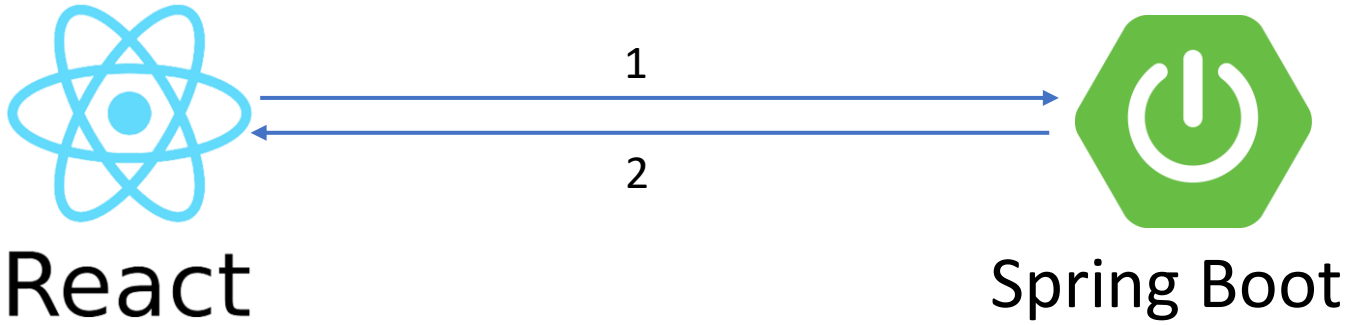
# Autentifikácia, časť 2.

