Criando uma aplicação *back-end* com Spring 1

1 Projeto

Para a criação do projeto utilizaremos o site Spring Initializr (https://start.spring.io/), com as opções padrão, com exceção de:

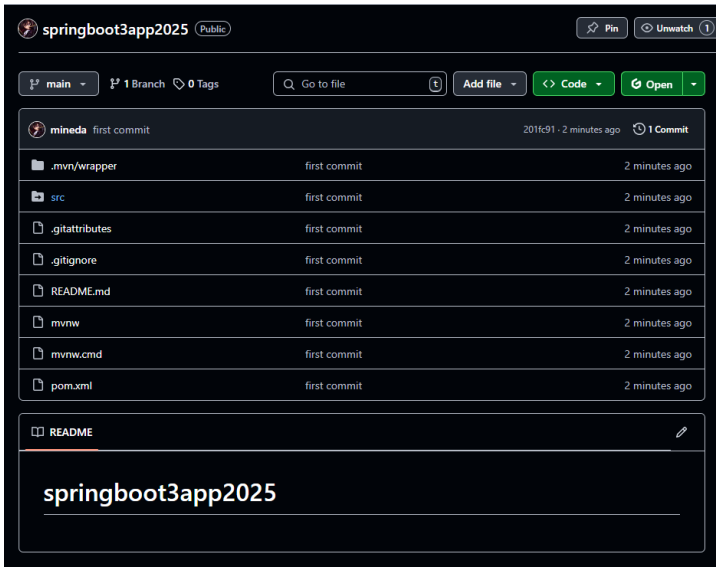
- *Project* – Escolha "Maven";
  Spring Boot – Mantenha a versão pré-selecionada;
- *Java* – Escolha "17";
- *Group* – Aqui você coloca algo único de sua organização, para tornar o identificador de seu projeto úinico. Exemplo: "br.gov.sp.fatec";
- *Artifact* – O nome do seu projeto. Exemplo: "springboot3app2025";
- *Description* – Uma descrição para seu projeto. Exemplo: "Sistema para registro de anotações";
- *Dependencies* (clique em *Add Dependencies*...) – Escolha Spring Boot DevTools, Spring Data JPA, Spring Web e o SGBD desejado (exemplos: MySQL, PostgreSQL, etc).

Pressione o botão *Generate* para baixar um arquivo compactado com o projeto. Descompacte o arquivo em uma pasta adequada (de preferência próxima à raiz do disco para evitar erros, causados por limitações do sistema operacional).
Iremos configurar o ambiente de desenvolvimento em nuvem a partir daqui. Caso tenha um ambiente Java local pule para a Seção 2.

1.1 GitHub Codespaces

O GitHub oferece um ambiente online de desenvolvimento gratuito, o Codespaces, que conta com a IDE Visual Studio Code (VSCode).
Suba o projeto para um repositório GitHub. A raiz do repositório deve conter os arquivos e diretórios apresentados na Figura 1.

