** Defina as configurações gerais do seu projeto.

Project: Maven Project

Language: Java

Spring Boot: 2.6.3

Group: com.loja

Artifact: vendas

Name: vendas

Description: Tutorial de servidor backend para recebimento de pedidos e envio automático de emails

Package name: com.loja.vendas

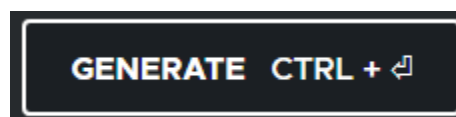
Packaging: Jar

Java: 17

**Passo 3:** Defina as dependências.

- **Lombok:** Java annotation library which helps to reduce boilerplate code.
- **Spring Boot DevTools:** Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- **Spring Web:** Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- **Spring Data JPA:** Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- **H2 Database:** 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.

**Passo 4:** Clique em GENERATE para gerar o projeto. Escolha uma pasta para salvar o arquivo .zip que será apresentado.



## Projeto no git-hub

### Criar projeto

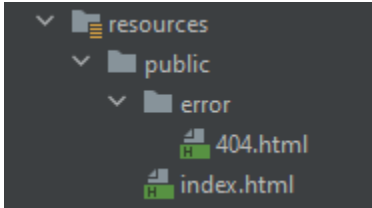
**Passo 1:** Acesse o link <https://github.com/new> e crie o projeto: desafioDioBanco

**Passo 2:** Quando você tiver seus primeiros arquivos, já pode iniciar o repositório

```
git init
git config --local user.name "Carlos Biagolini-Jr"
git config --local user.email "c.biagolini@gmail.com"
git remote add origin https://github.com/biagolini/desafioDioBanco.git
git add .
git commit -m "Base"
git branch -M main
git push -u origin main
```

## Criar páginas HTML para ter uma navegação mais agradável

**Passo 1:** Crie os pacotes(i.e. pastas) `src\main\resources\public\` + `src\main\resources\public\error\`. Dentro da primeira pasta você cria um arquivo `index.html` + `style.css`. Dentro da segunda pasta você cria um arquivo `404.html` para representar uma página a ser apresentada em caso de erro.



**Passo 2:** Desenvolva as páginas.

`src\main\resources\public\index.html`

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <title>Consulta de tópicos</title>
    <!-- STYLES-->
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <main>
      <section class="section" id="home">
        <div class="text">
          <h1>SEU BANCO</h1>
        </div>
      </section>
    </main>
  </body>
</html>
```

`src\main\resources\public\error\404.html`

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <title>Consulta de tópicos</title>
    <!-- STYLES-->
    <link rel="stylesheet" href="../../style.css" />
  </head>
  <body>
    <main>
      <section class="section" id="home">
        <div class="text">
          <h1>404: Página não encontrada</h1>
          <p>Consulte a página inicial:</p>
          <a href="http://localhost:8080"> http://localhost:8080 </a>
        </div>
      </section>
    </main>
  </body>
</html>
```

```
/* ===== VARIABLES ===== */
:root {
  --text-color: rgb(90, 90, 90);
  --link-color: rgb(200, 200, 200);
}

/* ===== BASE ===== */
a {
  text-decoration: none;
  color: var(--link-color);
}

body {
  font: 0 1rem "DM Sans", sans-serif;
  font-size: 35px;
  color: var(--text-color);
  background: var(--body-color);
  -webkit-font-smoothing: antialiased;
}

h1 {
  font-size: 60px;
  color: var(--title-color);
  margin-bottom: 1rem;
}

p {
  margin-bottom: 0rem;
}

/* ===== LAYOUT ===== */
.section {
  padding: 5rem 0;
  text-align: center;
}
```

### **Passo 3:** Postar parcial no git

```
git add .
git commit -m "Parte 01 - Paginas html: index + 404"
git push origin main
```

## Definir entidade e configuração de BD

**Passo 1:** Definir as entidades.

*src\main\java\br\com\banco\dioBank\entities\Agencia.java*

```
package br.com.banco.dioBank.entities;

import lombok.*;
import javax.persistence.*;
import java.util.List;

@Entity
@Getter
@Setter
@AllArgsConstructor
public class Agencia {

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

    private String nome;

    @OneToMany(mappedBy = "agencia")
    private List<Conta> conta;

}
```

*src\main\java\br\com\banco\dioBank\entities\Cliente.java*

```
package br.com.banco.dioBank.entities;

import lombok.*;
import javax.persistence.*;

@Entity
@Getter
@Setter
@AllArgsConstructor
public class Cliente {

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

    private String nome;

    @ManyToOne
    private Conta conta;

}
```

src\main\java\br\com\banco\dioBank\entities\Conta.java

```
package br.com.banco.dioBank.entities;

import lombok.*;
import javax.persistence.*;
import java.util.List;

@Entity
@Getter
@Setter
@AllArgsConstructor
public class Conta {

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

    private Long idTipoConta;

    @ManyToOne
    private Agencia agencia;

    @OneToMany(mappedBy = "conta")
    private List<Cliente> cliente;

}
```

src\main\java\br\com\banco\dioBank\entities\TipoConta.java

```
package br.com.banco.dioBank.entities;

import lombok.*;
import javax.persistence.*;

@Entity
@Getter
@Setter
@AllArgsConstructor
public class TipoConta {

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

    private String description;

}
```

