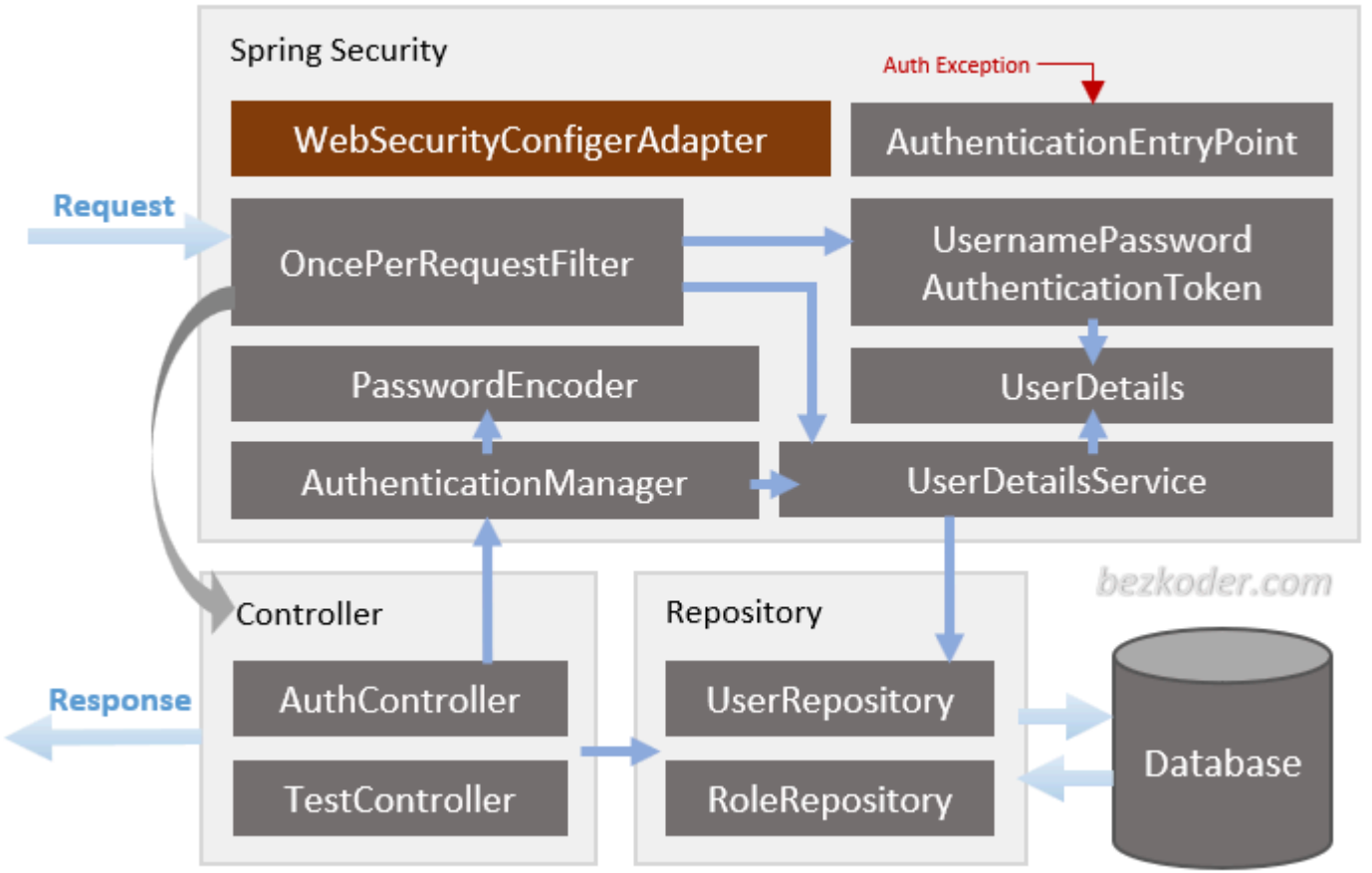
Spring Security e JWT

Laboratório de Programação

Spring Security

- Conjunto de funcionalidades pré-construidas que permitem adicionar camadas de segurança na aplicação
- No nosso caso, queremos:
  - Proteger a API de acesso não autorizado

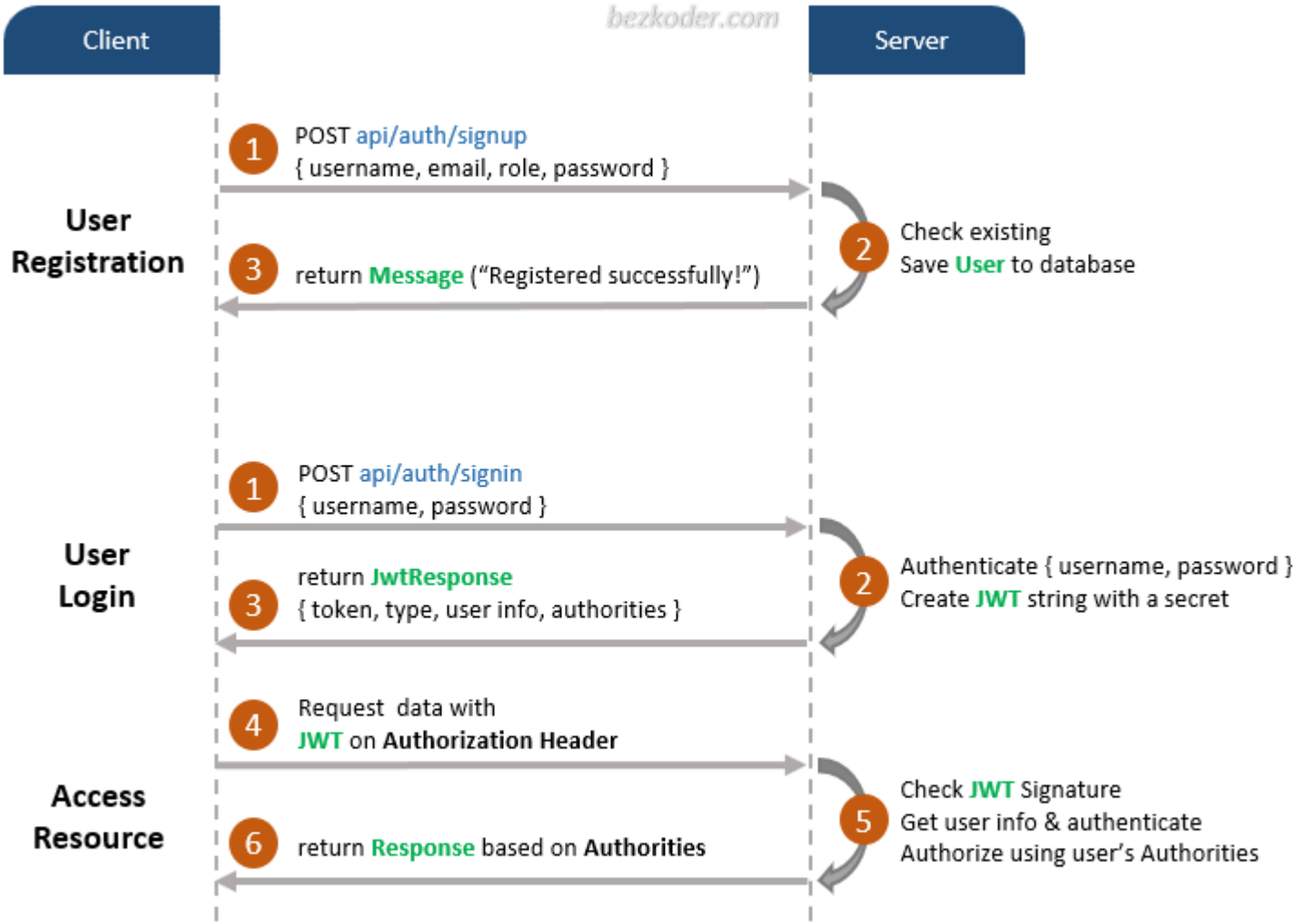
Spring Security Architecture



JSON Web Tokens (JWT)

- Padrão RFC 7519 (site jwt.io)
- Codificadas como um objeto JSON
- Permite que as declarações sejam assinadas digitalmente ou protegidas por integridade com um Código de Autenticação de Mensagem (MAC) e / ou criptografado.

JSON Web Tokens (JWT)

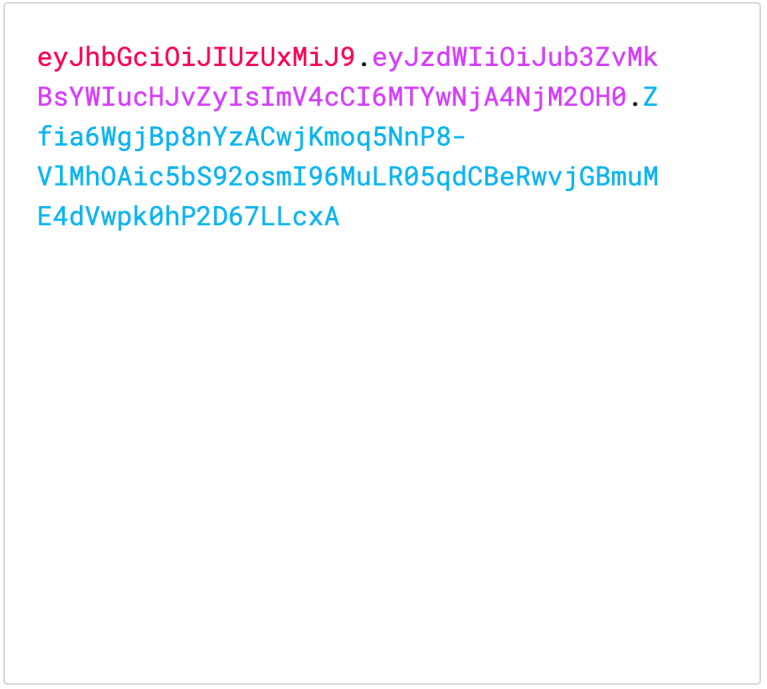


JSON Web Tokens (JWT)

- Um JWT é composto por três partes separadas por ponto . hhh.ppp.sss
  - header: informações sobre o token
  - payload: os atributos (claims)
  - signature: verificação de remetente
- Ex:

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJub3ZvMkBsYWIucHJvZyIsImV4cCI6MTYwNjA4NjM
```

JSON Web Tokens (JWT)



Habilitando JWT em nossa API: Agenda

- Configuração do POM.XML
- Mudanças na classe com Main para adicionar o `PasswordEncoder` bean
- Mudanças em UsuarioCotroller para adicionar criptografia
- Mudanças em UsuarioService para implementar `UserDetailsService`
- Construir o `WebSecurityConfigurerAdapter` para o projeto
- Construir os filtros de Autenticação (que fazem a autenticação)
- Construir os filtros de Autorização (checa o token do JWT no header)

tem coisa!

Configuração do POM.xml

- Necessário adicionar dependências `security`, `jsonwebtoken`
- Test é opcional

```
<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>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.7.0</version>
</dependency>
```

Revisão em PatrimonioApplication:

- Criar um `@Bean` para poder fazer criptografia de senha

```
@SpringBootApplication
@EnableWebMvc
public class PatrimonioApplication {

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

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

}
```

BCryptPasswordEncoder

- Criptografia de via única
- Adiciona um ruído dentro do hash
- Mais informações: Registration with Spring Security – Password Encoding

Revisão em UsuarioController

- Adicionando o bean para poder usar criptografia

```
@RestController
@RequestMapping("/api/usuarios")
public class UsuarioController {
    @Autowired
    UsuarioService service;

    @Autowired
    private PasswordEncoder passwordEncoder;
```

Revisão em UsuarioController

- Revisando o salvar usuario, adicionando criptografia da senha

```
@PostMapping
public ResponseEntity salvar(@RequestBody UsuarioDTO dto) {

    Usuario usuario = Usuario.builder()
                                .nome(dto.getNome())
                                .email(dto.getEmail())
                                .senha(passwordEncoder.encode(dto.g
    try {
        Usuario salvo = service.salvar(usuario);
        return new ResponseEntity(salvo, HttpStatus.CREATED);
    }
    catch (RegraNegocioRunTime e) {
        return ResponseEntity.badRequest().body(e.getMessage());
    }
}
```

Revisão em UsuarioService

- Para se adequar ao `SpringSecurity`, precisamos implementar um mecanismo de obter informações de usuários
  - necessário implementar a interface `UserDetailsService`
  - e o método `public UserDetails loadUserByUsername(String username)` que fornece como resposta um `User` para o `SpringSecurity`
  - esse `User` pode também conter a lista de `Roles` da aplicação

Revisão em UsuarioService

```
@Service
public class UsuarioService implements UserDetailsService {

...
  @Override
  public UserDetails loadUserByUsername(String email)
      throws UsernameNotFoundException {

      Optional<Usuario> usr = repository.findByEmail(email);
      if (!usr.isPresent())
          throw new UsernameNotFoundException(email);

      Usuario a = usr.get();
      return new User(a.getEmail(), a.getSenha(), emptyList());
  }

}
```

Instanciando o controle de acesso

- Precisamos dizer ao `SpringSecurity` como usar a segurança
- Para isso configuramos o `WebSecurityConfigurerAdapter`
- Aqui, criei uma classe que extende:

```
package com.labprog.patrimonio.security;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration
          extends WebSecurityConfigurerAdapter {
    @Autowired
    private UsuarioService service;

    @Autowired
    private PasswordEncoder passwordEncoder;
}
```

Instanciando o controle de acesso

- Para facilitar, criado uma classe com algumas configurações:

```
package com.labprog.patrimonio.security;

public class SecurityConstants {
    public static final String SIGN_UP_URL = "/api/usuarios";
    public static final String KEY = "q3t6w9z$C&F)J@NcQfTjWnZr4u7x!
    public static final String HEADER_NAME = "Authorization";
    public static final Long EXPIRATION_TIME = 1000L*60*30;
}
```

Instanciando o controle de acesso

- 3 métodos básicos: configure: método de autenticação e as requisições e cors

```
@Override
  protected void configure(HttpSecurity http) throws Exception {
      http.cors().and().csrf().disable()
        .authorizeRequests()
        //a linha a seguir pode ser retirada
        .antMatchers(HttpMethod.POST, SIGN_UP_URL).permitAll()
        //URL pública
        .antMatchers(HttpMethod.POST, "/login").permitAll()
        .anyRequest().authenticated()
        .and()
        //quem vai autenticar e como
        .addFilter(new AuthenticationFilter(authenticationManager())
        //quem vai autorizar e como
        .addFilter(new AuthorizationFilter(authenticationManager()))
        .sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
  }
```

Instanciando o controle de acesso

```
@Override
  public void configure(AuthenticationManagerBuilder auth)
      throws Exception {

      // configura o método de autenticação
      auth.userDetailsService(service)
          .passwordEncoder(passwordEncoder);
  }
```

Instanciando o controle de acesso

```
@Bean
  CorsConfigurationSource corsConfigurationSource() {
      final UrlBasedCorsConfigurationSource source =
        new UrlBasedCorsConfigurationSource();
      source
        .registerCorsConfiguration("/**", new CorsConfiguration()
        .applyPermitDefaultValues());
      return source;
  }
```

Construir o filtro de Autenticação

- extende o `UsernamePasswordAuthenticationFilter`
- implementa o `attemptAuthentication` que tenta a autenticação
- e o `successfulAuthentication` que caso tenha dado certo, adicionar o `JWT token` no Header

Filtro de autenticação:

```
public class AuthenticationFilter
       extends UsernamePasswordAuthenticationFilter {

    private AuthenticationManager authenticationManager;

    public AuthenticationFilter(AuthenticationManager authenticatic
        this.authenticationManager = authenticationManager;
    }

}
```

Filtro de autenticação:

```
@Override
public Authentication attemptAuthentication(
                        HttpServletRequest req,
                        HttpServletResponse res)
            throws AuthenticationException {
    try {
        UsuarioDTO usuario =
            new ObjectMapper().readValue(req.getInputStream(), Us


        return authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                        usuario.getEmail(),
                        usuario.getSenha(),
                        new ArrayList<>())
        );
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Filtro de autenticação:

```
@Override
protected void successfulAuthentication(HttpServletRequest req,
                                        HttpServletResponse res,
                                        FilterChain chain,
                                        Authentication auth)
                    throws IOException, ServletException {

    String JWT = Jwts.builder()
                    .setSubject(auth.getName())
                    .setExpiration(new Date(System.currentTimeMil
                                    + EXPIRATION_TIME))
                    .signWith(SignatureAlgorithm.HS512, KEY)
                    .compact();
    res.addHeader("token", JWT);
}
```

Construir o filtro de Autorização

- Todas as requisições bloqueadas terão que ter um token ativo e autenticado
- Autorização checa o token

```
public class AuthorizationFilter extends BasicAuthenticationFilter

    public AuthorizationFilter(AuthenticationManager authManager) {
        super(authManager);
    }
```

Filtro de Autorização

```
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain chain)
            throws IOException, ServletException {

    //pega o token
    String header = request.getHeader(HEADER_NAME);

    if (header == null) {
        chain.doFilter(request, response);
        return;
    }

    //tenta autenticar
    UsernamePasswordAuthenticationToken authentication = authenti

    SecurityContextHolder.getContext().setAuthentication(authenti
    chain.doFilter(request, response);
}
```

Filtro de Autorização

```
private UsernamePasswordAuthenticationToken authenticate(HttpServ
    //pega o toke
    String token = request.getHeader(HEADER_NAME);

    if (token != null) {
        // faz parse do token
        String user = Jwts.parser()
            .setSigningKey(KEY)
            .parseClaimsJws(token)
            .getBody()
            .getSubject();
        if (user != null) {
            return new UsernamePasswordAuthenticationToken(user,
        }else{
            return  null;
        }
    }
    return null;
}
```

Teste com insomnia

+ React

- Em `response.headers` procure pelo `token`, e guarde em sessão
- E mude o token em `ApiService.js`

```
class ApiService {
    constructor (apiUrl, apiToken) {
        this.apiUrl = apiUrl
        instance.defaults.headers.common['Authorization'] = apiToke
    }
    ...
}
```

Fonte: Axios