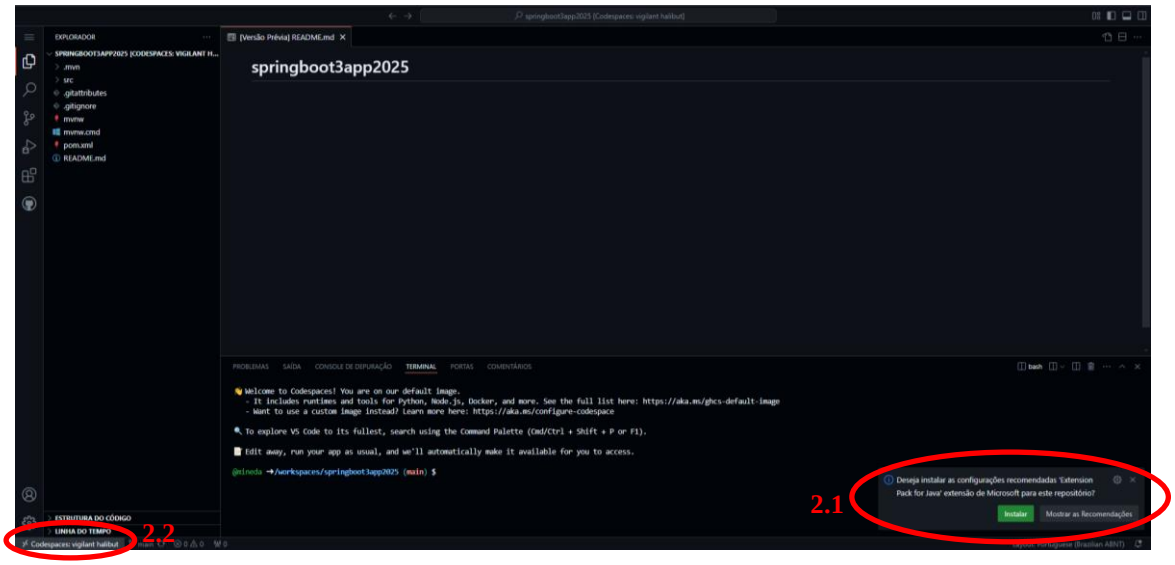
Figura 1 - Raiz do repositório Git



Para abrir seu projeto no Codespaces, pressione o botão “Code” e, na janela apresentada, pressione o botão “Create codespace on main” (o nome pode variar se sua branch principal não se chamar “main”). Nesse momento você será apresentado a um VSCode dentro do navegador, conforme a Figura 2. Instale a extensão “*Extension Pack for Java*” conforme sugerido (2.1). O tempo disponível para uso por mês é

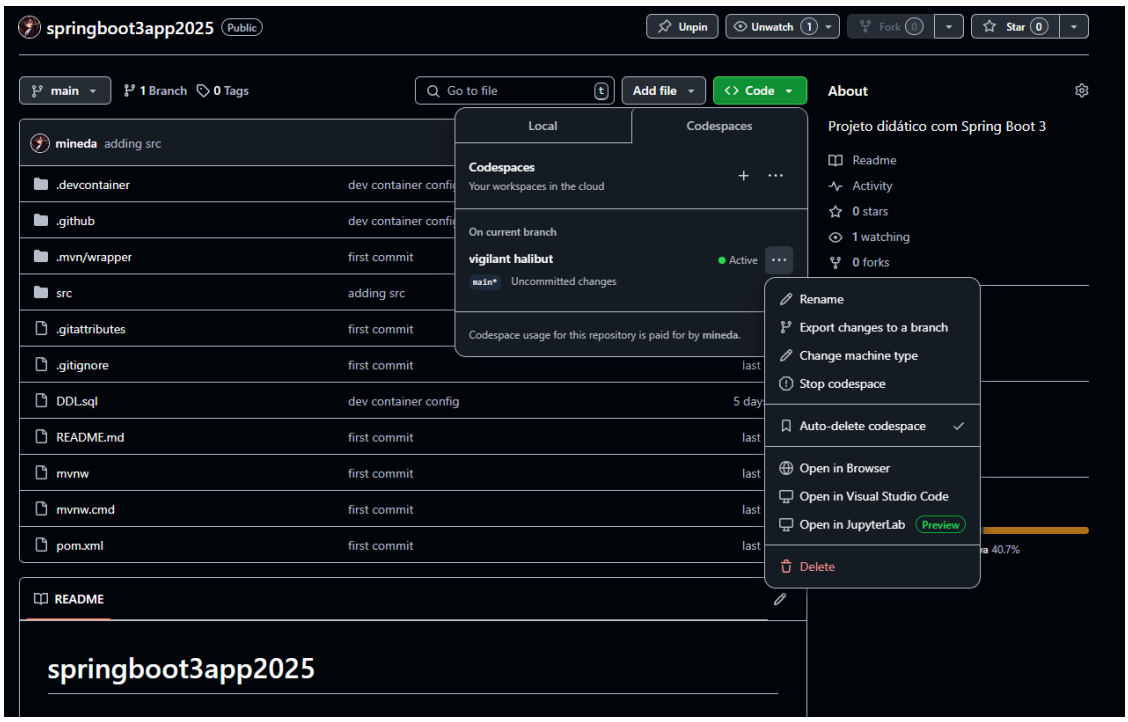
limitado, então sempre desligue o Codespace ao terminar seu trabalho, clicando na área localizada no cando inferior esquerdo (2.2) e escolhendo a opção "*Stop Current Codespace*" no menu que aparecerá na parte superior.