**Passo 2:** Configurar o acesso ao seu banco de dados (endereço, usuário, senha). Em projetos com Spring Boot, quase todas as configurações ficam em um arquivo chamado `application.properties`. Se você olhar no diretório `"src/main/resources"`, você vai ver que, quando criamos o projeto, já tem esse arquivo `application.properties`, que no caso está vazio porque não precisamos configurar nada até então. Nesse arquivo, temos que colocar algumas linhas para configurar o H2 e o JPA. Você não precisa decorar nenhuma delas, basta seguir exemplos de implementação para seus projetos.

O que quer dizer cada seção

- **Datasource:** Configurações relacionadas ao acesso do seu BD. Aqui você passa o driver do seu banco de dados, URL, usuário e senha.
- **JPA:** Configurações específicas da JPA. Aqui você passa o dialeto do seu BD e informe se o Hibernate deve fazer ou não a atualização automática do BD.
- **H2:** Configurações específicas do BD H2. O `"console.enable"` indica a criação de uma interface para ser acessada no navegador. Já o `"path"` é o caminho para acessar a interface visual indicada anteriormente.

*src/main/resources/application.properties*

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

# jpa
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true

# Nova propriedade a partir da versao 2.5 do Spring Boot:
spring.jpa.defer-datasource-initialization=true
```

**Passo 3:** Insira um arquivo `.sql` para ter registros básicos no BD.

*src/main/resources/data.sql*

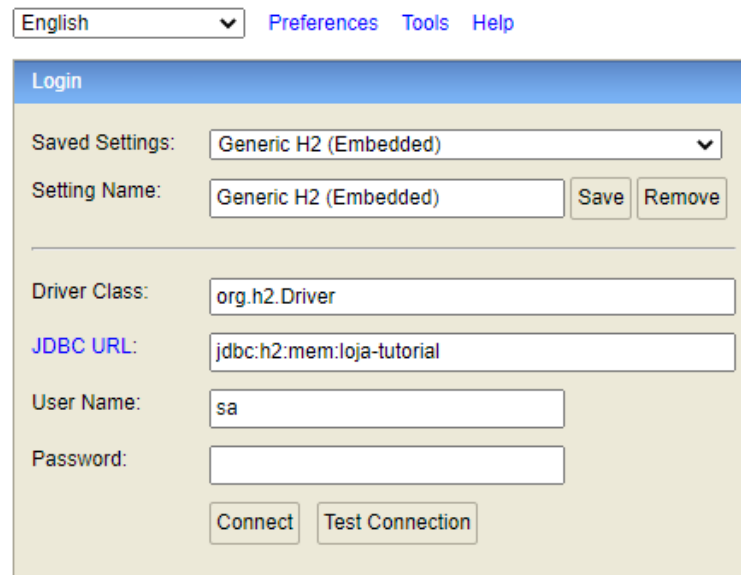
```
INSERT INTO TIPO_CONTA (description) VALUES
('Poupança'),
('Conta Corrente');

INSERT INTO AGENCIA (nome) VALUES
('Centro'),
('Zona Norte'),
('Zona Sul'),
('Zona Leste'),
('Zona Oeste');

INSERT INTO CLIENTE (nome, agencia_id) VALUES
('Joao',1),
('Mariae',2),
('Nicolau',2),
('Zidane',3);
```

```
INSERT INTO CONTA (id_Tipo_Conta, saldo, cliente_id) VALUES
(1,0,1),
(2,100,1),
(1,52.5,2),
(2,1000000,3);
```

**Passo 4:** Veja seu banco de dados de pé. Acessar o link <http://localhost:8080/h2-console/>, e indique a URL “jdbc:h2:mem:loja-tutorial”. Clique em Connect. Em seguida observe seu BD.



The screenshot shows the H2 Console Login dialog box. At the top, there is a language dropdown set to 'English' and menu links for 'Preferences', 'Tools', and 'Help'. The dialog has a 'Login' title bar. Below the title bar, there is a 'Saved Settings' section with a dropdown menu showing 'Generic H2 (Embedded)'. Underneath, the 'Setting Name' is also 'Generic H2 (Embedded)', with 'Save' and 'Remove' buttons. A horizontal line separates this from the connection configuration section. This section includes fields for 'Driver Class' (org.h2.Driver), 'JDBC URL' (jdbc:h2:mem:loja-tutorial), 'User Name' (sa), and 'Password' (empty). At the bottom, there are 'Connect' and 'Test Connection' buttons.

**Passo 5:** Postar parcial no git

