

## Sumário

1.	Site para Criar o projeto - Spring Initializr	2
2.	Dependências do Projeto	2
3.	Arquivo pom.xml	2
4.	Criar UsuarioController	3
5.	Criar Entidade Usuario	3
6.	Criar UsuarioRepository	4
7.	Criar arquivos de configuração de conexão com banco de dados	4
7.1.	application.properties	4
7.2.	application-test.properties	4
7.3.	application-dev.properties	4
8.	Testar adicionar um usuário	6
9.	Atualizar UsuarioController Teste – Mais simples	6
10.	Criar UsuarioDTO	6
11.	Criar UsuarioService	6
12.	Atualizar UsuarioController – Correção padrão correto	7
13.	Atualizar UsuarioController – Correção retornar Pageable	7
14.	Atualizar UsuarioService – Correção retornar Pageable	7
15.	Atualizar UsuarioController	8
16.	Atualizar UsuarioService	8
17.	Adicionar configuração do Bean Validation no pom.xml	9
18.	Atualizar UsuarioDTO com validações do Bean Validation	9
19.	Atualizar UsuarioController com Bean Validation métodos (Post e Put)	10
20.	Tratamento de Exception Personalizado	10
20.1.	Criar DefaultException	10
20.2.	Criar StandardError	11
20.3.	Criar FieldMessage	12
20.4.	Criar ValidationError	12
20.5.	Criar ControllerExceptionHandler	13
21.	Tratar Erro de Duplicidade de E-mail no banco	14
21.1.	Adicionar no arquivo UsuarioRepository	14
21.2.	Atualizar UsuarioService para Tratamento de Exception	14
22.	Configuração do Swagger-ui – Documentação da API Rest	15
22.1.	Adicionar no arquivo pom.xml	15
22.2.	Adicionar Anotação no arquivo de inicialização do Springboot	15
22.3.	Adicionar comando no arquivo application.properties	16
22.4.	Criar arquivo de configuração do Swagger	16

23.	Configurar o Cors da Aplicação	16
23.1.	Adicionar ao arquivo pom.xml	16
23.2.	Criar SecurityConfig	16
24.	Atualizar UsuarioService para criptografar a senha do usuário	17
25.	Bonus Criar Tela de Login	18

## 1. Site para Criar o projeto - Spring Initializr

<https://start.spring.io/>

## 2. Dependencias do Projeto

- Spring Web
- Spring Boot Dev Tools
- Spring Data JPA
- H2 Database
- MySQL Driver (MS SQL Server Driver), (PostgreSQL Driver), (Oracle Driver)

## 3. Arquivo pom.xml

```

<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>

<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.28</version>
</dependency>

<!-- Cors Configuration -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>

<!--Banco de Dados -->

```

```

<!-- Beans Validation -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<!-- Beans Validation -->

```

```

<!-- Swagger API -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>

```

#### 4. Criar UsuarioController

```

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

    @GetMapping
    public String teste() {
        return "Teste de Controller Rest API";
    }
}

```

#### 5. Criar Entidade Usuario

```

import java.util.Date;

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

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

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String documento;
    private Date nascimento;
    private String periodo;
    private String sexo;
}

```

```

    private Boolean userStatus;
    private String recado;
    private String contato;
    private String senha;
    private String observacao;

    @Column(unique = true)
    private String email;
}

```

## 6. Criar UsuarioRepository

```

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

import com.linconviana.api_rest.entities.Usuario;

@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long>{

}

```

## 7. Criar arquivos de configuração de conexão com banco de dados

### 7.1. application.properties

```

#http://localhost:8080/h2-console
#http://localhost:8080/swagger-ui.html
spring.mvc.pathmatch.matching-strategy=ant-path-matcher
spring.profiles.active=${APP_PROFILE:test}
spring.jpa.open-in-view=false