**Projekt 1**

**Prednáška 5**

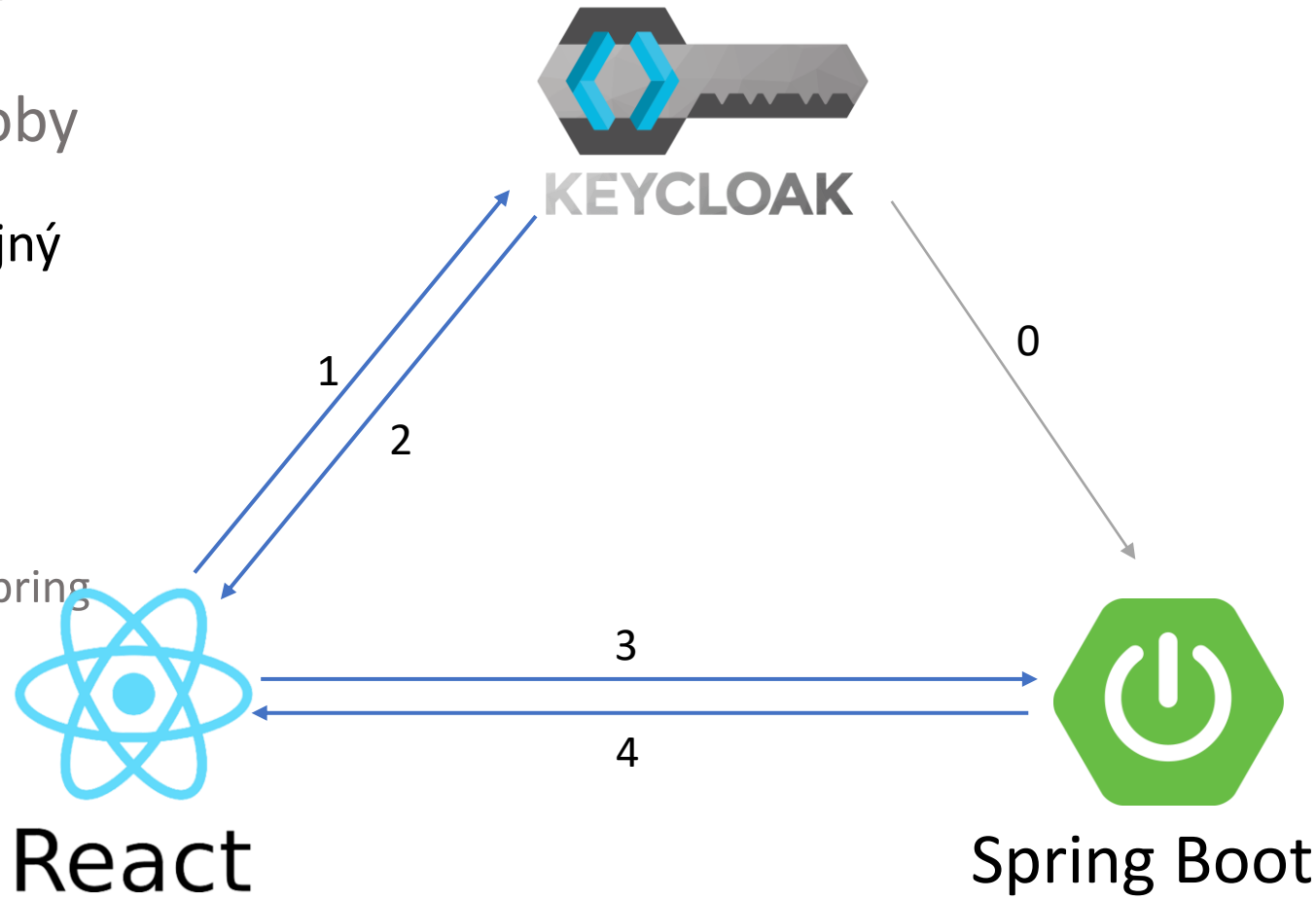
# Full-stack aplikácia

- Pre našu appku Entrance máme
  - Aplikačný server (Web UI) – React
  - Aplikačný server (REST API) – Spring Boot
- Príklad: Lucka chce zobraziť všetky osoby v Entrance
  - 1. Lucka chce v UI vykonať akciu
    - React pošle GET /persons požiadavku na Spring Boot
  - 2. Spring Boot pošle zoznam osôb



# Full-stack aplikácia s OIDC

- Máme
  - Aplikačný server (Web UI) – React
  - Aplikačný server (REST API) – Spring Boot
  - Autentifikačný server (OIDC) – Keycloak
- Príklad: Lucka chce zobraziť všetky osoby v Entrance
  - 0. Spring Boot si pri spustení vypýta verejný kľúč od Keycloaku
  - 1. React Lucku presmeruje na Keycloak
    - Tá sa tam prihlási cez meno+heslo
  - 2. Keycloak vydá JWT a pošle ho Reactu
  - 3. Lucka chce v UI vykonať akciu
    - React pošle GET /persons požiadavku na Spring Boot
    - No teraz do hlavičky pribalí JWT
  - 4. Spring Boot overí platnosť JWT
    - ...a pošle zoznam osôb



# Trocha fástforvardnime

- Potrebujeme Web UI
  - Zoberme si React appku z PAZ1c
- Potrebujem REST API
  - Zoberme si *Controller.java triedy z PAZ1c
  - Dajme ich do nášho repa
  - Prerobme z DAO na JPARepository
- Alebo použime už predpripravený projekt v PRO1a

- Všimnime si, že som v tichosti premenoval entitu User na entitu Person
  - Lebo User zvykne byť entita, ktorá sa vie prihlásiť. Ale „užívatelia“ nášho pomysleného prístupového systému nebudú mať schopnosť používať našu appku. Naša appka predstavuje iba administračnú časť tohto systému, ktorú budú používať iba admini. Preto chcem rozlíšiť, kto je užívateľ appky (*User* v Keycloak) a kto je užívateľ čipov do čítačiek (*Person*)

# Upravme si MySQL kontajner

- MySQL bude lokálne zdieľaná medzi dvoma appkami – náš Entrance a Keycloak
- Každá appka by mala mať svoj vlastný objekt DATABASE
- V reálnom nasadení by Entrance aj Keycloak používali vlastné MySQL inštancie
  - Asi by som príkazy z init.sql pre Keycloak nedával do migračných Flyway skriptov
- Nevýhoda tohto prístupu: musíme najprv zmazať kontajner (napr. cez `docker compose down`)

**docker-compose.yaml**

```
services:
  mysql:
    image: mysql:8.3.0
    environment:
      - 'MYSQL_DATABASE=entrance'
      - 'MYSQL_PASSWORD=secret'
      - 'MYSQL_ROOT_PASSWORD=verysecret ‘
      - 'MYSQL_USER=myuser'
    ports:
      - '3306:3306'
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    healthcheck:
      test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]
      interval: 10s #default is 30s
      timeout: 5s   #default is 30s
      retries: 5    #default is 3
```

**init.sql**

```
CREATE DATABASE IF NOT EXISTS `entrance`;
CREATE DATABASE IF NOT EXISTS `keycloak`;

CREATE USER IF NOT EXISTS 'myuser'@'%' IDENTIFIED BY 'secret';
CREATE USER IF NOT EXISTS 'kcuser'@'%' IDENTIFIED BY 'secret';

GRANT ALL PRIVILEGES ON `entrance`.* TO 'myuser'@'%';
GRANT ALL PRIVILEGES ON `keycloak`.* TO 'kcuser'@'%';
```