```
git add .
git commit -m "Parte 02 - Entidades + BD H2"
git push origin main
```



## Criar endpoint para postar e resgatar dados do BD

**Passo 1:** Criar repositórios para manipular dados no BD.

*src/main/java/br/com/banco/dioBank/repository/ContaRepository.java*

```
package br.com.banco.dioBank.repository;

import br.com.banco.dioBank.entities.Conta;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ContaRepository extends JpaRepository<Conta, Long> {
}
```

*src/main/java/br/com/banco/dioBank/repository/TipoContaRepository.java*

```
package br.com.banco.dioBank.repository;

import br.com.banco.dioBank.entities.TipoConta;
import org.springframework.data.jpa.repository.JpaRepository;

public interface TipoContaRepository extends JpaRepository<TipoConta, Long> {
}
```

**Passo 2:** Criar classe DTO.

*src/main/java/br/com/banco/dioBank/controller/dto/ContaDto.java*

```
package br.com.banco.dioBank.controller.dto;

import br.com.banco.dioBank.entities.Conta;
import br.com.banco.dioBank.entities.TipoConta;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class ContaDto {

    private Long numeroConta;

    private Boolean contaAtiva;

    private String tipoConta;

    private String nomeCliente;

    private Long idAgencia;

    private String nomeAgencia;

    private Double saldo;

    public ContaDto(Conta conta, TipoConta tipoConta) {
```

```

        this.numeroConta = conta.getId();

        this.contaAtiva = conta.getIsActive();

        this.tipoConta = tipoConta.getDescription();

        this.nomeCliente = conta.getCliente().getNome();

        this.idAgencia = conta.getCliente().getAgencia().getId();

        this.nomeAgencia = conta.getCliente().getAgencia().getNome();

        this.saldo = conta.getSaldo();

    }
}

```

**Passo 3:** Estabelecer serviço para executar os endpoints.

*src/main/java/br/com/banco/dioBank/services/OperacoesService.java*

```

package br.com.banco.dioBank.services;

import br.com.banco.dioBank.controller.dto.ContaDto;
import br.com.banco.dioBank.entities.Conta;
import br.com.banco.dioBank.entities.TipoConta;
import br.com.banco.dioBank.repository.ContaRepository;
import br.com.banco.dioBank.repository.TipoContaRepository;
import lombok.AllArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;

@Service
@AllArgsConstructor
public class OperacoesService {

    private final ContaRepository contaRepository;

    private final TipoContaRepository tipoContaRepository;

    public ContaDto consultarSaldo(Long id) {
        Conta conta = contaRepository.findById(id).orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND));
        TipoConta tipoConta = tipoContaRepository.findById(conta.getIdTipoConta()).orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND));
        return new ContaDto(conta, tipoConta);
    }

    public void activeConta(Long id) {
        Conta conta = contaRepository.findById(id).orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND));
        conta.setIsActive(true);
        contaRepository.save(conta);
    }

    public void inactiveConta(Long id) {

```

```

        Conta conta = contaRepository.findById(id).orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND));
        conta.setIsActive(false);
        contaRepository.save(conta);
    }
}

```

**Passo 4:** Desenvolver o endpoint.

*src/main/java/br/com/banco/dioBank/controller/OperacoesController.java*

```

package br.com.banco.dioBank.controller;

import br.com.banco.dioBank.controller.dto.ContaDto;
import br.com.banco.dioBank.services.OperacoesService;
import lombok.AllArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("operacoes")
@AllArgsConstructor
public class OperacoesController {

    private final OperacoesService operacoesService;

    @GetMapping("perfil/{id}")
    public ResponseEntity<?> consultarSaldo(@PathVariable Long id) {
        ContaDto searchData = this.operacoesService.consultarSaldo(id);
        return ResponseEntity.status(HttpStatus.OK).body(searchData);
    }

    @PutMapping("status/{id}")
    public ResponseEntity<?> activeConta(@PathVariable Long id) {
        this.operacoesService.activeConta(id);
        return ResponseEntity.status(HttpStatus.OK).body("Conta ativada com sucesso");
    }

    @DeleteMapping("status/{id}")
    public ResponseEntity<?> inactiveConta(@PathVariable Long id) {
        this.operacoesService.inactiveConta(id);
        return ResponseEntity.status(HttpStatus.OK).body("Conta desativada com sucesso");
    }
}

```

**Passo 5.** Testar endpoints

Mapping	URL	Descrição
GET	http://localhost:8080/operacoes/consultaConta/{id}	Descreve o perfil de uma conta
PUT	http://localhost:8080/operacoes/status/{id}	Ativa uma conta
DELETE	http://localhost:8080/operacoes/status/{id}	Desativa uma conta

**Passo 6:** Postar parcial no git

```
git add .  
git commit -m "Parte 3 - Endpoints"  
git push origin main
```