**Executando os testes:**

Rodando a classe de teste criada no projeto:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**@Component**

Annotation que pode ser utilizada para qualquer classe auxiliar que criamos no projeto com o objetivo de executar alguma tarefa secundária que ajude o domínio a implementar uma regra de negócio.

Neste projeto, na parte da **infraestrutura**, vamos criar uma camada de componentes que possa prover essas rotinas de “apoio” para a execução das regras de negócio do domínio.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

/components/MD5Component.java

Classe “componente” para realizar a operação

de criptografia no formato MD5:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** br.com.cotiinformatica.infra.components;

**import** java.math.BigInteger;

**import** java.security.MessageDigest;

**import** java.security.NoSuchAlgorithmException;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** MD5Component {

**public** String encrypt(String value) {

MessageDigest md;

**try** {

md = MessageDigest.*getInstance*("MD5");

} **catch** (NoSuchAlgorithmException e) {

**throw** **new** RuntimeException(e);

}

BigInteger hash = **new** BigInteger

(1, md.digest(value.getBytes()));

**return** hash.toString(16);

}

}

Voltando na classe de serviço de domínio:

/services/UsuarioDomainService.java

**package** br.com.cotiinformatica.domain.services;

**import** java.util.Date;

**import** org.modelmapper.ModelMapper;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** br.com.cotiinformatica.application.dtos.GetUsuarioDTO;

**import** br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

**import** br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

**import** br.com.cotiinformatica.domain.models.Usuario;

**import** br.com.cotiinformatica.infra.components.MD5Component;

**import** br.com.cotiinformatica.infra.repositories.IUsuarioRepository;

@Service

**public** **class** UsuarioDomainService **implements** IUsuarioDomainService {

@Autowired //injeção de dependência

**private** IUsuarioRepository usuarioRepository;

**@Autowired //injeção de dependência**

**private MD5Component md5Component;**

@Override

**public** ResponseAutenticarDTO autenticar(PostAutenticarDTO dto) {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** ResponseCriarContaDTO criarConta(PostCriarContaDTO dto) {

//verificar se já existe um usuário cadastrado

//com o email informado

**if**(usuarioRepository.findByEmail(dto.getEmail()) != **null**)

**throw** **new** IllegalArgumentException("O email

informado já está cadastrado. Tente outro.");

//transferir os dados do DTO para

//a classe de modelo de entidade

ModelMapper modelMapper = **new** ModelMapper();

Usuario usuario = modelMapper.map(dto, Usuario.**class**);

**usuario.setSenha(md5Component.encrypt**

**(usuario.getSenha()));**

usuario.setDataHoraCriacao(**new** Date());

//gravando no banco de dados

usuarioRepository.save(usuario);

ResponseCriarContaDTO response

= **new** ResponseCriarContaDTO();

response.setStatus(201);

response.setMensagem("Usuário cadastrado com sucesso");

response.setDataHoraCadastro(**new** Date());

response.setUsuario(modelMapper.map

(usuario, GetUsuarioDTO.**class**));

**return** response;

}

@Override

**public** ResponseRecuperarSenhaDTO recuperarSenha

(PostRecuperarSenhaDTO dto) {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

**Para verificarmos se a modificação feita no projeto não gerou nenhum erro no cadastro do usuário, basta executarmos os testes:**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

