

Spring Security

KI/WAJ/22

RNDr. Dávid Držík, PhD. – ddrzik@ukf.sk



Úvod do bezpečnosti webových aplikácií

- každá webová aplikácia komunikuje cez internet
 - dáta a používateľské účty sú **potenciálne ohrozené**
- útočníci sa môžu pokúsiť:
 - získať **citlivé údaje** (heslá, osobné informácie)
 - zneužívať **identitu** používateľa
 - vykonávať **neautorizované operácie** (napr. mazanie, zmena dát)
 - narúšať **dostupnosť systému** (napr. cez útoky DDoS)
- **najčastejšie typy útokov:**
 - SQL Injection, Cross-Site Scripting, Cross-Site Request Forgery, Brute-force útoky, Session hijacking, ...

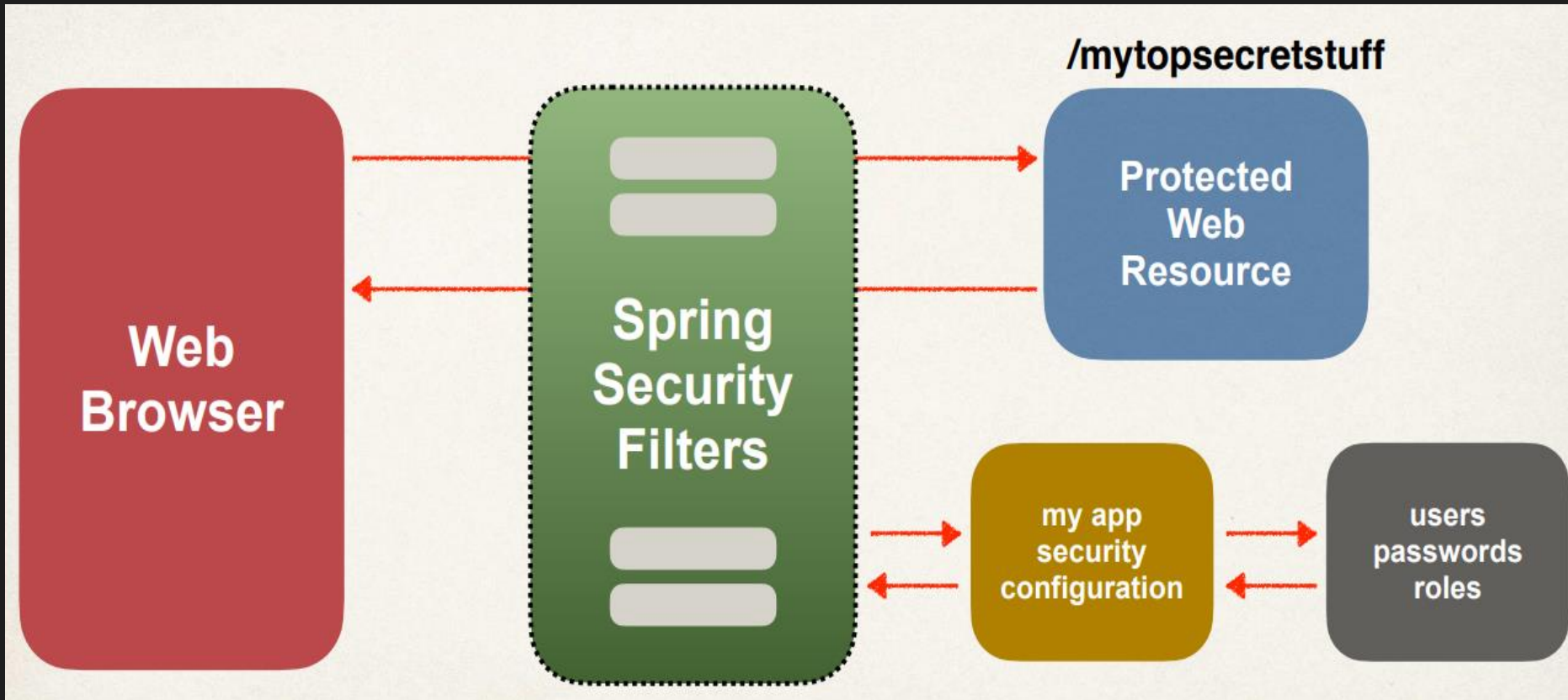
Spring Security

- bezpečnostný framework pre Spring aplikácie, ktorý zabezpečuje **autentifikáciu** a **autorizáciu** používateľov a poskytuje **ochranu** pred bežnými webovými útokmi
- Autentifikácia
 - overenie totožnosti používateľa
- Autorizácia
 - kontrola oprávnení používateľa