Figura 2 - Codespaces



Uma vez criado, seu Codespace pode ser acessado diretamente pelo seu repositório, conforme Figura 3. Note que você pode abrí-lo no navegador (como da primeira vez) ou dentro de sua instalação local do VSCode (altamente recomendável).

Figura 3 – Acessando seu Codespace



O ambiente padrão do Codespaces possui apenas o básico, como Maven e a versão 11 do Java. Uma grande vantagem é que ele também possui o Docker que nos permite utilizar o recurso de Dev Containers do VSCode para criar um ambiente de desenvolvimento personalizado.

1.2 Dev Containers

O recurso de Dev Containers do VSCode permite executar seu projeto em uma imagem Docker personalizada. Para utilizá-lo, basta acessar o menu do VSCode (atalho “ctrl+shift+p”), digitar “dev container” e selecionar a opção “Codespaces: Add Dev Container Configuration Files...”. A partir daí selecione “Create a new configuration...”. O primeiro passo consiste em selecionar um modelo e, para nosso projeto, utilizaremos “Java & PostgreSQL” (para que esse opção apareça, pode ser necessário clicar primeiro em “Show all Definitions...”). O próximo passo consiste em escolher uma imagem Linux com a versão adequada do Java (em nosso exemplo, “17-bookworn”). A seguir devemos escolher entre Maven e Gradle (como nosso projeto foi criado em Maven, usaremos ele). No último passo podemos escolher elementos adicionais, mas isso não é necessário.

Resumo:

1. Pressione “ctrl+shift+p”;
2. Digite “dev container” e selecione a opção “Codespaces: Add Dev Continer Configuration Files...”;
3. Selecione “Create a new configuration...”;
4. Clique em “Show all Definitions...”;
5. Digite “Java” e selecione “Java & PostgreSQL”;
6. Selecione a imagem Linux com a versão correta do Java, “17-bookworn”;
7. Marque a caixa de seleção do Maven e pressione “Ok”;
8. Inclua elementos adicionais, se quiser, e pressione “Ok”.

Após esse processo, uma nova pasta chamada “.devcontaine” será criada, com os seguintes arquivos:

- devcontainer.json – Configuração geral do dev container;
- docker-compose.yml – Configuração do docker compose. Aqui é possível configurar nome, usuário e senha do Banco de Dados, entre outras coisas;
- Dockerfile – Configuração da imagem utilizada para o desenvolvimento. Aqui são realizadas as instalações do Maven e dos componentes adicionais.

Para que possamos acessar nossa aplicação e o SGBD (PostgreSQL) externamento, precisamos modificar o arquivo “devcontainer.json”, acrescentando a opção “,"forwardPorts": [5432, 8080]”.
Para finalizar a configuração pressionamos “ctrl+shift+p”, digitamos “rebuild” e selecionamos a opção “Codespaces: Rebuild Container”. Na janela que aparece, clicamos em “Full Rebuild”.