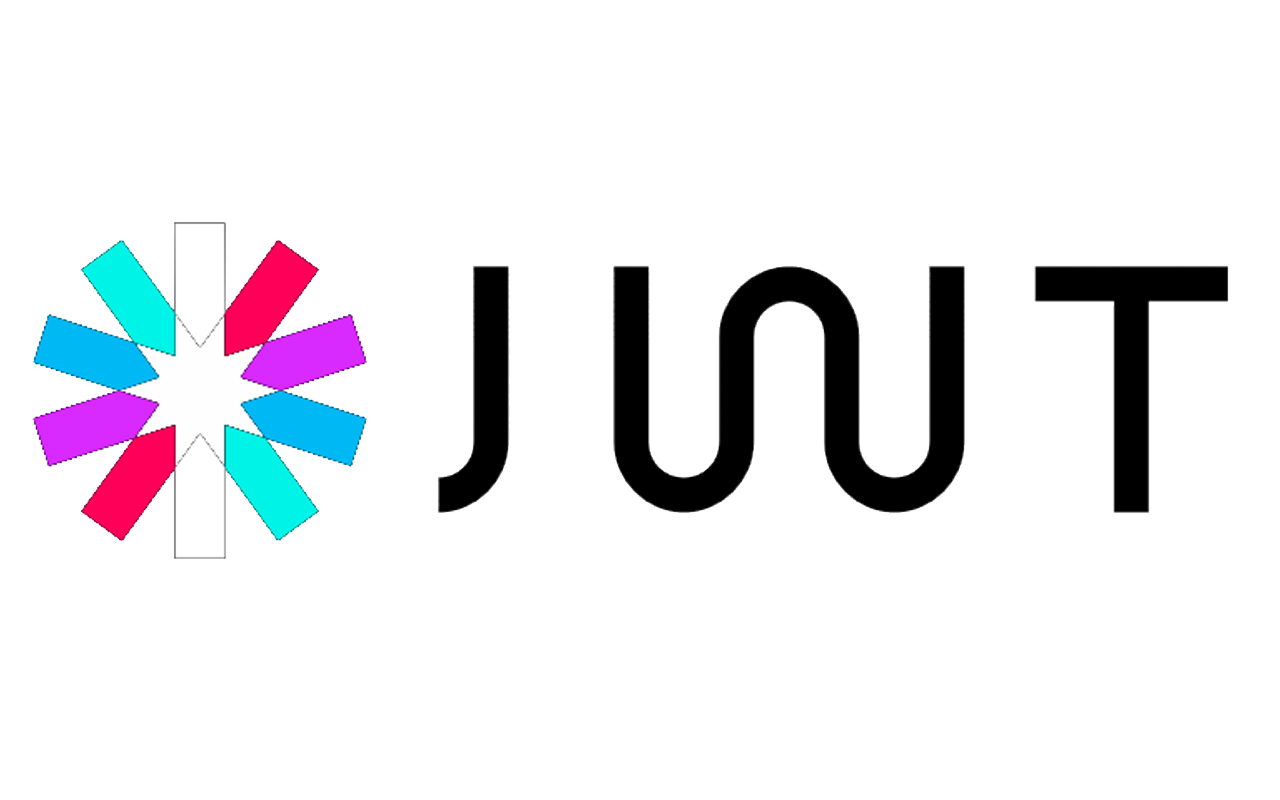
**Implementar a autenticação do usuário:**

Em API, o fluxo de autenticação será feito da seguinte forma:

1. O cliente da API deverá fazer uma requisição **POST** para o endpoint **/api/autenticar**. Enviando email e senha do usuário.
2. A API deverá validar o email e senha informados e então gerar um **TOKEN (Chave de autenticação)** e retornar este token para o cliente da API juntamente com os dados do usuário autenticado.
3. O cliente da API deverá guardar este TOKEN e então devolve-lo para a API sempre que for acessar um ENDPOINT que exija autenticação.

**JWT – JSON WEB TOKENS**

Framework para autenticação em projetos do tipo API, utilizando o conceito de identificação por meio de TOKENs.



O Spring Boot já possui suporte a biblioteca do JWT, ou seja, podemos implementar este tipo de autenticação nos nossos projetos de uma forma bem simples.

/pom.xml

<dependency>

<groupId>**org.springframework.boot**</groupId>

<artifactId>**spring-boot-starter-security**</artifactId>

</dependency>

<dependency>

<groupId>**io.jsonwebtoken**</groupId>

<artifactId>**jjwt**</artifactId>

<version>**0.2**</version>

</dependency>

<dependency>

<groupId>**org.apache.httpcomponents**</groupId>

<artifactId>**httpclient**</artifactId>

<scope>**test**</scope>

</dependency>

Criando uma classe para definir a política de autenticação do projeto (JWT) e gerar uma **CHAVE ANTIFALSIFICAÇÃO** para os TOKENS gerados pela API.

