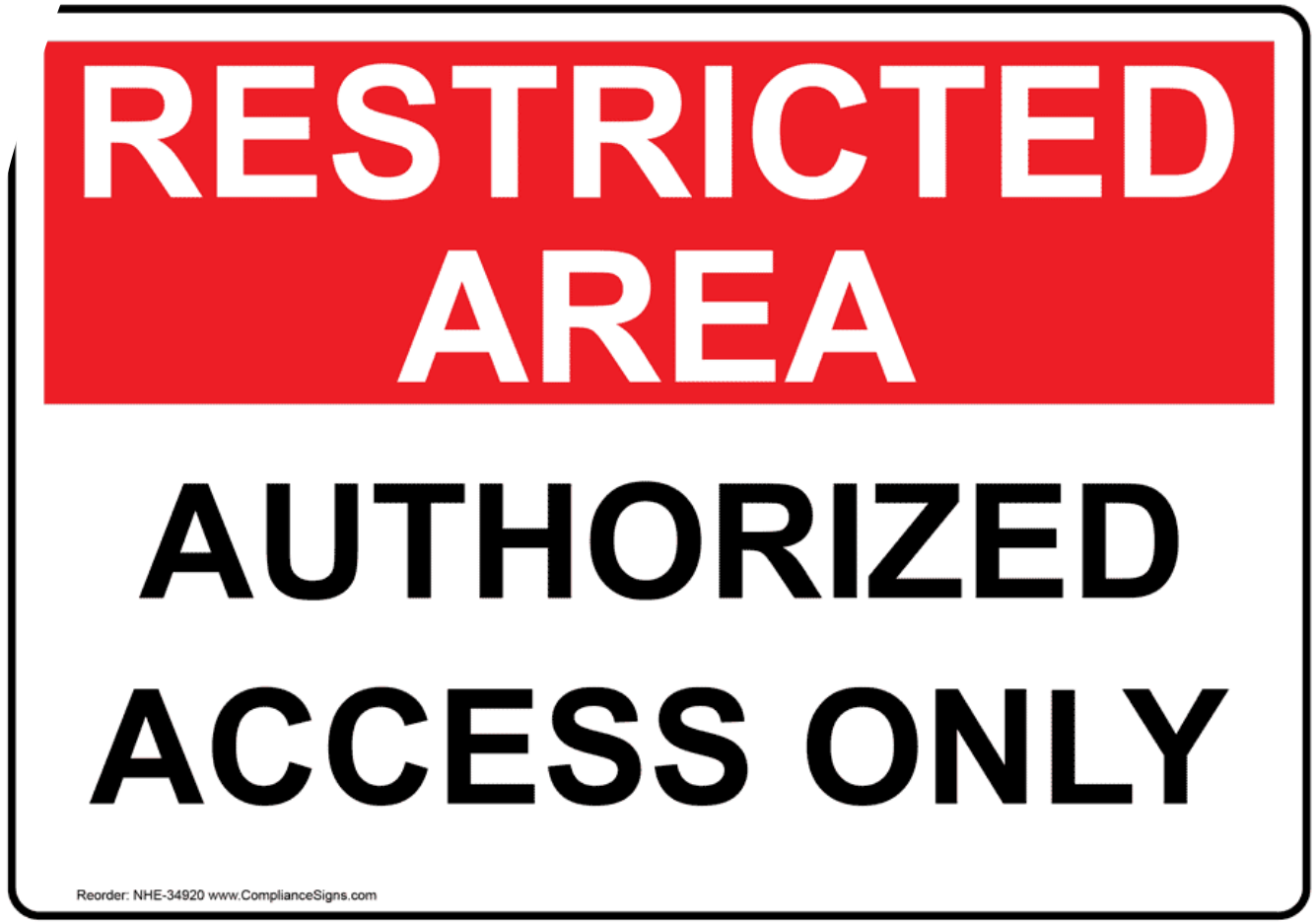
Autorizácia

Projekt 1

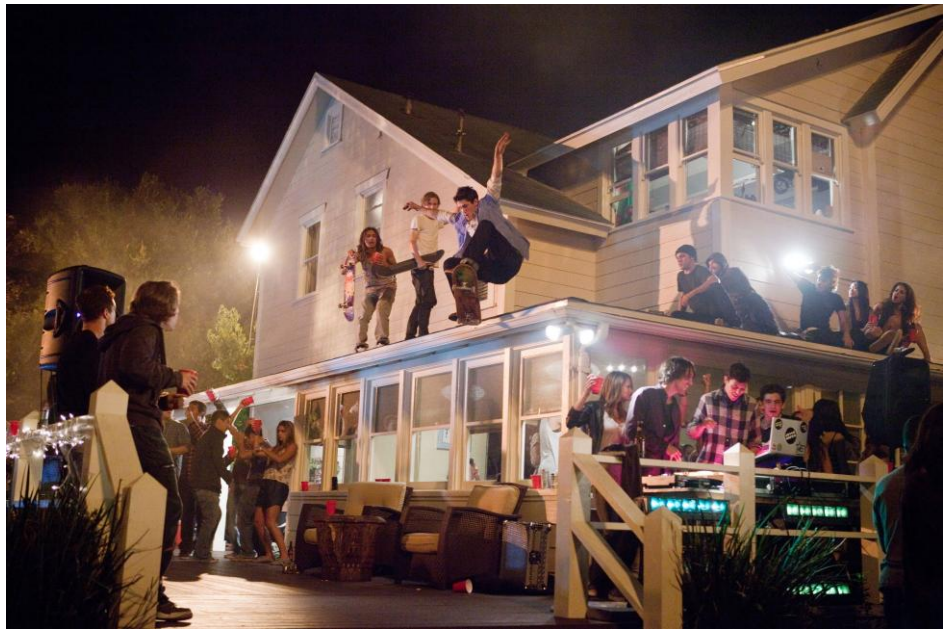


Autentifikácia

- Mám prístup do systému
- meno+heslo

Autorizácia

- Čo môžem v systéme robiť



Autorizačný model RBAC

- Role-Based Access Control
- Máme zoznam hard-coded stringov - povolení
  - Napr. v číselníkovej (enumerated) tabuľka permissions
  - Jednotlivé stringy predstavujú povolenia pre každú funkcionalitu
    - get_person, create_person, update_person, assign_chip_reader_to_person, ...
- Vytvoríme si entitu role
  - Rola predstavuje nejakú podmnožinu povolení
  - Zadefinujeme si nejaké základné role
    - admin, standard_user
  - Dáme adminom možnosť role plne editovať
    - okrem možno samotnej admin role
  - Každý kontrolér kontroluje, či rola prihláseného užívateľa obsahuje vhodné povolenie

Autorizačný model ABAC

- Attribute-Based Access Control
  - Tiež známy ako PBAC (Policy-Based), ale nie je to 100% to isté
- Podobne ako RBAC, ale namiesto povolení máme pravidlá (rules)
  - must_be_older_than_18, must_be_admin, must_be_white, ...
- Pravidlo môže byť string, komplexnejší objekt, niekedy aj plnohodnotná metóda, ktorú si môže admin nakódiť priamo vo vašej appke
- Pravidlo je plnohodnotná entita, má mať CRUD
- Možno ich aj zgrupovať do skupín pravidiel (niečo ako rola – teda tiež entita)
- Pravidlá prideľujeme užívateľom, ale aj **komponentom** (tlačidlám, ...) a/alebo **REST endpointom** cez GUI
- Každý kontrolér musí