```

### 7.2. application-test.properties

```

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

```

### 7.3. application-dev.properties

```

##### BANCO MYSQL #####
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/db_cadastro
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.show-sql: true

```

##### BANCO MYSQL #####

##### BANCO ORACLE #####

```
#spring.datasource.url=jdbc:oracle:thin@:localhost:1521:xe
#spring.datasource.username=admin
#spring.datasource.password=12345678
#spring.jpa.show-sql=true
#spring.jpa.properties.hibernate.format_sql=true
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
##### BANCO ORACLE #####
```

##### BANCO POSTGRESQL #####

```
#spring.jpa.properties.javax.persistence.schema-generation.create-source=metadata
#spring.jpa.properties.javax.persistence.schema-generation.scripts.action=create
#spring.jpa.properties.javax.persistence.schema-generation.scripts.create-target=create.sql
#spring.jpa.properties.hibernate.hbm2ddl.delimiter=;
```

```
#spring.datasource.url=jdbc:postgresql://localhost:5432/db_cadastro
#spring.datasource.username=postgres
#spring.datasource.password=123456
```

```
#spring.jpa.show-sql=true
#spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
#spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
#spring.jpa.hibernate.ddl-auto=none
##### BANCO POSTGRESQL #####
```

##### BANCO SQL SERVER #####

```
#spring.datasource.url=jdbc:sqlserver://localhost;databaseName=db_cadastro
#spring.datasource.username=sa
#spring.datasource.password=123456
#spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
#spring.jpa.show-sql=true
#spring.jpa.hibernate.ddl-auto=update
##### BANCO SQL SERVER #####
```

← → ↻ ⓘ localhost:8080/h2-console/login.jsp?sessionId=53b405e5ad434975f8922f5683ec8701

English ▾ Preferences Tools Help

Login	
Saved Settings:	Generic H2 (Embedded) ▾
Setting Name:	Generic H2 (Embedded) <input type="button" value="Save"/> <input type="button" value="Remove"/>
<hr/>	
Driver Class:	<input type="text" value="org.h2.Driver"/>
JDBC URL:	<input type="text" value="jdbc:h2:mem:testdb"/>
User Name:	<input type="text" value="sa"/>
Password:	<input type="password"/>
<input type="button" value="Connect"/> <input type="button" value="Test Connection"/>	

## 8. Testar adicionar um usuário

```
INSERT INTO `tb_usuario`(`id`, `contato`, `documento`, `email`, `nascimento`, `nome`, `observacao`, `periodo`, `recado`, `senha`, `sexo`, `user_status`) VALUES (1,(12) 98745-1212,'123.456.789-00','lincon@gmail.com','1982-11-30','Lincon','Teste Observação','manhã',(12) 3211-7845,'123456','masculino',true);
```

```
INSERT INTO `tb_usuario`(`id`, `contato`, `documento`, `email`, `nascimento`, `nome`, `observacao`, `periodo`, `recado`, `senha`, `sexo`, `user_status`) VALUES (2,(12) 3211-1299,'987.477.789-91','heitor@gmail.com','2015-02-26','Heitor','Teste Observação','tarde',(12) 3211-7845,'123456','masculino',false);
```

## 9. Atualizar UsuarioController Teste – Mais simples

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

    @Autowired
    private UsuarioRepository repository;

    @GetMapping
    public ResponseEntity<List<Usuario>> findAll() {

        List<Usuario> list = repository.findAll();

        return ResponseEntity.ok().body(list);
    }
}
```

## 10. Criar UsuarioDTO

```
import java.util.Date;

public class UsuarioDTO {

    private Long id;
    private String nome;
    private String documento;
    private Date nascimento;
    private String periodo;
    private String sexo;
    private Boolean userStatus;
    private String recado;
    private String contato;
    private String senha;
    private String observacao;
    private String email;
}
```

## 11. Criar UsuarioService

```
import java.util.List;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
```

```

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.linconviana.api_rest.dto.UsuarioDTO;
import com.linconviana.api_rest.entities.Usuario;
import com.linconviana.api_rest.repositories.UsuarioRepository;

@Service
public class UsuarioService {

    @Autowired
    private UsuarioRepository repository;

    @Transactional(readOnly = true)
    public List<UsuarioDTO> findAll() {

        List<Usuario> list = repository.findAll();
        return list.stream().map(x -> new UsuarioDTO(x)).collect(Collectors.toList());
    }
}

```

## 12. Atualizar UsuarioController – Correção padrão correto

```

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

    @Autowired
    private UsuarioService service;

    @GetMapping
    public ResponseEntity<List<UsuarioDTO>> findAll() {

        List<UsuarioDTO> list = service.findAll();

        return ResponseEntity.ok().body(list);
    }
}

```

## 13. Atualizar UsuarioController – Correção retornar Pageable

```

@GetMapping
public ResponseEntity<Page<UsuarioDTO>> findAll(Pageable pageable) {

    Page<UsuarioDTO> list = service.findAll(pageable);

    return ResponseEntity.ok().body(list);
}

```

## 14. Atualizar UsuarioService – Correção retornar Pageable

```

@Transactional(readOnly = true)
public Page<UsuarioDTO> findAll(Pageable pageable) {

```

```

        Page<Usuario> list = repository.findAll(pageable);

        return list.map(x -> new UsuarioDTO(x));
    }

```

## 15. Atualizar UsuarioController

```

@GetMapping(value =("/{id}")
public ResponseEntity<UsuarioDTO> findById(@PathVariable Long id) {

    UsuarioDTO usuarioDTO = service.findById(id);

    return ResponseEntity.ok().body(usuarioDTO);
}

@PostMapping
public ResponseEntity<UsuarioDTO> create(@RequestBody UsuarioDTO usuarioDTO) {

    usuarioDTO = service.create(usuarioDTO);

    return ResponseEntity.status(HttpStatus.CREATED).body(usuarioDTO);
}

@PutMapping(value =("/{id}")
public ResponseEntity<UsuarioDTO> findById(@PathVariable Long id, @RequestBody UsuarioDTO usuarioDTO) {

    usuarioDTO = service.update(id, usuarioDTO);

    return ResponseEntity.ok().body(usuarioDTO);
}

@DeleteMapping(value =("/{id}")
public ResponseEntity<Void> delete(@PathVariable Long id) {

    service.delete(id);

    return ResponseEntity.noContent().build();
}

```

## 16. Atualizar UsuarioService

```

@Transactional(readOnly = true)
public UsuarioDTO findById(Long id) {

    Optional<Usuario> obj = repository.findById(id);
    Usuario entity = obj.orElse();

    return new UsuarioDTO(entity);
}

@Transactional
public UsuarioDTO create(UsuarioDTO usuarioDTO) {

    Usuario entity = new Usuario();
    dtoToEntity(usuarioDTO, entity);
}

```



```

        entity = repository.save(entity);

        return new UsuarioDTO(entity);
    }

    @Transactional
    public UsuarioDTO update(Long id, UsuarioDTO usuarioDTO) {

        Optional<Usuario> obj = repository.findById(id);
        Usuario entity = obj.orElse();

        dtoToEntity(usuarioDTO, entity);
        entity = repository.save(entity);

        return new UsuarioDTO(entity);
    }

    public void delete(Long id) {

        repository.deleteById(id);
    }

    private void dtoToEntity(UsuarioDTO dto, Usuario entity) {

        entity.setNome(dto.getNome());
        entity.setDocumento(dto.getDocumento());
        entity.setNascimento(dto.getNascimento());
        entity.setPeriodo(dto.getPeriodo());
        entity.setSexo(dto.getSexo());
        entity.setUserStatus(dto.getUserStatus());
        entity.setRecado(dto.getRecado());
        entity.setContato(dto.getContato());
        entity.setSenha(dto.getSenha());
        entity.setObservacao(dto.getObservacao());
        entity.setEmail(dto.getEmail());
    }
}

```

## 17. Adicionar configuração do Bean Validation no pom.xml