/config/JwtFilter.java

Definir a política de autenticação do projeto (JWT) e gerar a chave antifalsificação dos tokens.

**package** br.com.cotiinformatica.api.config;

**import** java.io.IOException;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** javax.servlet.FilterChain;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.authority.SimpleGrantedAuthority;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.web.filter.OncePerRequestFilter;

**import** io.jsonwebtoken.Claims;

**import** io.jsonwebtoken.Jwts;

**public** **class** JwtFilter **extends** OncePerRequestFilter {

**private** **final** String HEADER = "Authorization";

**private** **final** String PREFIX = "Bearer ";

**public** **static** **final** String ***SECRET***

= "745b7323-7704-4aa8-81b3-e1a7f1c4759e";

@Override

**protected** **void** doFilterInternal(HttpServletRequest request,

HttpServletResponse response, FilterChain chain)

**throws** ServletException, IOException {

**try** {

**if** (checkJWTToken(request, response)) {

Claims claims = validateToken(request);

**if** (claims.get("authorities") != **null**) {

setUpSpringAuthentication(claims);

} **else** {

SecurityContextHolder.*clearContext*();

}

} **else** {

SecurityContextHolder.*clearContext*();

}

chain.doFilter(request, response);

} **catch** (Exception e) {

response.setStatus

(HttpServletResponse.***SC\_UNAUTHORIZED***);

((HttpServletResponse) response).sendError

(HttpServletResponse.***SC\_UNAUTHORIZED***, e.getMessage());

**return**;

}

}

**private** Claims validateToken(HttpServletRequest request) {

String jwtToken = request.getHeader(HEADER)

.replace(PREFIX, "");

**return** Jwts.*parser*().setSigningKey(***SECRET***.getBytes())

.parseClaimsJws(jwtToken).getBody();

}

**private** **void** setUpSpringAuthentication(Claims claims) {

@SuppressWarnings({ "unchecked", "rawtypes" })

List<String> authorities = (List)

claims.get("authorities");

UsernamePasswordAuthenticationToken auth

= **new** UsernamePasswordAuthenticationToken

(claims.getSubject(), **null**,

authorities.stream().map

(SimpleGrantedAuthority::**new**)

.collect(Collectors.*toList*()));

SecurityContextHolder.*getContext*()

.setAuthentication(auth);

}

**private** **boolean** checkJWTToken(HttpServletRequest request,

HttpServletResponse res) {

String authenticationHeader = request.getHeader(HEADER);

**if** (authenticationHeader == **null**

|| !authenticationHeader.startsWith(PREFIX))

**return** **false**;

**return** **true**;

}

}

/config/JwtConfig.java

Definir as configurações necessários para o funcionamento da autenticação de usuários através do JWT.

**package** br.com.cotiinformatica.api.config;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.http.HttpMethod;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.builders.WebSecurity;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

**import** org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@SuppressWarnings("deprecation")

@Configuration

@EnableWebSecurity

**public** **class** JwtConfig **extends** WebSecurityConfigurerAdapter {

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.csrf().disable().addFilterAfter(**new** JwtFilter(),

UsernamePasswordAuthenticationFilter.**class**)

.authorizeRequests()

.antMatchers

(HttpMethod.***POST***, "/api/autenticar")

.permitAll()

.antMatchers

(HttpMethod.***POST***, "/api/criar-conta")

.permitAll()

.antMatchers

(HttpMethod.***POST***, "/api/recuperar-senha")

.permitAll()

.antMatchers(HttpMethod.***OPTIONS***, "/\*\*")

.permitAll().anyRequest().authenticated();

}

// configuração para liberar a documentação do SWAGGER

**private** **static** **final** String[] ***SWAGGER*** = { "/v2/api-docs",

"/swagger-resources", "/swagger-resources/\*\*",

"/configuration/ui", "/configuration/security",

"/swagger-ui.html", "/webjars/\*\*",

"/v3/api-docs/\*\*",

"/swagger-ui/\*\*" };

@Override

**public** **void** configure(WebSecurity web) **throws** Exception {

web.ignoring().antMatchers(***SWAGGER***);

}

}

/components/TokenComponent.java

Classe criada como um componente para

realizar a geração dos TOKENS JWT na API.

