Autorizácia, integrácia užívateľov

Projekt 1

Prednáška 7

Autentifikácia

- Mám prístup do systému
- meno+heslo

Autorizácia

- Čo môžem v systéme robiť

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

- 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
- Autorizačné modely **RBAC**, ABAC, PBAC, ...

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": “normal_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

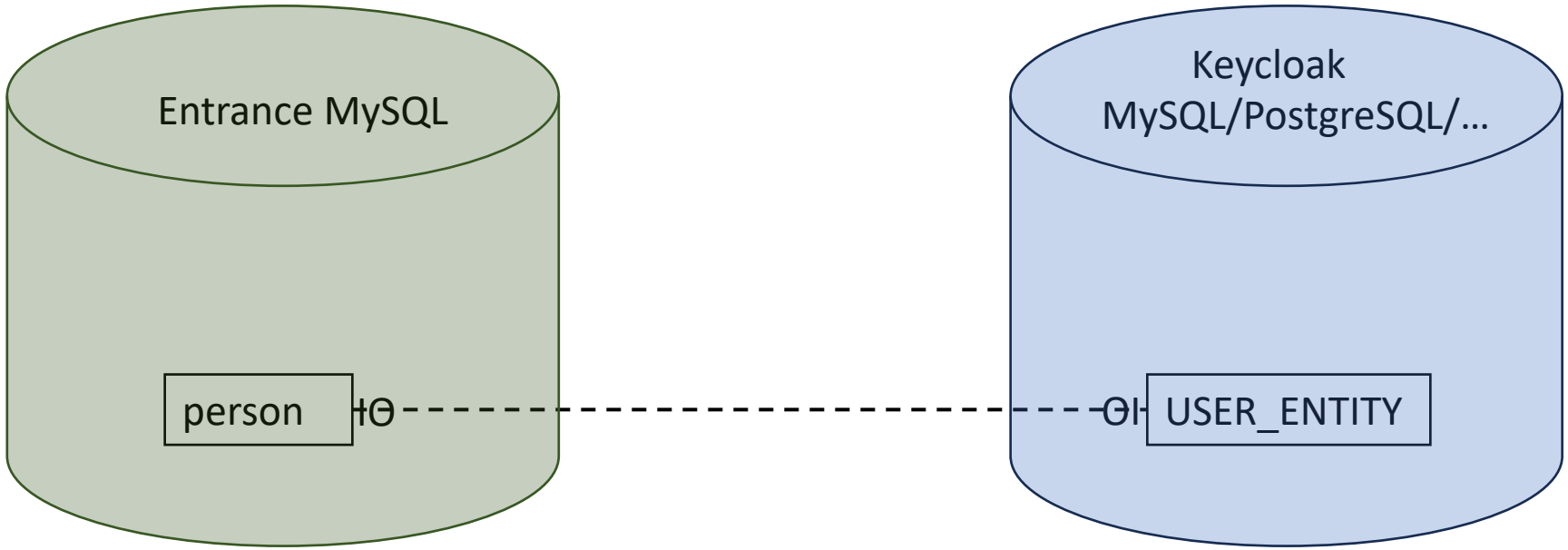
- Prihlásme sa do KC ako admin.
- Vytvorme si používateľa v KC s menom emailom lucka@upjs.sk
  - Dajme jej aj nejaké heslo
- Prihlásme sa do FE ako lucka@upjs.sk.
  - Neuvidíme nič.
- Prihlásme sa do FE ako entrance-admin
  - Uvidíme veľa osôb, ale nie Lucku
- Vytvorme v FE osobu s emailom lucka@upjs.sk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

Príklad

Integrované vytváranie používateľov

Manažment užívateľov

- Keycloak slúži ako centrálna databáza užívateľov (user)
  - ... pre aplikácie (klientov) v organizácii (ríši)
- Aplikácie majú svoje vlastné databázy s tabuľkou používateľov (napr. tabuľka person)



Manažment užívateľov

- Potrebujeme párovanie medzi KC **user**-mi a Entrance **person**-ami
- Teraz admin môže vytvárať person v Entrance a následne ručne v vytvoriť user-a v KC s tým istým emailom, s rolou normal_user a nejakým heslom.
- Potrebujeme vhodný „cudzí kľúč“ medzi entitami person a user
  - ID nemôžeme použiť, lebo Entrance aj Keycloak si ID generujú samostatne
  - Použime atribút **email**
    - V Entrance je email unikátny pre person-u
    - V KC máme email unikátny pre user-a **(POZOR**, musí to byť zapnuté na úrovni ríše)