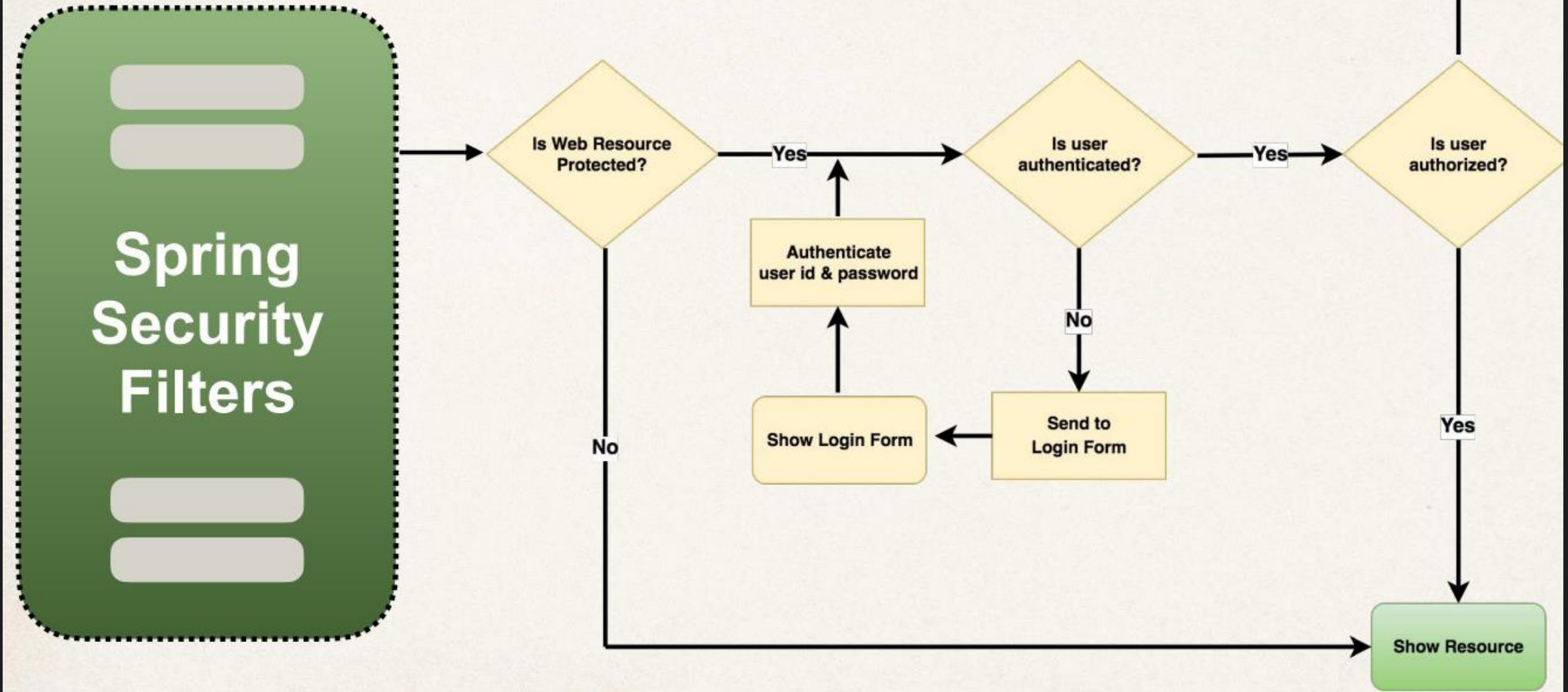
Ako Spring Security spracováva požiadavky (Filter Chain)

- implementácia pomocou **Servlet filtrov** na pozadí



Každý request prejde reťazcom filtrov, ktoré **overia identitu používateľa**, jeho **oprávnenia** a až potom mu umožnia prístup k chráneným častiam aplikácie.

Security Filters v praxi



Pridanie dependencies pre Security

pom.xml

```
<!-- Spring Security -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- Thymeleaf Security -->
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
<!-- Password Encoder (BCrypt) -->
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-crypto</artifactId>
</dependency>
```

Integruje **základné bezpečnostné funkcie** do aplikácie.
Zahŕňa autentifikáciu, autorizáciu a ochranu proti bežným útokom.

Umožňuje **prístup k bezpečnostným informáciám** priamo v Thymeleaf šablónach, napríklad na kontrolu, či je používateľ prihlásený alebo má určité role (oprávnenia) v systéme.

Poskytuje **kryptografické funkcie**, ako je hashovanie hesiel pomocou Bcrypt.