**package** br.com.cotiinformatica.infra.components;

**import** java.util.Date;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** org.springframework.security.core.GrantedAuthority;

**import** org.springframework.security.core.authority.AuthorityUtils;

**import** org.springframework.stereotype.Component;

**import** br.com.cotiinformatica.api.config.JwtFilter;

**import** io.jsonwebtoken.Jwts;

**import** io.jsonwebtoken.SignatureAlgorithm;

@Component

**public** **class** TokenComponent {

**public** String generateToken(String userName) **throws** Exception {

String secretKey = JwtFilter.***SECRET***;

List<GrantedAuthority> grantedAuthorities = AuthorityUtils

.*commaSeparatedStringToAuthorityList*("ROLE\_USER");

String token = Jwts.*builder*().setId

("api\_usuarios").setSubject(userName)

.claim("authorities",

grantedAuthorities.stream()

.map(GrantedAuthority::getAuthority).collect(Collectors.*toList*()))

.setIssuedAt

(**new** Date(System.*currentTimeMillis*()))

.setExpiration

(**new** Date(System.*currentTimeMillis*()

+ 12000000))

.signWith(SignatureAlgorithm.***HS512***,

secretKey.getBytes()).compact();

**return** token;

}

}

/services/UsuarioDomainService.java

Implementando a geração dos TOKENS JWT.

package br.com.cotiinformatica.domain.services;

import java.util.Date;

import org.modelmapper.ModelMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import br.com.cotiinformatica.application.dtos.GetUsuarioDTO;

import br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

import br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

import br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

import br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

import br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

import br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

import br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

import br.com.cotiinformatica.domain.models.Usuario;

import br.com.cotiinformatica.infra.components.MD5Component;

import br.com.cotiinformatica.infra.components.TokenComponent;

import br.com.cotiinformatica.infra.repositories.IUsuarioRepository;

@Service

public class UsuarioDomainService implements IUsuarioDomainService {

@Autowired //injeção de dependência

private IUsuarioRepository usuarioRepository;

@Autowired //injeção de dependência

private MD5Component md5Component;

@Autowired //injeção de dependência

private TokenComponent tokenComponent;

@Override

public ResponseAutenticarDTO autenticar(PostAutenticarDTO dto) {

//procurar o usuário no banco de dados através do email e da senha

Usuario usuario = usuarioRepository.findByEmailAndSenha

(dto.getEmail(), md5Component.encrypt(dto.getSenha()));

//verificar se o usuário não foi encontrado

if(usuario == null)

throw new IllegalArgumentException

("Acesso negado. Usuário inválido.");

//transferir os dados do usuário para o DTO

ModelMapper modelMapper = new ModelMapper();

GetUsuarioDTO usuarioDTO = modelMapper.map

(usuario, GetUsuarioDTO.class);

//gerando o token JWT para o usuário

String token = null;

try {

token = tokenComponent.generateToken(usuario.getEmail());

}

catch(Exception e) {

e.printStackTrace();

}

ResponseAutenticarDTO response = new ResponseAutenticarDTO();

response.setStatus(200);

response.setMensagem("Usuário autenticado com sucesso.");

response.setToken(token);

response.setUsuario(usuarioDTO);

return response;

}

@Override

public ResponseCriarContaDTO criarConta(PostCriarContaDTO dto) {

//verificar se já existe um usuário cadastrado com o email informado

if(usuarioRepository.findByEmail(dto.getEmail()) != null)

throw new IllegalArgumentException

("O email informado já está cadastrado. Tente outro.");

//transferir os dados do DTO para a classe de modelo de entidade

ModelMapper modelMapper = new ModelMapper();

Usuario usuario = modelMapper.map(dto, Usuario.class);

usuario.setSenha(md5Component.encrypt(usuario.getSenha()));

usuario.setDataHoraCriacao(new Date());