2 Banco de Dados

A extensão SQLTool do VSCode é extremamente útil para interagir com um Banco de Dados (BD). Para utilizá-la é preciso instalar também a extensão correspondente ao SGBD desejado (exemplo: SQLTools PostgreSQL). Caso estejam utilizando o Codespaces e não alteraram nenhuma configuração, o banco de dados, usuário e senha são “postgres”.
Para criar as tabelas do BD, utilize as instruções SQL apresentadas no Código 2:

1. Cria uma nova tabela "usr_usuario" com colunas "usr_id", "usr_nome" e "usr_senha".
2. Cria uma nova tabela "aut_autorizacao" com colunas "aut_id" e "aut_nome";
3. Cria uma nova tabela de ligação "uau_usuario_autorizacao" com colunas "usr_id" e "aut_id";
4. Cria uma nova tabela "ant_anotacao" com colunas "ant_id", "ant_texto", "ant_data_hora" e "ant_usr_id";
5. Insere registros em todas as tabelas criadas;
6. Cria um novo usuário “spring” com senha “pass123”;
7. Dá permissão de alteração, remoção, inserção e leitura em todas as tabelas ao novo usuário.

Código 1 – DDL para criação das tabelas

```
1  create table usr_usuario (
     usr_id bigint generated always as identity,
     usr_nome varchar(20) not null,
     usr_senha varchar(150) not null,
     primary key (usr_id),
     unique (usr_nome)
   );

2  create table aut_autorizacao (
     aut_id bigint generated always as identity,
     aut_nome varchar(20) not null,
     primary key (aut_id),
     unique (aut_nome)
   );

3  create table uau_usuario_autorizacao (
     usr_id bigint not null,
     aut_id bigint not null,
     primary key (usr_id, aut_id),
     foreign key (usr_id) references usr_usuario (usr_id) on delete restrict on update
   cascade,
     foreign key (aut_id) references aut_autorizacao (aut_id) on delete restrict on
   update cascade
   );

4  create table ant_anotacao (
     ant_id bigint generated always as identity,
     ant_texto varchar(256) not null,
     ant_data_hora timestamp not null,
     ant_usr_id bigint not null,
     primary key (ant_id),
     foreign key (ant_usr_id) references usr_usuario(usr_id)
   );

5  insert into usr_usuario (usr_nome, usr_senha)
     values ('admin', '$2a$10$i3.Z8Yv1Fwl0I5SNjdCGkOTRGQjGvHjh/gMZhdc3e7LIovAklqM6C');
   insert into aut_autorizacao (aut_nome)
     values ('ROLE_ADMIN');
   insert into uau_usuario_autorizacao (usr_id, aut_id)
     values (1, 1);
   insert into ant_anotacao(ant_texto, ant_data_hora, ant_usr_id)
     values('Meu novo projeto', '2023-08-01 19:10', 1);

6  create user spring with password 'pass123';

7  grant update, delete, insert, select on all tables in schema public to spring;
```

Como passo final da configuração, precisamos configurar o acesso ao BD no Spring. Para isso, edite o arquivo "application.properties" (localizado em "src/main/resources"), incluindo a configuração apresentada em Código 4. Segue uma breve explicação dos parâmetros:

- logging.level.org.hibernate.SQL – Apresenta no Terminal todas as instruções SQL executadas pelo Spring;
- spring.datasource.url – Configuração da URL de conexão do BD (varia de acordo com o SGBD). Neste exemplo estamos conectando no **BD "posgresql"**, criado anteriormente;
- spring.datasource.username – Nome de usuário para conectar no BD;
- spring.datasource.password – Senha do usuário;
- spring.jpa.hibernate.ddl-auto – Configuração que permite a criação das tabelas do BD a partir do código Java. A configuração recomendável é "validade", para apenas validar se o código corresponde ao BD existente.

Código 2 – Configuração do Spring Boot

```
## Logging
# Show sql statement
logging.level.org.hibernate.SQL = debug
```

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url = jdbc:postgresql://localhost:5432/postgres
spring.datasource.username = spring
spring.datasource.password = pass123