Používatelia v našej databáze

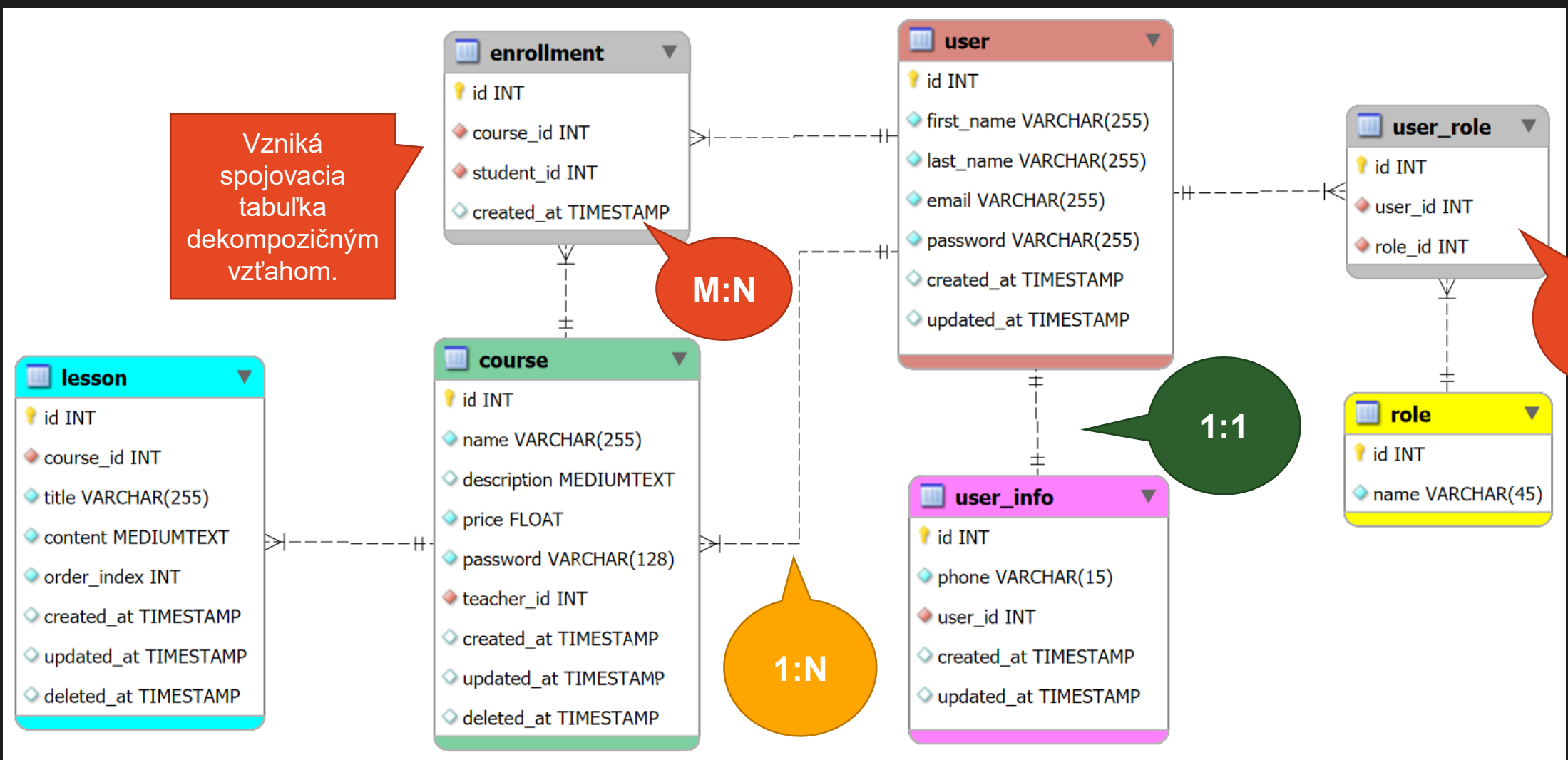
- heslá sú v DB už šifrované pomocou **BCrypt** hashovacej funkcie
 - nástroj na bezpečné "zašifrovanie" hesla pred jeho uložením
 - aj keď dvaja ľudia majú **rovnaké heslo**, výsledný **hash bude iný**

user_id	first_name	last_name	email	password	roles
1	Ján	Novák	jan.novak@ukf.sk	\$2y\$10\$D8koo29QyrKq6/Myt6al	ROLE_student
2	Mária	Kováčová	maria.kovacova@ukf.sk	\$2y\$10\$KfX00SkUIK.w6WttDfzV	ROLE_student
3	Peter	Horváth	peter.horvath@ukf.sk	\$2y\$10\$D8koo29QyrKq6/Myt6al	ROLE_teacher
4	Dávid	Držík	ddrzik@ukf.sk	\$2y\$10\$D8koo29QyrKq6/Myt6al	ROLE_student, ROLE_teacher
5	Zuzana	Novotná	zuzana.novotna@ukf.sk	\$2y\$10\$KfX00SkUIK.w6WttDfzV	ROLE_student

- pre naše účely vývoja a testovania aplikácie majú všetci **muži** v DB heslo **123** a **ženy** majú heslo **456**

V Spring Boot je požadovaný tvar:
ROLE_nazov

Naša databáza UDEMY – ešte raz

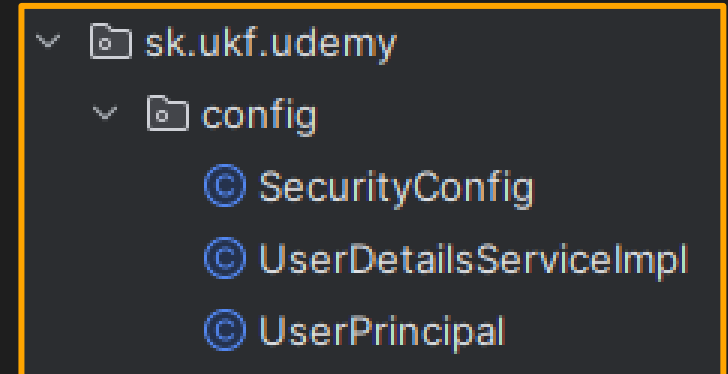


Konfigurácia Spring Security v projekte

- tri základné triedy, ktoré spolu vytvárajú bezpečnostnú logiku aplikácie

- **SecurityConfig**

- **hlavná konfiguračná trieda** pre Spring Security
 - definuje **prístupové pravidlá**, login a logout proces
 - prepája autentifikáciu s databázou používateľov