```

<!-- Beans Validation -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<!-- Beans Validation -->

```

## 18. Atualizar UsuarioDTO com validações do Bean Validation

```

private Long id;

@NotBlank(message = "Campo email é obrigatório")
private String nome;

```

```
@NotBlank(message = "Campo documento é obrigatório")
private String documento;
```

```
@NotNull(message = "Campo data de nascimento é obrigatório")
private Date nascimento;
```

```
@NotBlank(message = "Campo periodo é obrigatório")
private String periodo;
```

```
@NotBlank(message = "Campo sexo é obrigatório")
private String sexo;
```

```
@NotNull(message = "Campo userStatus é obrigatório")
private Boolean userStatus;
private String recado;
```

```
@NotBlank(message = "Campo contato é obrigatório")
private String contato;
```

```
@NotBlank(message = "Campo senha é obrigatório")
@Size(min = 6, max = 128, message = "Senha deve conter no minimo 8 caracteres e no maximo 128 caracteres")
private String senha;
private String observacao;
```

```
@NotBlank(message = "Campo email é obrigatório")
>Email(message = "Email não é valido", regexp = "^[a-zA-Z0-9_!#$%&'*/+=?`{|}~^.-]+@[a-zA-Z0-9.-]+$")
private String email;
```

## 19. Atualizar UsuarioController com Bean Validation métodos (Post e Put)

```
@PostMapping
public ResponseEntity<UsuarioDTO> create(@Valid @RequestBody UsuarioDTO usuarioDTO) {

    usuarioDTO = service.create(usuarioDTO);

    return ResponseEntity.status(HttpStatus.CREATED).body(usuarioDTO);
}

@PutMapping(value =("/{id}")
public ResponseEntity<UsuarioDTO> findById(@PathVariable Long id, @Valid @RequestBody UsuarioDTO usuarioDTO) {

    usuarioDTO = service.update(id, usuarioDTO);

    return ResponseEntity.ok().body(usuarioDTO);
}
```

## 20. Tratamento de Exception Personalizado

### 20.1. Criar DefaultException

```
public class DefaultException extends RuntimeException {

    private static final long serialVersionUID = 1L;
```

```

    public DefaultException(String msg) {
        super(msg);
    }
}

```

## 20.2. Criar StandardError

```

import java.io.Serializable;
import java.time.Instant;

public class StandardError implements Serializable{

    private static final long serialVersionUID = 1L;

    private Instant timestamp;
    private Integer status;
    private String error;
    private String message;
    private String path;

    public StandardError() {}

    public Instant getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(Instant timestamp) {
        this.timestamp = timestamp;
    }

    public Integer getStatus() {
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }

    public String getError() {
        return error;
    }

    public void setError(String error) {
        this.error = error;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getPath() {
        return path;
    }
}

```

```

    }

    public void setPath(String path) {
        this.path = path;
    }
}

```

### 20.3. Criar FieldMessage

```

import java.io.Serializable;

public class FieldMessage implements Serializable {

    private static final long serialVersionUID = 1L;

    private String fieldName;
    private String message;

    public FieldMessage() {}

    public FieldMessage(String fieldName, String message) {
        super();
        this.fieldName = fieldName;
        this.message = message;
    }

    public String getFieldName() {
        return fieldName;
    }

    public void setFieldName(String fieldName) {
        this.fieldName = fieldName;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}

```

### 20.4. Criar ValidationError

```

import java.util.ArrayList;
import java.util.List;
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonInclude.Include;

@JsonInclude(Include.NON_NULL)
public class ValidationError extends StandardError{

    private static final long serialVersionUID = 1L;

```

```

private List<FieldMessage> errors = new ArrayList<>();

public List<FieldMessage> getErros(){
    return errors;
}

public void addError(String fieldName, String message) {
    errors.add(new FieldMessage(fieldName, message));
}
}

```

## 20.5. Criar ControllerExceptionHandler

```

import java.time.Instant;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@ControllerAdvice
public class ControllerExceptionHandler extends ResponseEntityExceptionHandler {

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException e,
        HttpHeaders headers, HttpStatus status, WebRequest request){

        ValidationError err = new ValidationError();
        status = HttpStatus.UNPROCESSABLE_ENTITY;

        err.setTimestamp(Instant.now());
        err.setStatus(status.value());
        err.setError("Validation Exception");
        err.setMessage(e.getMessage());
        err.setPath(request.getContextPath());

        for(FieldError f : e.getBindingResult().getFieldErrors()) {
            err.addError(f.getField(), f.getDefaultMessage());
        }

        return super.handleExceptionInternal(e, err, headers, status, request);
    }

    @ExceptionHandler(DefaultException.class)
    public ResponseEntity<Object> entityNotFound(DefaultException e, WebRequest request){

        StandardError err = new StandardError();
        HttpStatus status = HttpStatus.BAD_REQUEST;

        err.setTimestamp(Instant.now());
        err.setStatus(status.value());
        err.setError("Resource not found");
    }
}

```

```

        err.setMessage(e.getMessage());
        err.setPath(request.getContextPath());

        return super.handleExceptionInternal(e, err, new HttpHeaders(), status, request);
    }
}

```

## 21. Tratar Erro de Duplicidade de E-mail no banco

### 21.1. Adicionar no arquivo UsuarioRepository

```
Optional<Usuario> findByEmail(String email);
```

```

@Query(nativeQuery = true, value = "SELECT * FROM TB_USUARIO WHERE email IN (:email)")
Usuario findByUserEmail(String email);

```

### 21.2. Atualizar UsuarioService para Tratamento de Exception

```

@Transactional(readOnly = true)
public UsuarioDTO findById(Long id) {

    Optional<Usuario> obj = repository.findById(id);
    Usuario entity = obj.orElseThrow(() -> new RuntimeException ("Usuario não encontrado com id : " + id));

    return new UsuarioDTO(entity);
}

@Transactional
public UsuarioDTO create(UsuarioDTO usuarioDTO) {

    /*Optional<Usuario> obj = repository.findByEmail(usuarioDTO.getEmail());
    if(!obj.isEmpty()) {
        throw new DefaultException("Já existe um usuario cadastrado com e-mail: " + usuarioDTO.getEmail());
    }*/

    Usuario obj = repository.findByUserEmail(usuarioDTO.getEmail());
    if(obj != null) {
        throw new DefaultException("Já existe um usuario cadastrado com e-mail: " + usuarioDTO.getEmail());
    }

    Usuario entity = new Usuario();
    dtoToEntity(usuarioDTO, entity);
    entity = repository.save(entity);

    return new UsuarioDTO(entity);
}

@Transactional
public UsuarioDTO update(Long id, UsuarioDTO usuarioDTO) {

    try {

        Optional<Usuario> obj = repository.findById(id);

```

```

        Usuario entity = obj.orElseThrow(() -> new DefaultException("Usuario não encontrado com id: " + id));

        dtoToEntity(usuarioDTO, entity);
        entity = repository.save(entity);

        return new UsuarioDTO(entity);

    } catch (EntityNotFoundException e) {
        throw new DefaultException("Usuario não encontrado com id: " + id);
    }
}

public void delete(Long id) {

    try {
        repository.deleteById(id);

    } catch (EmptyResultDataAccessException e) {
        throw new DefaultException("Usuario não encontrado com id: " + id);
    }
    catch (DataIntegrityViolationException e) {
        throw new DefaultException("Integrity violation");
    }
}

```

## 22. Configuração do Swagger-ui – Documentação da API Rest

### 22.1. Adicionar no arquivo pom.xml

```

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>

```

### 22.2. Adicionar Anotação no arquivo de inicialização do Springboot

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@SpringBootApplication
@EnableSwagger2
public class ApiRestApplication {

    public static void main(String[] args) {
        SpringApplication.run(ApiRestApplication.class, args);
    }
}

```

```
}
```

### 22.3. Adicionar comando no arquivo application.properties

```
spring.mvc.pathmatch.matching-strategy=ant-path-matcher
```

### 22.4. Criar arquivo de configuração do Swagger

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
```

```
@Configuration
public class SwaggerConfiguration {

    @Bean
    public Docket docket() {

        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.any())
            .paths(PathSelectors.any())
            .build();
    }
}
```

## 23. Configurar o Cors da Aplicação

### 23.1. Adicionar ao arquivo pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

### 23.2. Criar SecurityConfig

```
import java.util.Arrays;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
```



```

import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private Environment env;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        if(Arrays.asList(env.getActiveProfiles()).contains("test")) {
            http.headers().frameOptions().disable();
        }

        http.cors().and().csrf().disable();
        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        http.authorizeRequests().anyRequest().permitAll();
    }

    @Bean
    CorsConfigurationSource corsConfigurationSource() {

        CorsConfiguration configuration = new CorsConfiguration().applyPermitDefaultValues();
        configuration.setAllowedOrigins(Arrays.asList("http://localhost:3000", "*"));
        configuration.setAllowedMethods(Arrays.asList("POST", "GET", "PUT", "DELETE", "OPTIONS"));
        final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

```

#### 24. Atualizar UsuarioService para criptografar a senha do usuário

```

private void dtoToEntity(UsuarioDTO dto, Usuario entity) {

    entity.setNome(dto.getNome());
    entity.setDocumento(dto.getDocumento());
    entity.setNascimento(dto.getNascimento());
    entity.setPeriodo(dto.getPeriodo());
    entity.setSexo(dto.getSexo());
    entity.setUserStatus(dto.getUserStatus());
    entity.setRecado(dto.getRecado());
    entity.setContato(dto.getContato());
    if(dto.getSenha() != "") {
        String encryptedPassword = new BCryptPasswordEncoder().encode(dto.getSenha());
        entity.setSenha(encryptedPassword);
    }
    entity.setObservacao(dto.getObservacao());
}

```

```
entity.setEmail(dto.getEmail());  
}
```

## 25. Bonus Criar Tela de Login

```
import javax.persistence.EntityNotFoundException;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.crypto.password.PasswordEncoder;  
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 com.linconviana.api_rest.dto.UsuarioDTO;  
import com.linconviana.api_rest.entities.Usuario;  
import com.linconviana.api_rest.exceptions.DefaultException;  
import com.linconviana.api_rest.repositories.UsuarioRepository;  
  
@RestController  
@RequestMapping(value = "/login")  
public class LoginController {  
  
    private final UsuarioRepository repository;  
    private final PasswordEncoder passwordEncoder;  
  
    private LoginController(final UsuarioRepository repository, PasswordEncoder passwordEncoder) {  
        this.repository = repository;  
        this.passwordEncoder = passwordEncoder;  
    }  
  
    @PostMapping  
    public ResponseEntity<String> login(@RequestBody UsuarioDTO dto) {  
  
        try {  
            Usuario usuario = this.repository.findByUserEmail(dto.getEmail());  
            if(usuario == null) {  
                throw new DefaultException("Usuario não encontrado!");  
            }  
  
            boolean verifyPassword = this.passwordEncoder.matches(dto.getSenha(), usuario.getSenha());  
  
            if(verifyPassword)  
                return ResponseEntity.ok().body("OK");  
  
            return ResponseEntity.notFound().build();  
        } catch (EntityNotFoundException e) {  
            throw new DefaultException("Usuario não encontrado!");  
        }  
    }  
}
```