Conclusão: Spring Boot 3 - Boas Práticas e Segurança em APIs REST

September 18, 2025

1 Introdução

Parabéns por concluir o curso *Spring Boot 3: Aplique Boas Práticas e Proteja uma API Rest*! Neste curso, aprimoramos um projeto existente, introduzindo boas práticas de desenvolvimento e segurança com Spring Boot. Organizamos a estrutura do projeto, padronizamos respostas HTTP, implementamos tratamento de erros centralizado, e configuramos autenticação e autorização com tokens JWT usando Spring Security. Esta conclusão recapitula os principais aprendizados e prepara para os próximos passos. As alterações foram feitas no `Visual Studio Code` ou sua IDE preferida, com testes no Insomnia.

2 Revisão dos Aprendizados

2.1 Estrutura do Projeto

Reorganizamos o projeto em três pacotes principais:

- `med.voll.api.controller`: Contém os controllers (`AutenticacaoController`, `HelloController`, `MedicoController`, `PacienteController`).
- `med.voll.api.domain`: Contém as entidades de domínio (`Medico`, `Paciente`, `Usuario`) e repositórios.
- `med.voll.api.infra`: Contém configurações de infraestrutura, como segurança (`security`) e tratamento de erros (`exception`).

2.2 Padronização de Respostas com `ResponseEntity`

No `MedicoController` (pacote `med.voll.api.controller`), padronizamos os métodos para usar `ResponseEntity`, aplicando códigos HTTP apropriados:

```
package med.voll.api.controller;

import jakarta.transaction.Transactional;
import jakarta.validation.Valid;
import med.voll.api.domain.medico.DadosCadastroMedico;
import med.voll.api.domain.medico.DadosDetalhamentoMedico;
import med.voll.api.domain.medico.Medico;
import med.voll.api.domain.medico.MedicoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.util.UriComponentsBuilder;

@RestController
@RequestMapping("/medicos")
public class MedicoController {
```

```
17
18      @Autowired
19      private MedicoRepository repository;
20
21      @PostMapping
22      @Transactional
23      public ResponseEntity cadastrar(@RequestBody @Valid DadosCadastroMedico dados,
            UriComponentsBuilder uriBuilder) {
24          var medico = new Medico(dados);
25          repository.save(medico);
26          var uri =
                uriBuilder.path("/medicos/{id}").buildAndExpand(medico.getId()).toUri();
27          return ResponseEntity.created(uri).body(new
                DadosDetalhamentoMedico(medico));
28      }
29
30      @DeleteMapping("/{id}")
31      @Transactional
32      public ResponseEntity excluir(@PathVariable Long id) {
33          var medico = repository.getReferenceById(id);
34          medico.excluir();
35          return ResponseEntity.noContent().build();
36      }
37  }
```

Explicação:

- `cadastrar`: Retorna código 201 (Created) com o cabeçalho `Location` contendo a URI do recurso criado, usando `UriComponentsBuilder`.
- `excluir`: Retorna código 204 (No Content) para deleções bem-sucedidas, sem corpo na resposta.

2.3 Tratamento de Erros com `ControllerAdvice`

No pacote `med.voll.api.infra.exception`, criamos a classe `TratadorDeErros` para centralizar o tratamento de exceções:

```
1   package med.voll.api.infra.exception;
2
3   import jakarta.persistence.EntityNotFoundException;
4   import org.springframework.http.ResponseEntity;
5   import org.springframework.validation.FieldError;
6   import org.springframework.web.bind.MethodArgumentNotValidException;
7   import org.springframework.web.bind.annotation.ExceptionHandler;
8   import org.springframework.web.bind.annotation.RestControllerAdvice;
9
10  import java.util.List;
11
12  @RestControllerAdvice
13  public class TratadorDeErros {
14
15      @ExceptionHandler(EntityNotFoundException.class)
16      public ResponseEntity tratarErro404() {
17          return ResponseEntity.notFound().build();
18      }
19
20      @ExceptionHandler(MethodArgumentNotValidException.class)
21      public ResponseEntity tratarErro400(MethodArgumentNotValidException ex) {
22          var erros = ex.getFieldErrors();
23          return
                ResponseEntity.badRequest().body(erros.stream().map(DadosErroValidacao::new).toList())
24      }
```

```
25
26      private record DadosErroValidacao(String campo, String mensagem) {
27          public DadosErroValidacao(FieldError erro) {
28              this(erro.getField(), erro.getDefaultMessage());
29          }
30      }
31  }
```

Explicação:

- `@RestControllerAdvice`: Centraliza o tratamento de exceções para todos os controllers.
- `tratarErro404`: Retorna código 404 (Not Found) para `EntityNotFoundException`.
- `tratarErro400`: Retorna código 400 (Bad Request) com um JSON simplificado para erros de validação.

2.4 Segurança com Spring Security e JWT

Dedicamos várias aulas à implementação de autenticação e autorização com Spring Security e tokens JWT:

- **SecurityConfigurations**: Configuramos autenticação *stateless* e hashing de senhas com BCrypt.

```
1  package med.voll.api.infra.security;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.context.annotation.Bean;
5  import org.springframework.context.annotation.Configuration;
6  import org.springframework.http.HttpMethod;
7  import
       org.springframework.security.config.annotation.web.builders.HttpSecurity;
8  import
       org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
9  import org.springframework.security.config.http.SessionCreationPolicy;
10 import org.springframework.security.web.SecurityFilterChain;
11 import
       org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
12
13 @Configuration
14 @EnableWebSecurity
15 public class SecurityConfigurations {
16
17     @Autowired
18     private SecurityFilter securityFilter;
19
20     @Bean
21     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
           Exception {
22         return http.csrf().disable()
23                 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
24                 .and().authorizeHttpRequests()
25                 .requestMatchers(HttpMethod.POST, "/login").permitAll()
26                 .anyRequest().authenticated()
27                 .and().addFilterBefore(securityFilter,
                       UsernamePasswordAuthenticationFilter.class)
28                 .build();
29     }
30 }
```

- **SecurityFilter**: Intercepta requisições, valida tokens JWT, e autentica usuários com `SecurityContextHol`

```java
package med.voll.api.infra.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import med.voll.api.domain.usuario.UsuarioRepository;
import org.springframework.beans.factory.annotation.Autowired;
import
    org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;

@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private TokenService tokenService;

    @Autowired
    private UsuarioRepository repository;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response, FilterChain filterChain) throws
        ServletException, IOException {
        var tokenJWT = recuperarToken(request);
        if (tokenJWT != null) {
            var subject = tokenService.getSubject(tokenJWT);
            var usuario = repository.findByLogin(subject);
            var authentication = new
                UsernamePasswordAuthenticationToken(usuario, null,
                usuario.getAuthorities());
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        filterChain.doFilter(request, response);
    }

    private String recuperarToken(HttpServletRequest request) {
        var authorizationHeader = request.getHeader("Authorization");
        if (authorizationHeader != null) {
            return authorizationHeader.replace("Bearer ", "");
        }
        return null;
    }
}
```

- **TokenService**: Gera e valida tokens JWT com a biblioteca Auth0 (`java-jwt`).

```java
package med.voll.api.infra.security;

import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.JWTCreationException;
import com.auth0.jwt.exceptions.JWTVerificationException;
import med.voll.api.domain.usuario.Usuario;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
```

```java

@Service
public class TokenService {

    @Value("${api.security.token.secret}")
    private String secret;

    public String gerarToken(Usuario usuario) {
        try {
            var algoritmo = Algorithm.HMAC256(secret);
            return JWT.create()
                    .withIssuer("API Voll.med")
                    .withSubject(usuario.getLogin())
                    .withExpiresAt(dataExpiracao())
                    .sign(algoritmo);
        } catch (JWTCreationException exception) {
            throw new RuntimeException("Erro ao gerar token JWT", exception);
        }
    }

    public String getSubject(String tokenJWT) {
        try {
            var algoritmo = Algorithm.HMAC256(secret);
            return JWT.require(algoritmo)
                    .withIssuer("API Voll.med")
                    .build()
                    .verify(tokenJWT)
                    .getSubject();
        } catch (JWTVerificationException exception) {
            throw new RuntimeException("Token JWT inválido ou expirado!",
                exception);
        }
    }

    private Instant dataExpiracao() {
        return
            LocalDateTime.now().plusHours(2).toInstant(ZoneOffset.of("-03:00"));
    }
}
```

- **Usuario e AutenticacaoService**: No pacote `med.voll.api.domain.usuario`, a classe `Usuario` implementa `UserDetails`, e `AutenticacaoService` implementa `UserDetailsService` para autenticação.

```java
package med.voll.api.domain.usuario;

import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class AutenticacaoService implements UserDetailsService {

    @Autowired
    private UsuarioRepository repository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
        UsernameNotFoundException {
        return repository.findByLogin(username);
    }
}
```

- **DadosAutenticacao**: DTO para representar login e senha.

```
package med.voll.api.domain.usuario;

public record DadosAutenticacao(String login, String senha) {
}
```

3 Próximos Passos

Concluímos o curso com uma API REST robusta, com respostas padronizadas, tratamento de erros centralizado, e autenticação/autorização seguras usando tokens JWT. No entanto, há mais a explorar no Spring Boot:

- Documentação da API com ferramentas como Swagger/OpenAPI.
- Implementação de novas funcionalidades, como agendamento de consultas.
- Testes automatizados com Spring Boot.

Esses tópicos serão abordados em um próximo curso. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

4 Dica do Professor

- **Revisite o Spring Security**: Consulte a documentação oficial (`https://docs.spring.io/spring-security/reference/index.html`) para aprofundar autenticação e autorização.
- **Comunidade no X**: Siga perfis como @SpringBootPro e @JWTSecurityGuru no X para dicas sobre Spring e segurança em APIs.
- **Pratique**: Na plataforma de aprendizado, explore os *challenges* para aplicar autenticação e autorização em outros projetos. Teste requisições no Insomnia com tokens válidos e inválidos para reforçar o entendimento.