  - zistiť, aké komponenty/REST endpointy majú pridelené ktoré pravidlá
  - pri akcii si nájsť vhodné pravidlo a zavolať niečo v zmysle ABAC.check(rule, user)
  - ABAC.check musí vhodne interpretovať pravidlo a skontrolovať, či atribúty usera sedia s pravidlom

Keycloak (KC) hierarchia

- Ríša (realm)
  - Je najvyšší objekt v Keycloaku
  - Napr. celá naša organizácia (UPJŠ)
- Ríša ma klientov (clients)
  - Klienti predstavujú jednotlivé aplikácie
  - Napr. AiS, Outlook 365, Jedáleň, **Entrance**, ...
- Ríša ma užívateľov
  - Užívateľ nemôže robiť v KC nič pokiaľ nemá správnu rolu
  - Role sú jednak interné pre KC
  - Pre naše aplikácie si v KC definujeme vlastné role

Role

- Užívatelia v Keycloaku môžu mať role
- KC má svoje role
  - Dovoľujú prístup do vnútra KC
- Môžeme si vytvárať vlastné role
  - Rola je iba string + voliteľné atribúty
    - Napr. admin, normal_user,
- Rolu priradíme užíveteľom
  - Aplikácia pozerá do JWT na role a rozhoduje sa čo dovolí a čo nie

Skupiny rolí

- Môžeme mať veľa rolí
  - Pre každú funkcionalitu
  - Napr. assign_card_readers, create_card_readers, view_users, create_users, ...
- Vytvoríme si skupinu rolí (group)
  - Napr. admins, normal_users, observers
  - Adminom dáme všetky role
  - „Normálnym“ používateľovi iba niektoré
  - „Pozorovateľom“ dáme iba role ako view_*, alebo read_*
- Užívateľov následne priradíme do skupín
- Autorizačný model **RBAC**

Skupiny rolí

- Môžeme mať veľa rolí

Pozor. Keycloak má trocha neštandardné názvoslovie pre RBAC.