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = validate
```

Execute o comando "mvn clean test" no Terminal para verificar se não há nenhum erro de configuração.

3 Mapeamento JPA

Agora precisamos mapear a tabela "usr_usuario" em uma classe Java, "Usuario.java", usando JPA. Como a tabela possui 3 colunas, a classe terá a mesma quantidade de atributos. Segue uma breve descrição das anotações utilizadas:

- @Entity – Indica que a classe mapeia uma tabela;
- @Table – Utilizada quando o nome da tabela difere do nome da classe. Aqui você indica, no parâmetro "name", o nome da tabela mapeada;
- @Column – Utilizada quando o nome do atributo difere do nome da coluna. Aqui você indica, no parâmetro "name", o nome da coluna associada ao atributo;
- @Id – Essa anotação deve ser colocada antes do atributo associado à coluna que possui a restrição de chave primária;
- @GeneratedValue – Essa anotação deve ser utilizada quando o valor do atributo é gerado automaticamente. O parâmetro "strategy" deve ser configurado com o tipo de geração utilizado ("GenerationType.IDENTITY" corresponde a "auto_increment").

Código 3 – Classe Usuario

```
package br.gov.sp.fatec. springboot3app2025.entity;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "usr_usuario")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "usr_id")
    private Long id;

    @Column(name = "usr_nome")
    private String nome;

    @Column(name = "usr_senha")
    private String senha;

    public Usuario() { }

    public Usuario(String nome, String senha) {
        this();
        this.nome = nome;
        this.senha = senha;
    }

    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 getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

}
```

As classes que mapeiam tabelas podem ficar em qualquer package, mas iremos colocá-las em "br.gov.sp.fatec.springboot3app2025.entity" na pasta "src/main/java".
Importante: Lembre-se que você deve alterar "br.gov.sp.fatec.springboot3app2025" de acordo com o que foi configurado em "Group" e "Artifact" durante a criação do projeto.

4 Spring Data JPA

Para acessar as tabelas, se faz necessária a criação de Repositórios, que contém os métodos para salvar, excluir, alterar e pesquisar dados. Cada classe de entidade precisa de um Repositório próprio. Eles serão criados na *package* " br.gov.sp.fatec.springboot3app2025.repository".
O Repositório da entidade Usuario, apresentado no Código 6, consiste em uma interface que estende JpaRepository (uma interface do Spring). O Spring gera automaticamente todo o código necessário, mas, para isso, ele precisa que você informa qual a classe de entidade associada (Usuario) e o tipo do atributo que mapeia a coluna com restrição de chave primária (Long).

Código 4 – Repositório da entidade Usuario

```
package br.gov.sp.fatec.springboot3app2025.repository;

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

import br.gov.sp.fatec.springboot3app2025.entity.Usuario;

public interface UsuarioRepository extends JpaRepository<Usuario, Long>{

}
```

5 Serviços

Repositórios não permitem programação de lógica de negócios e colocá-las nos Controllers quebra o padrão de projeto MVC. Para solucionar esse problema criaremos uma classe de serviço na package "br.gov.sp.fatec.springboot3app2025.service".
O Código 7 apresenta o código do serviço UsuarioService. A anotação @Service indica ao Spring que ele deve instanciar essa classe e gerenciá-la. Por outro lado, a anotação @Autowired carrega automaticamente o repositório criado na Seção anterior, de forma que o atributo não fique "null".

Código 5 – Serviço para gerenciamento de usuários

```
package br.gov.sp.fatec.springboot3app2025.service;

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

import org.springframework.http.HttpStatus;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;

import br.gov.sp.fatec.springboot3app2025.entity.Usuario;
import br.gov.sp.fatec.springboot3app2025.repository.UsuarioRepository;

@Service
public class UsuarioService {

    @Autowired
    private UsuarioRepository usuarioRepo;

    public Usuario buscarPorId(Long id) {
        Optional<Usuario> usuarioOp = usuarioRepo.findById(id);
        if(usuarioOp.isPresent()) {
            return usuarioOp.get();
        }
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Id inválido!");
    }

    public Usuario novoUsuario(Usuario usuario) {
        if(usuario == null ||
                usuario.getNome() == null ||
                usuario.getSenha() == null) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Nome e senha
inválidos!");
        }
        return usuarioRepo.save(usuario);
    }

    public List<Usuario> buscarTodos() {
        return usuarioRepo.findAll();
    }

}
```

Para finalizar, execute "mvn clean test" no Terminal para verificar se não há nenhum erro.