# Vložme Keycloak

**docker-compose.yaml**

```
services:
  ...
  keycloak:
    image: keycloak/keycloak:24.0.1-0
    environment:
      - KEYCLOAK_ADMIN=admin
      - KEYCLOAK_ADMIN_PASSWORD=admin
      - KC_DB=mysql
      - KC_DB_URL_HOST=mysql
      - KC_DV_URL_DATABASE=keycloak
      - KC_DB_USERNAME=kcuser
      - KC_DB_PASSWORD=secret
      - KC_HEALTH_ENABLED=true
      - KC_HTTP_PORT=9080
    command: ["start-dev", "--import-realm"]
    ports:
      - '9080:9080'
    volumes:
      - ./init_keycloak.json:/opt/keycloak/data/import/init_keycloak.json
    depends_on:
      mysql:
        condition: service_healthy
    healthcheck:
      test: ["CMD-SHELL", "exec 3<>/dev/tcp/127.0.0.1/9080;echo -e \"GET /health/ready HTTP/1.1\r\nhost: http://localhost\r\nConnection: close\r\n\r\n\" >&3;grep \"HTTP/1.1 200 OK\" <&3"]
      interval: 10s
      timeout: 5s
      retries: 20
```

# Nastavenie Keycloaku

- Každý projekt potrebuje *realm-u* (ríšu)
- Realm-a obsahuje
  - nastavenia pre náš projekt,
  - užívateľov,
  - nastavenia pre klientské aplikácie.
    - V našom prípade Web UI
- Realm-u si vieme založiť
  - vo webovom rozhraní Keycloaku,
  - v JSON formáte, ktorý si Keycloak načíta

# Budujme našu ríšu

**init_keycloak.json**

```
{
  "realm": "entrance",
  "enabled": true,
  "sslRequired": "external",
  "registrationAllowed": false,
  "users": [
    {
      "username": "entrance-admin",
      "enabled": true,
      "emailVerified": false,
      "credentials": [
        {
          "type": "password",
          "value": "entrance-admin",
          "temporary": true
        }
      ]
    }
  ],

  "clients": [
    {
      "clientId": "entrance-app",
      "secret": "CHANGE_ME",
      "rootUrl": "http://localhost:3000",
      "adminUrl": "http://localhost:3000",
      "surrogateAuthRequired": false,
      "enabled": true,
      "redirectUris": [
        "http://localhost:3000/*"
      ],
      "webOrigins": ["*"],
      "notBefore": 0,
      "bearerOnly": false,
      "publicClient": true,
      "standardFlowEnabled": true,
      "implicitFlowEnabled": false,
      "directAccessGrantsEnabled": true,
      "serviceAccountsEnabled": false,
      "authorizationServicesEnabled": true
    }
  ]
}
```

# Priebežná sumarizácia

- Spustime Docker Compose
- Máme MySQL s databázami *entrance* a *keycloak*
- Keycloak má dve realm-y
  - *master*, kde je užívateľ *admin*
    - Tento užívateľ má plný prístup k rozhraniu Keycloaku
  - *entrance*, kde je užívateľ *entrance-admin*
    - Tento užívateľ má veľmi obmedzený prístup k rozhraniu Keycloaku
    - Bude mať ale plný prístup k našej appke Entrance
    - Máme ešte klienta *entrance-app*, ktorý nastavuje OIDC pre našu appku Entrance


# Nastavme Web API

Spring Boot + OIDC

# Spring Security

**pom.xml**

```
...
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
...
```

# Nastavenie Spring Security OIDC

**src/main/resources/application.properties**

```
...

spring.security.oauth2.resourceserver.jwt.issuer-uri=${KEYCLOAK_URL:http://localhost:9080}/realms/entrance

spring.security.oauth2.resourceserver.jwt.jwk-set-uri=${spring.security.oauth2.resourceserver.jwt.issuer-uri}/protocol/openid-connect/certs

app.token.converter.principal-attribute=preferred_username # email

app.token.converter.resource-id=entrance-app
```

# Mapujme JWT

- Potrebujeme prekonvertovať surový JWT na formát pre Spring Security

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

    private final String principalAttribute;

    public KeycloakJwtTokenConverter(String principalAttribute) {
        this.principalAttribute = principalAttribute;
    }

    @Override
    public JwtAuthenticationToken convert(@NonNull Jwt jwt) {

        String claim = Objects.requireNonNullElse(principalAttribute, JwtClaimNames.SUB);
        String principalClaimName = jwt.getClaimAsString(claim);