//gravando no banco de dados

usuarioRepository.save(usuario);

ResponseCriarContaDTO response = new ResponseCriarContaDTO();

response.setStatus(201);

response.setMensagem("Usuário cadastrado com sucesso");

response.setDataHoraCadastro(new Date());

response.setUsuario(modelMapper.map

(usuario, GetUsuarioDTO.class));

return response;

}

@Override

public ResponseRecuperarSenhaDTO recuperarSenha

(PostRecuperarSenhaDTO dto) {

// TODO Auto-generated method stub

return null;

}

}

Voltando no controlador, vamos implementar

o ENDPOINT da API para autenticação:

/controllers/AutenticarController.java

**package** br.com.cotiinformatica.application.controllers;

**import** javax.validation.Valid;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

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

**import** br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

**import** br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

**import** io.swagger.annotations.Api;

**import** io.swagger.annotations.ApiOperation;

@Api(tags = "Autenticação de usuários")

@RestController

**public** **class** AutenticarController {

@Autowired //injeção de dependência

**private** IUsuarioDomainService usuarioDomainService;

@ApiOperation("ENDPOINT para autenticação

de usuários e obtenção de Token.")

@PostMapping("/api/autenticar")

**public** ResponseEntity<ResponseAutenticarDTO>

post(@Valid @RequestBody PostAutenticarDTO dto) {

ResponseAutenticarDTO response = **null**;

**try** {

response = usuarioDomainService.autenticar(dto);

**return** ResponseEntity.*status*

(HttpStatus.***OK***).body(response);

}

**catch**(IllegalArgumentException e) {

response = **new** ResponseAutenticarDTO();

response.setStatus(401);

//UNAUTHORIZED (CLIENT ERROR)

response.setMensagem(e.getMessage());

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***)

.body(response);

}

**catch**(Exception e) {

response = **new** ResponseAutenticarDTO();

response.setStatus(500); //INTERNAL SERVER ERROR

response.setMensagem(e.getMessage());

**return** ResponseEntity.*status*

(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(response);

}

}

}

**Executando o projeto:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Tabela, Excel

Descrição gerada automaticamente

<http://localhost:8083/swagger-ui/index.html>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Testando através do POSTMAN:**

Primeiro, vamos criar um usuário:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Cadastrando o usuário:**

**POST** http://localhost:8083/api/criar-conta

REQUEST BODY:

**{**

**"email": "sergio.coti@bol.com",**

**"nome": "Sergio Mendes",**

**"senha": "@Admin123"**

**}**

RESPONSE:

**{**

**"status": 201,**

**"mensagem": "Usuário cadastrado com sucesso",**

**"usuario": {**

**"idUsuario": 7,**

**"nome": "Sergio Mendes",**

**"email": "sergio.coti@bol.com"**

**},**

**"dataHoraCadastro": 1678490777312**

**}**

**Testando o serviço de autenticação:**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Autenticando o usuário:**

**POST** http://localhost:8083/api/autenticar

REQUEST BODY:

**{**

**"email": "sergio.coti@bol.com",**

**"senha": "@Admin123"**

**}**

RESPONSE:

**{**

**"status": 200,**

**"mensagem": "Usuário autenticado com sucesso.",**

**"usuario": {**

**"idUsuario": 7,**

**"nome": "Sergio Mendes",**

**"email": "sergio.coti@bol.com"**

**},**

**"token": "eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiJhcGlfdXN1YXJp**

**b3MiLCJzdWIiOiJzZXJnaW8uY290aUBib2wuY29tIiwiYXV0aG9yaXRpZXMiOlsiUk9MRV9VU0VSIl0sImlhdCI6MTY3ODQ5MTQzOCwiZXhwIjoxNjc4NTAzNDM4fQ.5E0Vaqid1r4Uok7jNQP\_mZJB0sm\_4rrrl1MTFiaMXU7Cba2PVENMGbl3qNDxB7N6zKAkyDC6hGJDG6hJg\_YT9g"**