- **UserDetailsServiceImpl**

- implementuje **načítanie používateľa** z databázy podľa emailu
 - poskytuje Spring Security údaje o používateľovi a jeho rolách

- **UserPrincipal**

- **adaptér** medzi entitou User a bezpečnostným rozhraním Springu
 - obsahuje prihlasovacie údaje, role a doplnkové informácie (meno, email, iniciály)



SecurityConfig 1/4

java/sk/ukf/udemy/config/SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private UserDetailsServiceImpl userDetailsService;

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

    ...

}
```

@Configuration označuje triedu ako konfiguračnú – Spring ju načíta pri štarte aplikácie a použije jej metódy označené ako **@Bean**.

Referencia na prezentáciu (01x03) z prvého seminára.

@EnableWebSecurity aktivuje Spring Security v aplikácii a umožňuje prispôbenie jej správania cez vlastnú konfiguráciu.

Vlastná implementácia **UserDetailsServiceImpl**, ktorou sa získavajú používatelia z databázy. Vďaka tomuto vie SecurityConfig, **ako a kde overiť používateľov** počas loginu.



SecurityConfig 2/4 – Hashovanie hesla

java/sk/ukf/udemy/config/SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private UserDetailsServiceImpl userDetailsService;

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

    ...
}
```

Heslá by **nikdy nemali byť uložené v čitateľnej podobe**.
Tento bean definuje spôsob, akým sa budú heslá **hashovať** a porovnávať.

BCryptPasswordEncoder

- používa algoritmus **BCrypt**, (bezpečný a moderný)
- generuje **unikátny „salt“**, takže ani rovnaké heslá nemajú rovnaký výsledný hash
- BCrypt je zároveň odolný voči „brute-force“ útokom vďaka svojej výpočtovej náročnosti

Pri **registrácii** sa heslo "student123" sa uloží ako **\$2a\$10\$7VqfWvbiZ...** (náhodne generovaný hash).
Pri **prihlasovaní** PasswordEncoder porovná zadané heslo s hashom z databázy.



SecurityConfig 3/4 - AuthenticationManager

Centrálny komponent Spring Security, ktorý **vykonáva autentifikáciu**, overuje prihlasovacie údaje používateľa (napr. email a heslo).

java/sk/ukf/udemy/config/SecurityConfig.java

```
@Bean
public AuthenticationManager authenticationManager(HttpSecurity http) throws Exception {
    AuthenticationManagerBuilder amb = http.getSharedObject(AuthenticationManagerBuilder.class);
    amb.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    return amb.build();
}
```

Pomocou **AuthenticationManagerBuilder** sa definuje ako sa majú používateľské údaje načítať a overovať.

Používa **našu vlastnú implementáciu** pre načítanie používateľa z DB.

Overuje heslo pomocou **BCrypt hashovania**.



SecurityConfig 4/4 – SecurityFilterChain 1/5

java/sk/ukf/udemy/config/SecurityConfig.java

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests(authorizeRequests ->
            authorizeRequests
                .requestMatchers("/", "/auth/login", "/auth/register").permitAll()
                .requestMatchers("/course/**", "/error").permitAll()
                .requestMatchers("/css/**", "/js/**", "/images/**", "/webjars/**").permitAll()
                .requestMatchers("/admin/**").hasAnyAuthority("ROLE_admin")
                .requestMatchers("/teacher/**").hasAnyAuthority("ROLE_teacher")
                .requestMatchers("/student/**").hasAnyAuthority("ROLE_student")
                .anyRequest().authenticated()
        )

    // + ďalšie: formLogin, logout, exceptionHandling

    return http.build();
}
```

Metóda **securityFilterChain** definuje, ako Spring Security **autentifikuje** používateľov a **autorizuje** ich prístup k jednotlivým častiam aplikácie.

Metóda **authorizeHttpRequests()** určuje, ktoré URL adresy sú verejné a ktoré vyžadujú prihlásenie alebo rolu.

- **permitAll()** - stránky prístupné každému
- **hasAnyAuthority("ROLE_admin")** - prístup len pre používateľov s danou rolou
- **anyRequest().authenticated()** - všetky ostatné URL vyžadujú prihlásenie



SecurityConfig 4/4 – SecurityFilterChain 2/5

java/sk/ukf/udemy/config/SecurityConfig.java

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .formLogin(formLogin ->
            formLogin
                .loginPage("/auth/login")
                .loginProcessingUrl("/log-in")
                .defaultSuccessUrl("/", true)
                .failureUrl("/auth/login?error=true")
                .permitAll()
        )

    // + ďalšie: logout, exceptionHandling

    return http.build();
}
```

Táto časť (**formLogin**) definuje proces prihlásenia používateľa.

Vlastná stránka pre login (naša Thymeleaf šablóna).

Spring automaticky spracuje odoslaný login formulár cez tento **endpoint**, bez nutnosti vlastného kontroléra.

Sem bude používateľ **presmerovaný po úspešnom prihlásení**.

Pri chybe prihlasovania je používateľ **presmerovaný späť na formulár**. Parameter **?error=true** umožňuje vo formulári zobrazíť chybové hlásenie.



SecurityConfig 4/4 – SecurityFilterChain 3/5

java/sk/ukf/udemy/config/SecurityConfig.java

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .formLogin(formLogin ->
            formLogin
                .loginPage("/auth/login")

                .loginProcessingUrl("/log-in")

                .defaultSuccessUrl("/", true)

                .failureUrl("/auth/login?error=true")

                .permitAll()
        )

        // + ďalšie: logout, exceptionHandling

    return http.build();
}
```

- po úspešnom prihlásení *Spring Security* vytvorí nový **SecurityContext**, ktorý obsahuje **informácie o prihlásenom používateľovi** (meno, roly)
- tento *SecurityContext* sa uloží do **serverovej HttpSession**
 - **session** je objekt na strane servera, ktorý uchováva údaje o stave používateľa počas prihlásenia
- server odošle do prehliadača **cookie** so session ID (napr. JSESSIONID)
 - **cookie** je malý súbor v prehliadači, ktorý identifikuje používateľa pri ďalších požiadavkách
- prehliadač túto cookie automaticky posiela pri každej ďalšej požiadavke
 - server vie, ktorý používateľ je aktuálne prihlásený





SecurityConfig 4/4 – SecurityFilterChain 4/5

java/sk/ukf/udemy/config/SecurityConfig.java

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        // + predošlé: authorizeHttpRequests, formLogin

        .logout(logout ->
            logout
                .logoutUrl("/logout")
                .logoutSuccessUrl("/auth/login?logout=true")
                .permitAll()
        )
        .exceptionHandling(exceptionHandling ->
            exceptionHandling
                .accessDeniedPage("/auth/error")
                .authenticationEntryPoint((request, response, authException) -> {
                    response.sendRedirect("/auth/login");
                })
        );

    return http.build();
}
```

URL adresa, ktorá spúšťa **odhlásenie** používateľa. Spring Security túto požiadavku spracuje automaticky.

Po úspešnom odhlásení bude používateľ presmerovaný na **login stránku**.
Parameter **?logout=true** umožní zobrazíť správu

Logout URL je **prístupná pre všetkých**, no samotné odhlásenie vykoná len používateľ, ktorý je aktuálne prihlásený.

Ak sa neprihlásený používateľ pokúsi vstúpiť na **chránenú URL** bude presmerovaný na login stránku **/auth/login**.

Ak sa používateľ pokúsi vstúpiť na stránku, na ktorú **nemá oprávnenie**, bude presmerovaný na stránku **/auth/error**.



SecurityConfig 4/4 – SecurityFilterChain 5/5

java/sk/ukf/udemy/config/SecurityConfig.java

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        // + predošlé: authorizeHttpRequests, formLogin

        .logout(logout ->
            logout
                .logoutUrl("/logout")
                .logoutSuccessUrl("/auth/login?logout=true")
                .permitAll()
        )
        .exceptionHandling(exceptionHandling ->
            exceptionHandling
                .accessDeniedPage("/auth/error")
                .authenticationEntryPoint((request, response, authException) -> {
                    response.sendRedirect("/auth/login");
                })
        );

    return http.build();
}
```

- pri odhlásení Spring Security **invaliduje aktuálnu session**
 - server odstráni všetky údaje spojené s prihláseným používateľom a session ID prestane byť platné
- tým sa zruší aj uložený **SecurityContext**, takže používateľ už **nie je autentifikovaný**
- ak sa pokúsi otvoriť chránenú stránku, **musí sa znova prihlásiť**, aby získal novú session



UserDetailsServiceImpl

- pri autentifikácii potrebujeme získať údaje o používateľovi
- predvolené rozhranie **UserDetailsService** nevie odkiaľ tieto dáta čítať, preto je potrebné vytvoriť vlastnú implementáciu **UserDetailsServiceImpl**

java/sk/ukf/udemy/config/UserDetailsServiceImpl.java

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmailWithRoles(email)
            .orElseThrow(() ->
                new UsernameNotFoundException("Používateľ s emailom " + email + " nebol nájdený"));

        return new UserPrincipal(user);
    }
}
```

Áno, je to vrstva Service, ale ... konfiguračná.

To nám umožňuje načítať používateľa podľa **emailu** a získať používateľa spolu s jeho **rolami** (**findByEmailWithRoles**).

Pozor na validáciu, už viete ako na to...
Takto NIE!!!

Transformácia entity **User** do objektu **UserDetails** (opäť vlastná implementácia **UserPrincipal**), ktorý Spring Security používa pri autentifikácii a autorizácii.



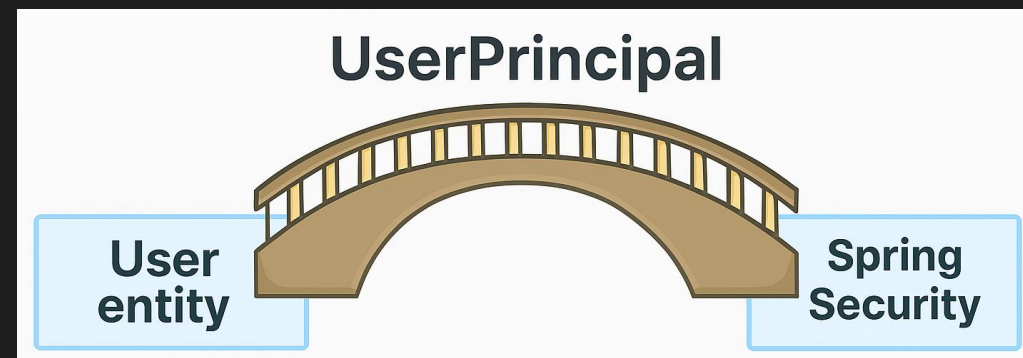
UserPrincipal ako implementácia

java/sk/ukf/udemy/config/UserPrincipal.java

```
public class UserPrincipal implements UserDetails {  
  
    private final User user;  
  
    public UserPrincipal(User user) { this.user = user; }  
  
    @Override  
    public Collection<? extends GrantedAuthority> getAuthorities() {  
        return user.getRoles().stream()  
            .map(role -> new SimpleGrantedAuthority(role.getName()))  
            .collect(Collectors.toSet());  
    }  
  
    public String getInitials() {  
        String firstName = user.getFirstName();  
        String lastName = user.getLastName();  
        return (firstName.substring(0, 1) +  
            lastName.substring(0, 1)).toUpperCase();  
    }  
  
    // + ďalšie get metódy (getFirstName, getLastName, getEmail, ...)  
}
```

Spring Security pracuje s rozhraním **UserDetails**, ktoré definuje, ako má objekt **User** vyzerat' (heslo, meno, roly, stav účtu atď.).

Trieda **UserPrincipal** obaľuje entitu **User**, implementuje rozhranie **UserDetails**, prevádza roly na objekty **GrantedAuthority** a sprístupňuje doplnkové údaje, napríklad iniciály používateľa cez metódu **getInitials()**.





UserRepository

java/sk/ukf/udemy/dao/UserRepository.java

```
public interface UserRepository extends JpaRepository<User, Integer> {  
  
    boolean existsByEmail(String email);  
  
    @Query("SELECT u FROM User u LEFT JOIN FETCH u.roles WHERE u.email = :email")  
    Optional<User> findByEmailWithRoles(@Param("email") String email);  
}
```

Jasně?

java/sk/ukf/udemy/config/UserDetailsServiceImpl.java

```
@Override  
public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {  
    User user = userRepository.findByEmailWithRoles(email)  
        .orElseThrow(() ->  
            new UsernameNotFoundException("Používateľ s emailom " + email + " nebol nájdený"));  
  
    return new UserPrincipal(user);  
}
```



UserServiceImpl 1/3

java/sk/ukf/udemy/service/UserServiceImpl.java

```
@Service
public class UserServiceImpl implements UserService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    @Autowired
    public UserServiceImpl(UserRepository userRepository, PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    ...
}
```

Spring automaticky injectne bean **PasswordEncoder**, ktorý sme definovali v triede **SecurityConfig**. Tým pádom v Service vrstve sa používa práve implementácia **BCryptPasswordEncoder**.

V triede **SecurityConfig** sme centralizovali konfiguráciu hashovania. Ak by sme chceli zmeniť algoritmus (napr. na *Argon2PasswordEncoder*), stačí zmeniť len ten **@Bean** v tej konfiguračnej triede.

Užitočná metóda: **passwordEncoder.matches()** porovná heslo zadané používateľom (plaintext) s uloženým zakódovaným heslom a vráti true, ak sa zhodujú. **Využiteľné pri zmene hesla...**





UserServiceImpl 2/3 – vloženie nového

REGISTRÁCIA

java/sk/ukf/udemy/service/UserServiceImpl.java

```
@Override  
@Transactional
```

```
public User saveUser(User incoming) {
```

```
    if (incoming.getId() == 0) {
```

Je IDčko == 0 ? Tak ide o nový záznam.

```
        if (userRepository.existsByEmail(incoming.getEmail())) {  
            throw new IllegalArgumentException("...už existuje");  
        }  
    }
```

Toto **nie je korektný spôsob validácie**,
vy poznáte ten správny...

```
    if (incoming.getUserInfo() != null) {  
        incoming.getUserInfo().setUser(incoming);  
    }
```

Ak obsahuje userInfo, prepojí ho s entitou User (obojsmerný vzťah).

```
    Role defaultRole = new Role();  
    defaultRole.setId(1);  
    incoming.setRoles(Set.of(defaultRole));  
    incoming.setPassword(passwordEncoder.encode(incoming.getPassword()));
```

Priradí predvolenú rolu (Rolu s id = 1).

```
    return userRepository.save(incoming);
```

```
}
```

```
...
```

Heslo zahashuje a uloží záznam do databázy.

id	name
1	ROLE_student
2	ROLE_teacher
3	ROLE_admin



UserServiceImpl 3/3 – úprava existujúceho

ÚPRAVA
PROFILU

java/sk/ukf/udemy/service/UserServiceImpl.java

```
User existing = userRepository.findById(incoming.getId())
    .orElseThrow(() -> new IllegalArgumentException("Používateľ neexistuje."));

if (incoming.getFirstName() != null) existing.setFirstName(incoming.getFirstName());
if (incoming.getLastName() != null) existing.setLastName(incoming.getLastName());
if (incoming.getEmail() != null) existing.setEmail(incoming.getEmail());

if (incoming.getUserInfo() != null) {
    if (existing.getUserInfo() == null) {
        existing.setUserInfo(incoming.getUserInfo());
        incoming.getUserInfo().setUser(existing);
    } else if (incoming.getUserInfo().getPhone() != null) {
        existing.getUserInfo().setPhone(incoming.getUserInfo().getPhone());
    }
}

return existing; // update sa vykoná automaticky pri commitnutí
}
```

Nájde existujúci záznam podľa ID.
Ak neexistuje, vyhodí výnimku.

Aktualizuje len tie polia,
ktoré nie sú null.

To isté aj pre telefónne
číslo, ktoré je
súčasťou UserInfo.

Roly, heslo a dátum vytvorenia
zostávajú nezmenené!!!

Objekt sa automaticky uloží pri **commitnutí transakcie** (automatické transakčné spracovanie).



AuthController 1/2

KONEČNE CONTROLLER

java/sk/ukf/udemy/controller/AuthController.java

```
@Controller
@RequestMapping("/auth")
public class AuthController {
    private final UserService userService;

    @Autowired
    public AuthController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping("/login")
    public String showLoginPage() {
        return "auth/login";
    }

    @GetMapping("/register")
    public String showRegistrationForm(Model model) {
        model.addAttribute("user", new User());
        return "auth/register";
    }
}
```

Zobrazí stránku s loginom
(auth/login.html).

Do modelu pridá nový prázdny objekt User, ktorý
sa v šablóne **viaže na formulár**.
Zobrazí registračnú stránku (auth/register.html).

```
resources
├── static
├── templates
│   └── auth
│       ├── login.html
│       └── register.html
```



AuthController 2/2

java/sk/ukf/udemy/controller/AuthController.java

```
@PostMapping("/register")
public String createUser(@ModelAttribute("user") User user,
                        BindingResult bindingResult) {
    try {
        user.setId(0);
        userService.saveUser(user);
        return "redirect:/auth/login?register";
    } catch (IllegalArgumentException e) {
        bindingResult.rejectValue("email", "email.exists", e.getMessage());
        return "auth/register";
    }
}

@GetMapping("/error")
public String showErrorMessage() {
    return "error";
}
```

Spracovanie registračného formulára, pri ktorom sa údaje z formulára automaticky mapujú na objekt User.

Ak všetko prebehne úspešne, používateľa presmeruje na login stránku s query parametrom **?register** (na zobrazenie flash správy).

Ak email už existuje, tak sa pomocou **bindingResult** pridá chybová správa ku konkrétnemu poľu email a vráti sa späť na registračný formulár.

Jednoduchý handler pre chybovú stránku – vracia šablónu error.html.

Funkcionalita úpravy profilu / zmeny hesla **nie je implementovaná**, zišlo by sa doprogramovať :)



Register.html

src/main/resources/templates/auth/register.html

```
<div xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}"
      layout:fragment="content">
```

Vkladá obsah stránky do hlavného layoutu.

```
<form th:action="@{/auth/register}" th:object="${user}" method="post">
  <h2>Registrácia</h2>
```

URL, kam sa formulár odošle po odoslaní a viaže formulár na objekt User v modeli.

```
<div th:if="${param.error}">
  <p>Registrácia zlyhala. Skúste znova.</p>
</div>
```

Zobrazí chybovú správu.

```
<label for="email">Email</label>
<input type="email" th:field="*{email}" id="email" placeholder="Zadajte email" required>
<div th:if="${#fields.hasErrors('email')}" th:errors="*{email}"></div>
```

Prepája vstupné pole s atribútom a zobrazuje validačné chyby pre konkrétne pole.

+ ďalšie vstupné polia

```
  <button type="submit">Registrovať</button>
</form>
</div>
```



Login.html

src/main/resources/templates/auth/login.html

```
<div xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}"
      layout:fragment="content">

  <form th:action="@{/log-in}" method="post">

    <div th:if="${param.error}"><p>Zadaný email alebo heslo nie sú správne.</p></div>

    <div th:if="${param.logout}"><p>Boli ste úspešne odhlásený.</p></div>

    <div th:if="${param.register}"><p>Boli ste úspešne zaregistrovaný.</p></div>

    <label for="username">Email</label>
    <input type="email" id="username" name="username" placeholder="Váš email" required>

    <label for="password">Heslo</label>
    <input type="password" id="password" name="password" placeholder="Vaše heslo" required>

    <button type="submit">Prihlásiť sa</button>
  </form>
</div>
```

Adresa, na ktorú sa formulár odošle pri prihlasovaní (v SecurityConfig).

Informujeme používateľa
na základe query
parametra.

Technický názov poľa je
username, ktorý Spring
Security očakáva.



Header.html – úprava 1/2

src/main/resources/templates/fragments/header.html

```
<header th:fragment="header"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">
  <div sec:authorize="isAnonymous()">
    <a th:href="@{/auth/login}">Prihlásiť sa</a>
    <a th:href="@{/auth/register}">Registrovat' sa</a>
  </div>
  <div sec:authorize="isAuthenticated()">
    <a href="/student/my-courses"
      sec:authorize="hasRole('ROLE_student')">Moje učenie</a>
    <a href="/teacher/my-learning"
      sec:authorize="hasRole('ROLE_teacher')">Moje vyučovanie</a>
    <span th:text="${#authentication.principal.firstName} + ' '
      + ${#authentication.principal.lastName}"></span>
    (<span th:text="${#authentication.principal.username}"></span>)
    <form th:action="@{/logout}" method="POST">
      <button type="submit">Odhlásiť sa</button>
    </form>
  </div>
</header>
```

Prihlásiť sa

Registrovat' sa

Tento blok je viditeľný len pre **neprihlásených** používateľov. Zobrazuje odkazy na prihlásenie a registráciu.

Moje učenie

Moje vyučovanie

DD

Tento blok je viditeľný len pre **prihlásených** používateľov.

Dávid Držík

ddrzik@ukf.sk

Môj profil

Nastavenia

Odhlásiť sa



Header.html – úprava 2/2

src/main/resources/templates/fragments/header.html

```
<header th:fragment="header"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">
  <div sec:authorize="isAnonymous()">
    <a th:href="@{/auth/login}">Prihlásiť sa</a>
    <a th:href="@{/auth/register}">Registrovať sa</a>
  </div>
  <div sec:authorize="isAuthenticated()">
    <a href="/student/my-courses"
      sec:authorize="hasRole('ROLE_student')">Moje učenie</a>
    <a href="/teacher/my-learning"
      sec:authorize="hasRole('ROLE_teacher')">Moje vyučovanie</a>
    <span th:text="${#authentication.principal.firstName} + ' '
      + ${#authentication.principal.lastName}"></span>
    (<span th:text="${#authentication.principal.username}"></span>)
    <form th:action="@{/logout}" method="POST">
      <button type="submit">Odhlásiť sa</button>
    </form>
  </div>
</header>
```

Tento blok je viditeľný len pre používateľov, ktorí majú **správnu rolu**. Zobrazuje odkaz na stránku s kurzami.

Zobrazuje meno a priezvisko spolu s emailom a tlačidlom na odhlásenie.

1



Hľadať kurzy...

Všetky

Názov (A-Z)

Registrácia

Krstné meno

Dávid

2

Priezvisko

Držík

Email

ddrzik@ukf.sk

Používateľ s týmto emailom už existuje.

Heslo

Zadajte heslo

Telefónne číslo

0903123456

Registovať



Struktúry a
jazyku Java



Objektové technológie

Dávid Držík

29,99 €



Webové aplikácie
platforme Java

Dávid Držík

49,99 €

Prihlásenie

Boli ste úspešne zaregistrovaný. Prihláste sa prosím.

Email

Váš email

3

Heslo

Vaše heslo

Prihlásiť sa

Používateľov otravuje, keď sa po registrácii musia prihlasovať, nemôže zostať používateľ prihlásený hneď?



Hľadať kurzy...

Všetky

Dávid Držík

ddrzik@ukf.sk

4

Môj profil

Nastavenia

Odhlásiť sa



Údajové štruktúry a
triedy v jazyku Java

Dávid Držík

16,99 €



Objektové technológie

Dávid Držík

29,99 €



Webové apli
platforme Ja

Dávid Držík

49,99 €

Prihlásenie

Boli ste úspešne odhlásený.

Email

Váš email

5

Heslo

Vaše heslo

Prihlásiť sa



Zdroje

- <https://spring.io/>
- <https://docs.spring.io/spring-security/reference/>
- <https://www.udemy.com/course/spring-hibernate-tutorial>
- <https://www.baeldung.com/spring-security-thymeleaf>
- <https://spring.io/guides/gs/securing-web>



Opakovanie

1. Načo slúžia atribúty `layout:decorate` a `layout:fragment` v Thymeleaf šablónach a aký problém rieši knižnica, ktorá ich poskytuje?
2. Aký je účel `BCryptPasswordEncoder` typu `PasswordEncoder`?
3. Popíšte zjednodušene, čo sa stane s HTTP požiadavkou na chránenú URL, ak používateľ nie je prihlásený (**Filter Chain**).
4. Načo slúži trieda `UserPrincipal` keď existuje `UserDetails`?
5. Ako server "vie", že používateľ je stále prihlásený, keď pošle novú HTTP požiadavku?
6. Ako by ste zabezpečili, aby sa chránený odkaz zobrazil iba používateľovi s konkrétnou rolou v Thymeleaf šablóne?