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

Identifikovanie užívateľa podľa principalAttribute (username/email). Ak neexistuje, tak použijeme atribút SUB (user ID z Keycloaku)

# Zapnutie zabezpečenia

- Nájdeme existujúcu konfiguračnú triedu a nastavíme v nej sekjúritu

```
@Configuration
@EnableWebSecurity
public class Config {
   ...
   private final KeycloakJwtTokenConverter keycloakJwtTokenConverter;

   public Config(
         @Value("app.token.converter.principal-attribute") String keycloakPrincipalAttribute) {
      this.keycloakJwtTokenConverter = new KeycloakJwtTokenConverter(keycloakPrincipalAttribute);
   }

   @Bean
   public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
      return http
            .authorizeHttpRequests(cfg -> cfg.anyRequest().authenticated())
            .oauth2ResourceServer(cfg -> cfg.jwt(jwt -> jwt.jwtAuthenticationConverter(keycloakJwtTokenConverter)))
            .sessionManagement(cfg -> cfg.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .build();
   }

}
```

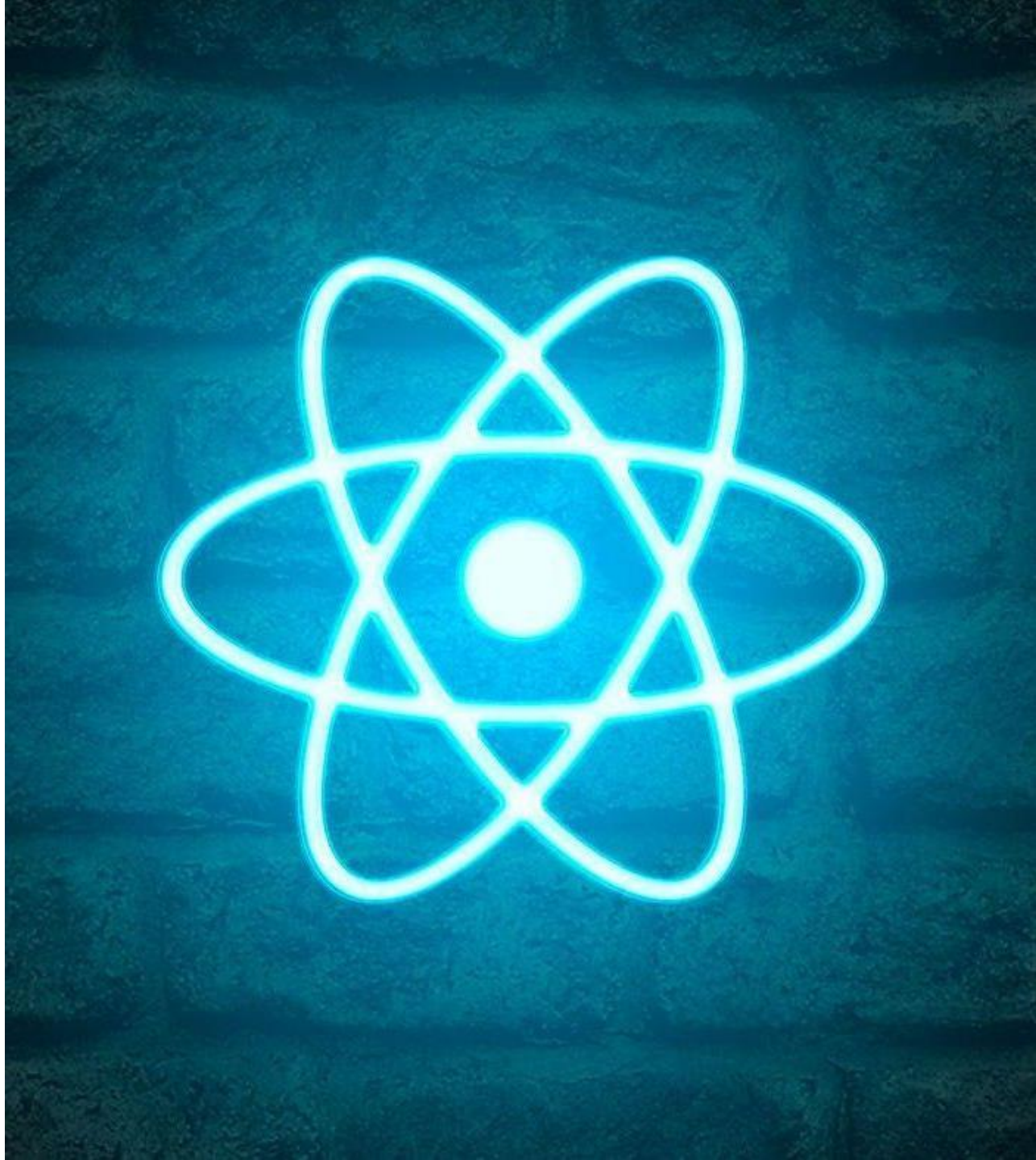
Inštancia nášho JWT konvertora

Autentifikácia pre všetky HTTP požiadavky

- REST API aplikácie by mali byť bezstavové
- Pri OIDC by si server appka nemala pamätať prihlásených užívateľov