**}**

**Validando o TOKEN JWT gerado:**

<https://jwt.io/>

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Site

Descrição gerada automaticamente

**Voltando na camada de testes:**

TDD – TEST DRIVEN DEVELOPMENT

package br.com.cotiinformatica;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.util.Locale;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.web.servlet.MockMvc;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.github.javafaker.Faker;

import br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

import br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

import br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

@SpringBootTest

@AutoConfigureMockMvc

class ApiUsuariosApplicationTests {

@Autowired

private MockMvc mock;

@Autowired

private ObjectMapper objectMapper;

private static PostCriarContaDTO postCriarContaDTO;

@SuppressWarnings("static-access")

@Test

public void postCriarContaTest() throws Exception {

PostCriarContaDTO dto = new PostCriarContaDTO();

Faker faker = new Faker(new Locale("pt-BR"));

dto.setNome("Usuário Testador");

dto.setEmail(faker.internet().emailAddress());

dto.setSenha("@Teste1234");

mock.perform(

post("/api/criar-conta")

.contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isCreated());

this.postCriarContaDTO = dto;

}

@Test

public void postAutenticarTest() throws Exception {

PostAutenticarDTO dto = new PostAutenticarDTO();

dto.setEmail(postCriarContaDTO.getEmail());

dto.setSenha(postCriarContaDTO.getSenha());

mock.perform(

post("/api/autenticar")

.contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isOk());

}

@Test

public void postRecuperarSenhaTest() throws Exception {

PostRecuperarSenhaDTO dto = new PostRecuperarSenhaDTO();

dto.setEmail(postCriarContaDTO.getEmail());

mock.perform(

post("/api/recuperar-senha")

.contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isOk());

}

}

**Executando os testes:**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente

**Configurando uma biblioteca no Spring Boot   
para realizar o envio de emails:**

/pom.xml

<dependency>

<groupId>**org.springframework.boot**</groupId>

<artifactId>**spring-boot-starter-mail**</artifactId>

</dependency>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Criando uma configuração para fazermos envio de email automático no projeto.**

/application.properties

server.port=8083

spring.datasource.url=jdbc:postgresql://localhost:5432/bd\_apiusuarios

spring.datasource.driver-class-name=org.postgresql.Driver

spring.datasource.username=postgres

spring.datasource.password=coti

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

**spring.mail.host=smtp-mail.outlook.com**

**spring.mail.port=587**

**spring.mail.username=cotiaulajava@outlook.com**

**spring.mail.password=@Admin123456**

**spring.mail.properties.mail.smtp.auth=true**

**spring.mail.properties.mail.smtp.starttls.enable=true**

Criando uma classe “Component” para implementar o envio de emails:

/infra/components/EmailComponent.java

**package** br.com.cotiinformatica.infra.components;

**import** javax.mail.internet.MimeMessage;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.mail.javamail.JavaMailSender;

**import** org.springframework.mail.javamail.MimeMessageHelper;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** EmailComponent {

@Autowired

**private** JavaMailSender javaMailSender;

@Value("${spring.mail.username}")

**private** String userName;

**public** **void** sendMessage

(String to, String subject, String body) {

**try** {

MimeMessage message

= javaMailSender.createMimeMessage();

MimeMessageHelper helper

= **new** MimeMessageHelper(message);

helper.setFrom(userName);

helper.setTo(to);

helper.setSubject(subject);

helper.setText(body, **true**);

javaMailSender.send(message);

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

Voltando na camada de domínio:

/services/UsuarioDomainService.java

package br.com.cotiinformatica.domain.services;

import java.util.Date;

import org.modelmapper.ModelMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.github.javafaker.Faker;

import br.com.cotiinformatica.application.dtos.GetUsuarioDTO;

import br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

import br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

import br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

import br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

import br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

import br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

import br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

import br.com.cotiinformatica.domain.models.Usuario;

import br.com.cotiinformatica.infra.components.EmailComponent;

import br.com.cotiinformatica.infra.components.MD5Component;

import br.com.cotiinformatica.infra.components.TokenComponent;

import br.com.cotiinformatica.infra.repositories.IUsuarioRepository;