Integrujme vytváranie užívateľov v React-e

- Nahrajme knižnicu na parsovanie JWT
  - npm install jwt-decoder
- Pomocná funkcia na zistenie, či prihlásený používateľ je admin

```
export function userIsAdmin(user: User | null | undefined):
boolean {
    if (!user) {
        return false
    }

    const access_token = jwtDecode<any>(user.access_token)
    const realmAccess = access_token["realm_access"]
    if (!realmAccess) {
        return false
    }
    const roles = realmAccess.roles as string[]
    if (!roles) {
        return false
    }

    return roles.includes("admin")
}
```

Vytvárať používateľov môže iba admin

typescript/src/persons/personList/PersonList.tsx

```
...
function PersonList() {
   const auth = useAuth();
   const isAdmin = userIsAdmin(auth.user)
    ...
   return (
      <>
         <TableContainer component={Paper}>
         ...
         </TableContainer>
         {isAdmin ? <IconButton color='primary' size="large"
                        onClick={() => navigate('/add-person')}><AddIcon/></IconButton> : null}
      </>
   );
   ...
}
```

Vytváranie používateľa v React-e

- Heslo nového používateľa musíme zadať už v našej appke

typescript/src/persons/PersonAdd.tsx

```
export function PersonAdd() {
  ...
  return (
    <Stack spacing={2}>
      ...
      <TextField
        fullWidth
        margin="normal"
        label="Password"
        name="password"
        value={person.password}
        type="password"
        onChange={(e) => {
          setPerson((currPerson) => ({...currPerson, password: e.target.value} as Person));
        }}
      />
      ...
    </Stack>
  );
}
```

typescript/src/persons/person.tsx

```
export type Person = {
  id?: number;
  name: string;
  email: string;
  active: boolean;
  gender?: Gender;
  cardReaders: CardReader[];

  password?: string;
}
```

Integrujme vytváranie užívateľov v Spring Boot-e

- React pri vytváraní entity person posiela do Javy aj atribút password
- Ale v Spring Boot-e trieda Person nemá inšt. premennú password
  - Žiaden problém, veď ju pridajme. Či?
  - Veľký problém, lebo password patrí do KC a nie do našej DB (trieda Person = MySQL tabuľka person)

.../controllers/PersonController.java

```
@PostMapping("/persons")
@ResponseStatus(HttpStatus.CREATED)
public Person create(..., @RequestBody Person person) {
  ...
}
```

.../entities/Person.java

```
@Data
@Entity
@Table(name = "person")
public class Person {
   ....
   private String password;
   ....
}
```

Data Transfer Object

- Entita person z perzistentnej vrstvy (Person.java + MySQL) vyzerá ináč ako potrebujeme na strane API vrstvy (PersonController.java + React)
- Navyše potrebujeme entitu Person aj s password-om nejako dostať do Keycloaku (t.j. entita USER_ENTITY v úplne inej DB)
- Univerzálne DTO objekty, ktoré potom prevádzame na entity

```
@Data
public class PersonDto {
   ...
   private String password;

   public PersonDto() {}

   public PersonDto(Person person) {   ...   }

   public Person toEntity() {  ...   }

   public enum Gender {  ...  }
}
```

Skopírujeme všetky inštančné premenné z Person.java

Pridáme potrebné nové inšt. premenné

Prázdny konštruktor pre JSON mapovanie

Konvertovanie z entity->dto a naopak

Pre „úplnosť“ by sme mali aj skopírovať aj Gender (a spraviť aj CardReaderDto)

Integrujme vytváranie užívateľov v Spring Boot-e (pokus 2)

.../controllers/PersonController.java

```
@PostMapping("/persons")
@ResponseStatus(HttpStatus.CREATED)
public PersonDto create(EntranceUser entranceUser, @RequestBody PersonDto personDto) {
    if (!entranceUser.isAdmin()) {
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, "Only admins can create persons");
    }

    if (personDto.getPassword() == null || personDto.getPassword().isBlank()) {
        return new PersonDto(personRepository.save(personDto.toEntity()));
    }

    ...
}
```

Iba admin môže vytvárať nových používateľov

Ak nenastavujeme heslo, tak iba uložíme do DB. T.j. nový užívateľ sa nebude môcť automaticky prihlásiť. (súčasný stav)

V opačnom prípade vytvoríme užívateľa aj v KC s heslom. T.j. nový užívateľ sa bude vedieť prihlásiť ako normal_user.

Keycloak Admin Client

- KC má vlastné administračné REST API pre programatické manažovanie užívateľov
- Zabalené v knižniciach pre Javu a JavaScript (+ neoficiálne pre Python)
  - V iných jazykoch používať surový HTTP klient

pom.xml

```
<dependency>
   <groupId>org.keycloak</groupId>
   <artifactId>keycloak-admin-client</artifactId>
   <version>24.0.1</version>
</dependency>
```

POZOR: Verzia musí byť identická s KC serverom (viď Docker obraz pre KC)

Nakonfigurujme KC klienta

- Potrebujeme v ríši zaregistrovať nového klienta pre našu Spring Boot appku

init_keycloak.json

```
{
  "realm": "entrance-be",
  ...
  "clients": [
     {
       "clientId": "entrance-be",
       "secret": "CHANGE_ME",
       "enabled": true,
       "directAccessGrantsEnabled": true
     }
  ]
}
```

Nakonfigurujme KC klienta

- A taktiež aj nového používateľa so správnou rolou

init_keycloak.json

```
...
{
 "username": "entrance-realm-admin",
 "email": "entrance-realm-admin@entrance.com",
 "firstName": "entrance-realm-admin",
 "lastName": "entrance-realm-admin",
 "enabled": true,
 "emailVerified": true,
 "credentials": [
  {
   "type": "password",
   "value": "CHANGE_ME",
   "temporary": false
  }
 ],
 "clientRoles": {
  "realm-management": ["realm-admin"]
 }
}
...
```

Nakonfigurujme KC clienta

- Potrebujeme nakonfigurovať parametre pre KC admin knižnicu

src/main/resources/application.properties

```
...
app.keycloak.admin.url=${KEYCLOAK_URL:http://localhost:9080}
app.keycloak.admin.client.id=${KEYCLOAK_ADMIN_CLIENT_ID:entrance-be}
app.keycloak.admin.client.secret=${KEYCLOAK_ADMIN_CLIENT_SECRET:CHANGE_ME}
app.keycloak.admin.username=${KEYCLOAK_ADMIN_USERNAME:entrance-realm-admin}
app.keycloak.admin.password=${KEYCLOAK_ADMIN_PASSWORD:CHANGE_ME}
```

Vytvorme KC clienta

- Továreň pre KC klienta

src/main/java/.../configs/Config.java

```
@Bean
public Keycloak keycloak(@Value("${app.keycloak.admin.url}") String authServerUrl,
                         @Value("${app.keycloak.realm}") String realm,
                         @Value("${app.keycloak.admin.client.id}") String clientId,
                         @Value("${app.keycloak.admin.client.secret}") String clientSecret,
                         @Value("${app.keycloak.admin.username}") String username,
                         @Value("${app.keycloak.admin.password}") String password)  {
   return KeycloakBuilder
          .builder()
          .serverUrl(authServerUrl)
          .realm(realm)
          .clientId(clientId)
          .clientSecret(Objects.equals(clientSecret, "") ? null : clientSecret)
          .username(username)
          .password(password)
          .grantType(OAuth2Constants.PASSWORD)
          .build();
}
```

Musíme do DTO dodať aj mapovanie pre KC user-a a KC user-password

.../PersonDto.java

```
public UserRepresentation toUserRepresentation() {
    UserRepresentation userRepresentation = new UserRepresentation();
    userRepresentation.setUsername(email);
    userRepresentation.setEmail(email);

    var splitName = name.split(" ");
    if (splitName.length == 0) {
        userRepresentation.setFirstName("");
        userRepresentation.setLastName("");
    } else {
        userRepresentation.setFirstName(splitName[0]);
        userRepresentation.setLastName(splitName[splitName.length - 1]);
    }

    userRepresentation.setEnabled(active);
    // TODO: remove once Keycloak email integration is implemented
    userRepresentation.setEmailVerified(true);

    return userRepresentation;
}
```

Heslá užívateľov sa v KC musia ukladať samostatne v objekte typu **CredentialRepresentation**.
Aj keď UserRepresentation má setter pre atribút typu CredentialRepresentation, tak to nebude fungovať a heslo musíme uložiť v samostatnom API volaní. Prečo? Lebo medveď.

```
public CredentialRepresentation toCredentialRepresentation(boolean tempPass) {
    var credentials = new CredentialRepresentation();
    credentials.setType(CredentialRepresentation.PASSWORD);
    credentials.setValue(password);
    credentials.setTemporary(tempPass);
    return credentials;
}
```

Vytvorenie user-a v KC zo Spring Boot-u

.../controllers/PersonController.java, metóda **create**

```
...
String keycloakId;
try (var response = keycloak.realm("entrance").users().create(personDto.toUserRepresentation())) {
    if (response.getStatus() != 201) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to create user");
    }
    keycloakId = CreatedResponseUtil.getCreatedId(response);
}
...
```

Vytvorenie hesla v KC zo Spring Boot-u

.../controllers/PersonController.java, metóda **create**

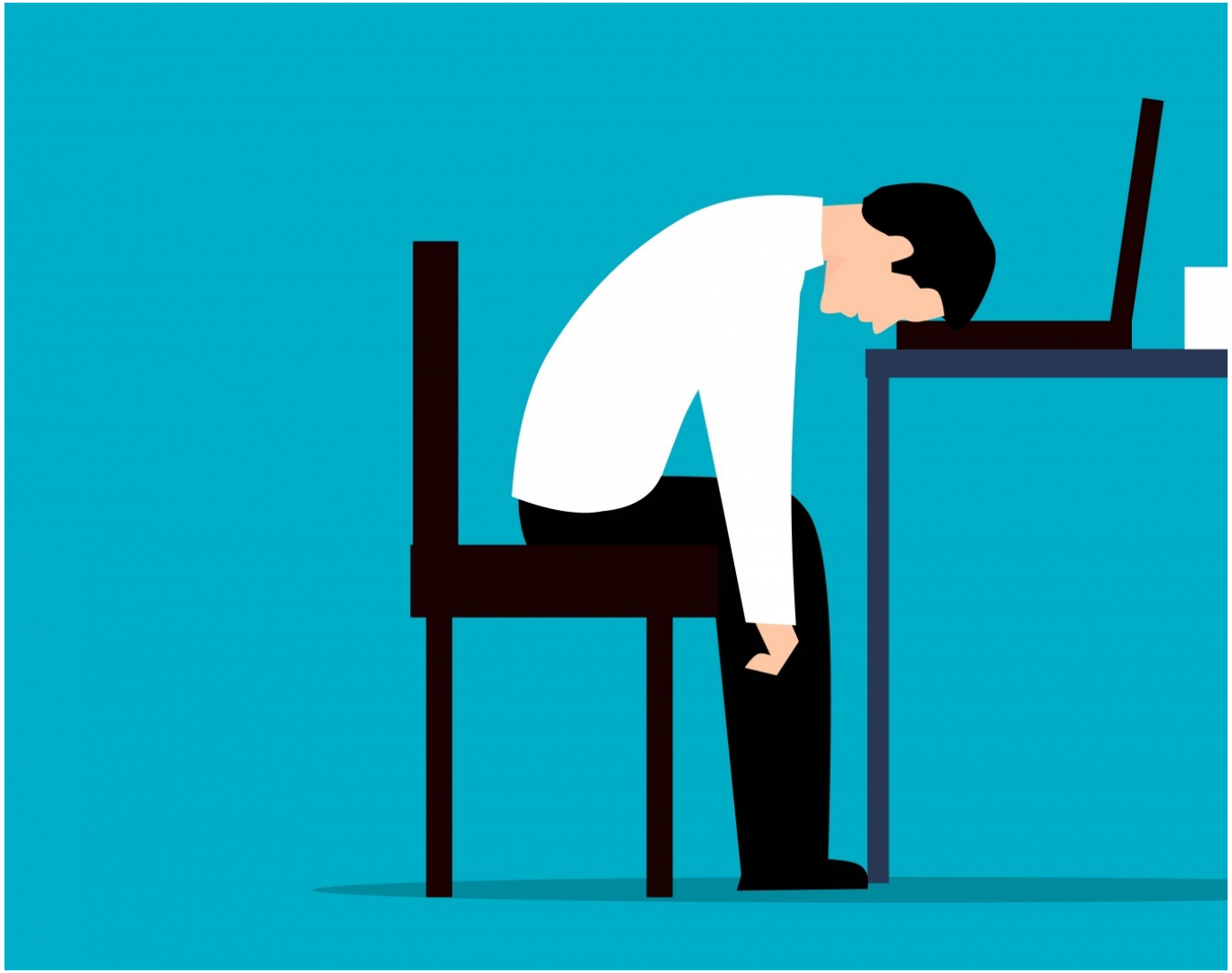
```
                                                                    Podľa interného KC ID user-a si ho vytiahneme
var keycloakUser = keycloak.realm("entrance").users().get(keycloakId);

                                                                    Nastavíme mu heslo z personDto
keycloakUser.resetPassword(personDto.toCredentialRepresentation(true));

                                                                    Nastavíme mu rolu
var role = keycloak.realm("entrance").roles().get("normal_user").toRepresentation();
keycloakUser.roles().realmLevel().add(List.of(role));


return new PersonDto(personRepository.save(personDto.toEntity()));
```

Treba ešte poriešiť update a delete



Ďakujem za pozornosť

Táto prednáška by zabila aj Chucka Norrisa