# Nastavme Web UI

React + OIDC



# Dodajme do Reactu OIDC

- Sme v priečinku *typescript*
- Nainštalujme závislosti
  - `npm install oidc-client-ts react-oidc-context --save`

Konfigurácia pre OIDC:

**typescript/src/config.ts**

```
export const oidcAuthority = process.env.REACT_APP_OIDC_URI ? process.env.REACT_APP_OIDC_URI : 'http://localhost:9080/realms/entrance';

export const oidcSecret = process.env.REACT_APP_OIDC_SECRET ? process.env.REACT_APP_OIDC_SECRET : 'CHANGE_ME';

export const oidcRedirectURI = process.env.REACT_APP_OIDC_REDIRECT_URI ? process.env.REACT_APP_OIDC_REDIRECT_URI : 'http://localhost:3000/';

export const oidcClientId = 'entrance-app'
```

# „Zabalenie“ aplikácie do OIDC

**typescript/src/index.tsx**

```
const oidcConfig = {
   authority: oidcAuthority,
   client_id: oidcClientId,
   client_secret: oidcSecret,
   redirect_uri: oidcRedirectURI,
   personStore: new WebStorageStateStore({ store: window.localStorage }),
};

const onSigninCallback = (): void => {
   window.history.replaceState(
      {},
      document.title,
      window.location.pathname
   )
}

const root = ReactDOM.createRoot(
   document.getElementById('root') as HTMLElement
);
root.render(
   <AuthProvider {...oidcConfig} onSigninCallback={onSigninCallback}>
      <App/>
   </AuthProvider>,
);
```

Nastavenia pre OIDC klienta

- Keď navštívime adresu našej appky,
  - tak budeme presmerovaní na adresu Keycloaku
- Po prihlásení nás Keycloak presmeruje späť do našej appky
- Toto vyčistí „neporiadok“ z URL
- Ináč nebude fungovať automatické obnovenie JWT

Celú appku zabalíme do OIDC klienta

# Prihlasovanie užívateľa

**typescript/src/App.tsx**

```
function App() {
  ...
  const auth = useAuth();
  const [hasTriedSignin, setHasTriedSignin] = useState(false);

  useEffect(() => {
    if (!hasAuthParams() &&
      !auth.isAuthenticated && !auth.activeNavigator && !auth.isLoading &&
      !hasTriedSignin
    ) {
      auth.signinRedirect();
      setHasTriedSignin(true);
    }
  }, [auth, hasTriedSignin]);

  if (auth.isLoading) {
    return <div>Signing you in/out...</div>;
  }

  if (!auth.isAuthenticated) {
    return <div>Unable to log in: {`${auth.error}`}</div>;
  }
  ...
}
```

Ak nie som prihlásený, ...

..., presmeruj ma do Keycloaku

Toto zobrazuj počas procesu prihlasovania

Ak stále nie som prihlásený, tak niečo je zle

# Posielanie JWT tokenu do REST API

**typescript/src/persons/personService.ts**

```
...
export async function getPersons(accessToken: string) {
    const response = await fetch(`${restURI}/persons`, {
        headers: {
            Authorization: `Bearer ${accessToken}`
        }
    });
    const data = await response.json();
    return data.content as Person[];
}
...
```

- Analogicky upravme ostatné funkcie
- Nezabudnime na **typescript/src/cardReaders/cardReaderService.ts**

# Získanie JWT tokenu v komponente

**typescript/src/persons/PersonList.tsx**

```
function PersonList() {
    const auth = useAuth();
    const accessToken = auth.user?.access_token ? auth.user?.access_token : ""
    ...
    useEffect(() => {
        getPersons(accessToken)
            .then(persons => setPersons(persons))
            .catch(error => setError(new Error("Could not fetch persons", {cause: error})));
    }, [])
    ...
}
```

- Analogicky upravme ostatné komponenty

# Ďakujem za pozornosť

Toto je ale chutný burger.

- Ezekiel 25:17