@Service

public class UsuarioDomainService implements IUsuarioDomainService {

@Autowired //injeção de dependência

private IUsuarioRepository usuarioRepository;

@Autowired //injeção de dependência

private MD5Component md5Component;

@Autowired //injeção de dependência

private TokenComponent tokenComponent;

@Autowired //injeção de dependência

private EmailComponent emailComponent;

@Override

public ResponseAutenticarDTO autenticar(PostAutenticarDTO dto) {

//procurar o usuário no banco de dados através do email e da senha

Usuario usuario = usuarioRepository.findByEmailAndSenha(dto.getEmail(),

md5Component.encrypt(dto.getSenha()));

//verificar se o usuário não foi encontrado

if(usuario == null)

throw new IllegalArgumentException

("Acesso negado. Usuário inválido.");

//transferir os dados do usuário para o DTO

ModelMapper modelMapper = new ModelMapper();

GetUsuarioDTO usuarioDTO

= modelMapper.map(usuario, GetUsuarioDTO.class);

//gerando o token JWT para o usuário

String token = null;

try {

token = tokenComponent.generateToken(usuario.getEmail());

}

catch(Exception e) {

e.printStackTrace();

}

ResponseAutenticarDTO response = new ResponseAutenticarDTO();

response.setStatus(200);

response.setMensagem("Usuário autenticado com sucesso.");

response.setToken(token);

response.setUsuario(usuarioDTO);

return response;

}

@Override

public ResponseCriarContaDTO criarConta(PostCriarContaDTO dto) {

//verificar se já existe um usuário cadastrado com o email informado

if(usuarioRepository.findByEmail(dto.getEmail()) != null)

throw new IllegalArgumentException

("O email informado já está cadastrado. Tente outro.");

//transferir os dados do DTO para a classe de modelo de entidade

ModelMapper modelMapper = new ModelMapper();

Usuario usuario = modelMapper.map(dto, Usuario.class);

usuario.setSenha(md5Component.encrypt(usuario.getSenha()));

usuario.setDataHoraCriacao(new Date());

//gravando no banco de dados

usuarioRepository.save(usuario);

ResponseCriarContaDTO response = new ResponseCriarContaDTO();

response.setStatus(201);

response.setMensagem("Usuário cadastrado com sucesso");

response.setDataHoraCadastro(new Date());

response.setUsuario(modelMapper.map(usuario, GetUsuarioDTO.class));

return response;

}

@Override

public ResponseRecuperarSenhaDTO recuperarSenha(PostRecuperarSenhaDTO dto) {

//procurar o usuário no banco de dados através do email

Usuario usuario = usuarioRepository.findByEmail(dto.getEmail());

//verificar se o usuário foi encontrado

if(usuario == null)

throw new IllegalArgumentException

("Email não encontrado. Usuário inválido.");

//gerar uma nova senha para o usuário

String novaSenha = new Faker().internet().password(8, 20, true);

//escrevendo o email para o usuário

String subject = "Recuperação de senha - API Usuários";

String body = "<div>"

+ "<p>Olá, " + usuario.getNome() + "</p>"

+ "<p>Uma nova senha foi gerada com sucesso!</p>"

+ "<p>Acesse o sistema com a senha: <strong>" + novaSenha

+ "</strong></p>"

+ "<p>Att,</p>"

+ "<p>Equipe API Usuários</p>";

//enviando o email

emailComponent.sendMessage(usuario.getEmail(), subject, body);

//atualizando a senha do usuário no banco de dados

usuario.setSenha(md5Component.encrypt(novaSenha));

usuarioRepository.save(usuario);

ResponseRecuperarSenhaDTO response

= new ResponseRecuperarSenhaDTO();

response.setStatus(200);

response.setMensagem("Recuperação de senha realizado com sucesso.");

return response;

}

}

Voltando no controlador:

/controllers/RecuperarSenhaController.java

**package** br.com.cotiinformatica.application.controllers;

**import** javax.validation.Valid;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

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

**import** br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

**import** br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

**import** br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

**import** io.swagger.annotations.Api;

**import** io.swagger.annotations.ApiOperation;

@Api(tags = "Recuperação de senha")

@RestController

**public** **class** RecuperarSenhaController {

@Autowired

**private** IUsuarioDomainService usuarioDomainService;

@ApiOperation("ENDPOINT para recuperação da senha

de acesso do usuário.")

