Criando uma aplicação *back-end* com Spring 2

1 Criando uma interface para o serviço

Uma boa prática de programação consiste em "programar para interfaces", portanto vamos criar uma interface para o nosso serviço "UsuarioService". Uma interface contém apenas as assinaturas dos métodos e seu código seria como o apresentado em Código 1.

Código 1 – Interface para UsuarioService

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

import java.util.List;

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

public interface IUsuarioService {

    public Usuario buscarPorId(Long id);

    public Usuario novoUsuario(Usuario usuario);

    public List<Usuario> buscarTodos();

}
```

Alteramos nosso serviço para indicar que ele implementa a interface, conforme Código 2. A partir de agora, precisaremos alterar a interface primeiro antes de incluir novos métodos.

Código 2 – Serviço alterado para uso de interface

```
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 implements IUsuarioService {

    .
    .
    .

}
```

2 Definindo Controllers REST

Continuando a partir da Seção anterior, onde finalizamos o Model do padrão MVC, seguiremos agora para o Controller.

Controllers são classes normais anotadas com @RestController. Um controller pode prover vários serviços, cada um respondendo em um URI próprio e atendendo a diversos métodos de requisição (POST, UPDATE, DELETE, GET, etc). Vamos começar com um controller com serviços voltados ao gerenciamento de usuários: UsuarioController (Código 3, na *package* br.gov.sp.fatec.springboot3app2025.controller). Reparem que aqui declaramos o serviço usando a interface criada na Seção anterior.

Uma breve explicação das novas anotações:

- @RequestMapping – Indica a URI base de todos os serviços definidos nesse Controller. Nesse caso, se estivermos rodando na máquina local com porta 8080, a URI base seria "http://localhost:8080/usuario";
- @CrossOrigin – Permite que os serviços definidos nessa classe possam ser acessados por aplicações JavaScript hospedadas em outros servidores;
- @GetMapping – Define que o método a seguir é um serviço que responde a requisições do tipo GET. Quando informato, o parâmetro "value" define o URI específico desse serviço. Seguindo o exemplo, seria "http://localhost:8080/usuario/{usuario}", onde "{usuario}" indica um parâmetro passado diretamente na URI;
- @PathVariable – Indica que o parâmetro foi passado na URI. Opcionalmente, você pode informar o nome associado ao parâmetro ("usuario" neste exemplo);
- @PostMapping – Define que o método a seguir é um serviço que responde a requisições do tipo POST;
- @RequestBody – Essa anotação acompanha o parâmetro "usuario" do método e indica que ele deve ser retirado do "corpo" da requisição. Nesse caso, os dados do usuário devem vir em um JSON com formato compatível à classe UsuarioDTO, apresentada em Código 3. A transformação do JSON para a classe ocorre automaticamente, desde que o formato seja compatível.

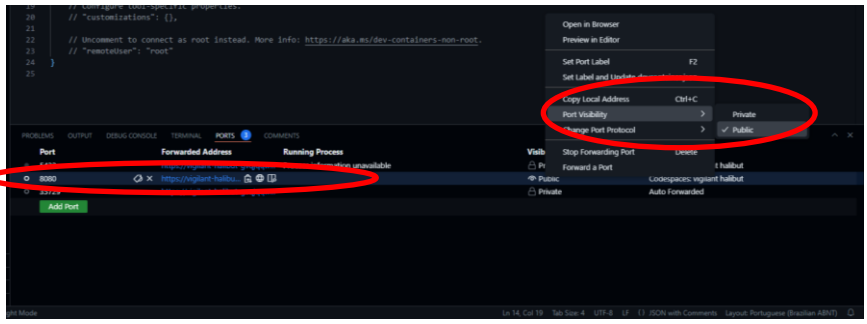
O retorno do método POST será um JSON gerado a partir da classe Usuario (retorno do método novoUsuario do serviço UsuarioService).

Código 3 – UsuarioController

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

import java.util.List;

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

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

@RestController
@CrossOrigin
@RequestMapping(value = "/usuario")
public class UsuarioController {

    @Autowired
    private IUsuarioService service;

    @GetMapping
    public List<Usuario> buscarTodos() {
        return service.buscarTodos();
    }

    @GetMapping(value = "/{usuario}")
    public Usuario buscarPorId(@PathVariable("usuario") Long id) {
        return service.buscarPorId(id);
    }

    @PostMapping
    public Usuario novoUsuario(@RequestBody Usuario usuario) {
        return service.novoUsuario(usuario);
    }

}
```

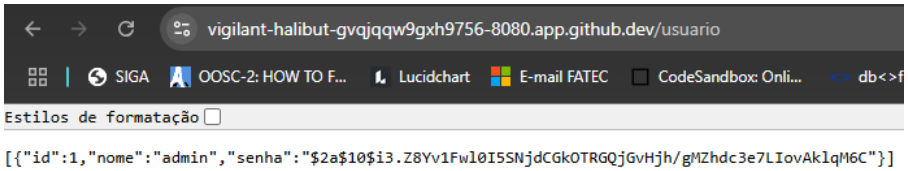
Para executar a aplicação basta executar no Terminal o comando: "mvn spring-boot:run". Verifique a aba "Ports" para descobrir a URL associada à aplicação, conforme Figura 1. Não esqueça de, antes de acessar o link, clicar com o botão direito na opção “Visibility” e alterá-la para “Public”.

Figura 1 - Ports



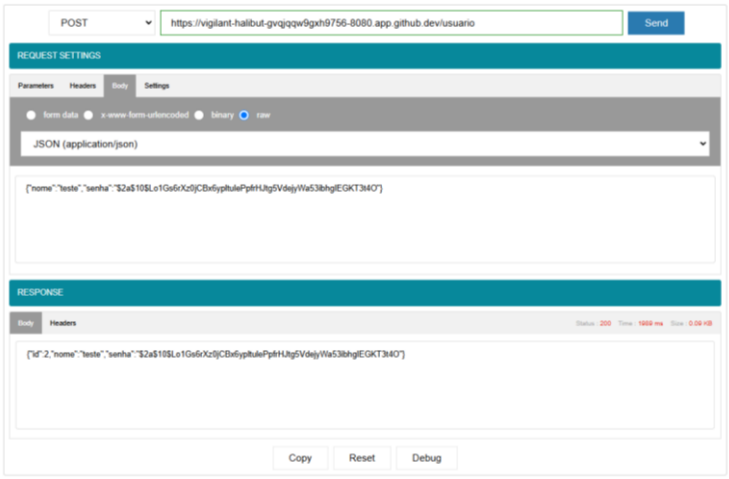
Ao acessar a URL diretamente pelo navegador, teremos um erro 404 (página não encontrada), pois não fizemos nenhum mapeamento para a URL raiz. Ao acrescentarmos "/usuario" à URL teremos acesso à página apresentada na Figura 2, que mostra o usuário que inserimos manualmente via SQL. A rota acessada corresponde ao método "buscarTodos" do controller. O navegador realiza apenas requisições do tipo GET, portanto para incluírmos um novo usuário precisaremos de uma outra ferramenta.

Figura 2 - Acessando o serviço pelo navegador



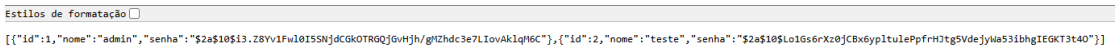
Para testar a aplicação existem diversas aplicações externas, como o Postman ou o Insomnia, além de extensões para o VS Code, como o Thunder Client. Para este exemplo, irei mostrar a utilização da aplicação externa Webtools: https://www.webtools.services/online-rest-api-client. **Importante:** Essa ferramenta não funciona se você estiver executando localmente em sua máquina. A Figura 3 apresenta uma requisição do tipo POST. Reparem que é importante informar o tipo de dado enviado no corpo da requisição, "application/json". Em outras ferramentas é preciso incluir manualmente um Header (aba Headers) do tipo "Content-type" com o valor "application/json". A resposta tem status 200 (Sucesso) e contém um JSON completo, incluindo o ID gerado automaticamente.

Figura 3 - Requisição POST



Ao atualizar a página, apresentada anteriormente na Figura 2, a lista conterá o registro recém criado. A Figura 4 apresenta o resultado após a criação do registro. A senha informada na requisição ("$2a$10$Lo1Gs6rXz0jCBx6ypltulePpfrHJtg5VdejyWa53ibhgIEGKT3t4O") corresponde à palavra "teste" com codificação BCrypt.

Figura 4 - Lista contendo o novo registro



2.1 Escondendo atributos do JSON

Uma forma simples de remover um atributo do JSON retornado consiste em usar a anotação @JsonProperty. Vamos acrescentá-la ao atributo “senha”, conforme Código 4, com acesso do tipo “somente escrita” (o atributo não aparecerá em um consulta, mas podemos informá-lo em uma inclusão ou alteração. **Importante:** a ordem das anotações não importa!

Código 4 – Classe Usuario sem Getters e Setters

```
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;

import com.fasterxml.jackson.annotation.JsonProperty;

@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")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private String senha;

    public Usuario() { }

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

    // Coloque aqui Getters e Setters
    .
    .
    .
}
```

Após a modificação, ao executarmos a aplicação teremos como resposta um JSON sem a senha, conforme apresentado na Figura 5.

Figura 5 - JSON formatado