- Pri RBAC máme **permissions**, čo sú jednoduché stringy predstavujúce povolenia pre jednotlivé funkcionality v našej appke. Tie potom môžeme zgrupovať to entity **rola** a tie priradiť jednotlivým užívateľom.
- V Keycloak ale máme **role**, čo sú jednoduché stringy, ktoré vieme zgrupovať do **role groups**. Užívateľom vieme priraďovať grupy aj jednotlivé role.
- Odteraz sa budem riadiť názvoslovím Keycloaku

Ríšske role a klientské role

- Role môžeme vytvárať na úrovni
  - Ríše (realm_roles)
  - Klienta (client_roles)
- Úrovne sú samostatné a v JWT sú v rozličných atribútoch
- Ríšska rola je spoločná pre všetkých klientov (aplikácie)
- Klientská rola je iba pre jednu aplikáciu

Klientský rámec

- Klientské role môžeme ešte priraďovať do klientských rámcov (Client Scope)
- Ak vytvoríme rámec a dáme do neho rolu
- Tak iba používateľ s danou rolou sa môže prihlásiť do tej konkrétnej aplikácie
- Môžeme mať teda používateľov, ktorí sa môžu prihlásiť napr. do AiS a do Outlooku, ale nie do Jedálne, alebo do Entrance

Príklad

Jednoduché rísske role

Vytvorme si role

- Vytvorme si role **admin** a **normal_user**
- Admin bude môcť všetko
- Normálny používateľ bude môcť
  - Zmeniť si meno, heslo
  - Vidieť svoje prístupy k čítačkám (card_readers)
  - Vidieť iba seba, nie ostatných
- Role si vytvoríme v KC
  - V UI, alebo lepšie rovno v init_keycloak.json
- Ale funkcionalitu/význam im dáme až v našej appke

Vytvorme si role

```
{
  "realm": "entrance",
  ...
  "roles": {
    "realm": [
      {"name": "admin"},
      {"name": "user"}
    ]
  },
  "users": [
    {
      "username": "entrance-admin",
      ...
      "realmRoles": [
        "admin"
      ]
    },
    ...
}
```

Role na úrovni ríše

Priraďme užívateľovi role

Role v Spring Boot

- Vieme, že role užívateľa budú v JWT
- JWT ale nemajú pre role jednotný formát
- Musíme si role z JWT vytiahnuť sami a prekonvertovať ich do objektov typu GrantedAuthority

Role v Spring Boot

src/main/.../configs/KeycloakJwtTokenConverter.java

```
public class KeycloakJwtTokenConverter implements Converter<Jwt, JwtAuthenticationToken > {
    ...

   @Override
   public JwtAuthenticationToken convert(@NonNull Jwt jwt) {
      var realmAccess = jwt.getClaimAsMap("realm_access");
      var roles = (Collection<String>) realmAccess.get("roles");

      var authorities = roles.stream()
            .map(SimpleGrantedAuthority::new)
            .toSet();

      ...
      return new JwtAuthenticationToken(jwt, authorities, principalClaimName);
   }
}
```

Zjednodušme si JWT

- Z triedy JwtAuthenticationToken sa role vyťahujú zložito
  - Napr. kontrola, či užívateľ je admin
    - getAuthorities().stream().anyMatch(a -> a.getAuthority().equals("admin"))
  - Musíme to robiť často, v každom Controller-i viackrát
- Rozšírme si teda tú triedu a naučme ju čo chceme

```
public class EntranceUser extends JwtAuthenticationToken {

    public EntranceUser(Jwt jwt, Collection<? extends GrantedAuthority> authorities, String name) {
        super(jwt, authorities, name);
    }

    public boolean isAdmin() {
        return getAuthorities().stream().anyMatch(a -> a.getAuthority().equals("admin"));
    }

    public String getEmail() {
        return (String) getToken().getClaims().get("email");
    }
}
```

Použime EntranceUser v JWT konverteri

- Naučme Spring Boot aby si surové JWT previedol na nášho EntranceUser-a

src/main/.../configs/KeycloakJwtTokenConverter.java

```
public class KeycloakJwtTokenConverter implements Converter<Jwt, EntranceUser> {
    ...

   @Override
   public EntranceUser convert(@NonNull Jwt jwt) {
       ...
      return new EntranceUser(jwt, authorities, principalClaimName);
   }
}
```

Použime EntranceUser v Controller-i

- Do každej HTTP metódy s @*Mapping v triede s @RestController môžeme dať parameter typu **EntranceUser** a použiť ho

```
@RestController
public class PersonController {
   ...
   @GetMapping("/persons")
   public Page<Person> list(EntranceUser entranceUser, ...) {
      if (entranceUser.isAdmin()) {
         var pagination = PageRequest.of(page.orElse(0), count.orElse(Integer.MAX_VALUE));
         return personRepository.findAll(pagination).map(Person::new);
      }

      var person = personRepository.findByEmail(entranceUser.getEmail());
      if (person == null) {
         return new PageImpl<>(List.of());
      }
      return new PageImpl<>(List.of(person.get()));
   }
}
```

Admin môže vidieť všetkých používateľov

Nie-admin môže vidieť iba seba

Ukážka

- Prihlásme sa do KC ako admin.
- Vytvorme si používateľa v KC s menom emailom lucka@upjs.sk
  - Dajme jej aj nejaké heslo
- Prihlásme sa do FE ako lucka@upjs.sk.
  - Neuvidíme nič.
- Prihlásme sa do FE ako entrance-admin
  - Uvidíme veľa osôb, ale nie Lucku
- Vytvorme v FE osobu s emailom lucka@upjs.sk
- Prihlásme sa znova do FE ako lucka@upjs.sk.
  - Uvidíme jednu osobu - Lucku

Ďakujem za pozornosť