@PostMapping("/api/recuperar-senha")

**public** ResponseEntity<ResponseRecuperarSenhaDTO> post

(@Valid @RequestBody PostRecuperarSenhaDTO dto) {

ResponseRecuperarSenhaDTO response = **null**;

**try** {

response = usuarioDomainService.recuperarSenha(dto);

**return** ResponseEntity.*status*(HttpStatus.***OK***)

.body(response);

}

**catch**(IllegalArgumentException e) {

response = **new** ResponseRecuperarSenhaDTO();

response.setStatus(400);

//BAD REQUEST (CLIENT ERROR)

response.setMensagem(e.getMessage());

**return** ResponseEntity.*status*

(HttpStatus.***BAD\_REQUEST***).body(response);

}

**catch**(Exception e) {

response = **new** ResponseRecuperarSenhaDTO();

response.setStatus(500); //INTERNAL SERVER ERROR

response.setMensagem(e.getMessage());

**return** ResponseEntity.*status*

(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(response);

}

}

}

**Executando os testes:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Classe de testes do Spring Boot**

package br.com.cotiinformatica;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.util.Locale;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.web.servlet.MockMvc;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.github.javafaker.Faker;

import br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

import br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

import br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

@SpringBootTest

@AutoConfigureMockMvc

class ApiUsuariosApplicationTests {

@Autowired

private MockMvc mock;

@Autowired

private ObjectMapper objectMapper;

@Test

public void postCriarContaTest() throws Exception {

PostCriarContaDTO dto = new PostCriarContaDTO();

Faker faker = new Faker(new Locale("pt-BR"));

dto.setNome("Usuário Testador");

dto.setEmail(faker.internet().emailAddress());

dto.setSenha("@Teste1234");

mock.perform(

post("/api/criar-conta").contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isCreated());

}

@Test

public void postAutenticarTest() throws Exception {

PostCriarContaDTO postCriarContaDTO = new PostCriarContaDTO();

Faker faker = new Faker(new Locale("pt-BR"));

postCriarContaDTO.setNome("Usuário Testador");

postCriarContaDTO.setEmail(faker.internet().emailAddress());

postCriarContaDTO.setSenha("@Teste1234");

mock.perform(post("/api/criar-conta").contentType("application/json")

.content(objectMapper.writeValueAsString

(postCriarContaDTO))).andExpect(status().isCreated());

PostAutenticarDTO dto = new PostAutenticarDTO();

dto.setEmail(postCriarContaDTO.getEmail());

dto.setSenha(postCriarContaDTO.getSenha());

mock.perform(

post("/api/autenticar").contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isOk());

}

@Test

public void postRecuperarSenhaTest() throws Exception {

PostCriarContaDTO postCriarContaDTO = new PostCriarContaDTO();

Faker faker = new Faker(new Locale("pt-BR"));

postCriarContaDTO.setNome("Usuário Testador");

postCriarContaDTO.setEmail(faker.internet().emailAddress());

postCriarContaDTO.setSenha("@Teste1234");

mock.perform(post("/api/criar-conta").contentType("application/json")

.content(objectMapper.

writeValueAsString(postCriarContaDTO)))

.andExpect(status().isCreated());

PostRecuperarSenhaDTO dto = new PostRecuperarSenhaDTO();

dto.setEmail(postCriarContaDTO.getEmail());

mock.perform(post("/api/recuperar-senha").contentType("application/json")

.content(objectMapper.writeValueAsString(dto)))

.andExpect(status().isOk());

}

}

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Testando no POSTMAN:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Tabela

Descrição gerada automaticamente

<http://localhost:8083/swagger-ui/index.html>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**No POSTMAN:**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente