Izvještaj (Ivan Futivić 0036522493)

Keycloak server:

- Preduvjet za korištenje Keycloaka je pokrenuti ga lokalno
    - Za pokretanje sam koristio Docker i navedenu komandu: `docker run -d -p 8080:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:21.0.1 start-dev`
    - Ovime je kreiran Keycloak server kojem se može lokalno pristupiti preko “localhost:8080”
- U Keycloaku sam kreirao novi realm pod nazivom “sosa”
    - U navedenom realmu sam kreirao novog klijenta sa client ID “sosa-client”
    - Kreirao sam jedan realm role koji se zove "admin"
    - Kreirao sam dva usera: Student i Admin
        - Student sadrži samo default role
        - Admin sadrži default role + admin role

Sprint Boot aplikacija:

- Klasa Security Config

```
@Configuration
@EnableWebSecurity
class SecurityConfig {
    2 usages
    private final JwtAuthConverter jwtAuthConverter;

    no usages  new *
    SecurityConfig(JwtAuthConverter jwtAuthConverter) { this.jwtAuthConverter = jwtAuthConverter; }

    no usages  futa125
    @Bean
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
    }

    no usages  futa125 *
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests() AuthorizeHttpRequestsConfigurer<...>.AuthorizationManagerRequestMatcherRegistry
                .antMatchers( ...antPatterns: "/records").hasRole("admin")
                .anyRequest() AuthorizeHttpRequestsConfigurer<...>.AuthorizedUrl
                .authenticated();
        http.oauth2ResourceServer() OAuth2ResourceServerConfigurer<HttpSecurity>
                .jwt() OAuth2ResourceServerConfigurer<...>.JwtConfigurer
                .jwtAuthenticationConverter(jwtAuthConverter);
        return http.build();
    }
}
```

  - Naveden kod sadrži `jwtAuthConverter` varijablu i funkciju `securityFilterChain` koja se brine oko autoriziranog pristupa
  - Funkcija securityFilterChain
    - `antMatchers` dio lanca se brine da samo korisnici sa admin role mogu pristupiti `/records` endpointu, neovisno o HTTP metodi.
    - `anyRequest()` i `authenticated` osiguravaju da svi ostali endpointovi su zaštićeni i njima mogu pristupiti samo autentificirani korisnici sa bilo kojim roleovima.
    - `oauth2ResourceServer()`, `jwt()` i `jwtAuthenticationConverter` služe za postavljanje OAuth2 resource servera i JWT konvertera koji će se brinuti oko pretvaranja “realm roles” u GrantedAuthority objekte

- JwtAuthConverter

```java
@Component
public class JwtAuthConverter implements Converter<Jwt, AbstractAuthenticationToken> {
    1 usage
    private final JwtGrantedAuthoritiesConverter jwtGrantedAuthoritiesConverter = new JwtGrantedAuthoritiesConverter();

    2 usages
    private static final Logger logger = LoggerFactory.getLogger(JwtAuthConverter.class);

    futa125 *
    @Override
    public AbstractAuthenticationToken convert(Jwt jwt) {
        Collection<GrantedAuthority> authorities = Stream.concat(
                jwtGrantedAuthoritiesConverter.convert(jwt).stream(),
                extractResourceRoles(jwt).stream()).collect(toSet());

        logger.info(authorities.toString());

        return new JwtAuthenticationToken(jwt, authorities, jwt.getClaim("preferred_username"));
    }

    1 usage  futa125 *
    private Collection<? extends GrantedAuthority> extractResourceRoles(Jwt jwt) {
        var realmAccess = new HashMap<String, Collection<String>>(jwt.getClaim("realm_access"));
        logger.info(realmAccess.toString());

        return realmAccess.values().isEmpty() ? emptySet() : realmAccess
                .get("roles") Collection<String>
                .stream() Stream<String>
                .map(r -> new SimpleGrantedAuthority( role: "ROLE_" + r)) Stream<SimpleGrantedAuthority>
                .collect(toSet());
    }
}
```

  - Funkcija `extractResourceRoles`
    - Iz našeg JWT tokena izvlačimo claim “realm_access”.
    - Iz HashMape izvlačimo vrijednost pod ključem “roles”.
    - Za svaki role kreiramo SimpleGrantedAuthority objekt koji sadrži naš realm role sa “ROLE_” prefiksom.
  - Funkcija `convert`
    - Pretvaramo naš JWT token u kolekciju GrantedAuthority objekata (dodajemo “SCOPE_” prefix svakom scopeu) i spajamo zajedno s kolekcijom GrantedAuthority objekata generiranih iz naših realm roleova.
    - Vraćamo novi JWT token koji sadrži novi skup GrantedAuthority objekata koji uključuje naše realm roleove

- application.yaml

```yaml
spring:
  application:
    name: sosa-client
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8080/realms/sosa
      client:
        registration:
          keycloak:
            client-id: sosa-client
            authorization-grant-type: authorization_code
            scope:
              - openid
        provider:
          keycloak:
            issuer-uri: http://localhost:8080/realms/sosa
            user-name-attribute: preferred_username

keycloak:
  realm: sosa
  resource: sosa-client
  auth-server-url: http://localhost:8080

server:
  port: 8082
```

  - Dodao sam potrebne podatke za Keycloak i Spring/OAuth2 (client ID, issuer URI, grant type, scope...) te sam postavio port na 8082.

Testiranje

- Za testiranje sam kreirao Python skriptu koja s korisnicima Student i Admin radi requestove prema svakom endpointu.
- Skripta sadrži assertove pomoću kojih provjerava vraća li svaki request prikladan status kod (403 ako je zabranjen pristup, 200 ako je dozvoljen)
-

```
Student access token: eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJZclZwWjd2QXpaTnJGc3I5TWp1N2xFdnJ3bTVP

Admin access token: eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJZclZwWjd2QXpaTnJGc3I5TWp1N2xFdnJ3bTVPWW

{'id': 1, 'userName': 'abc', 'passWord': 'abc'}
Student created student abc

{'id': 2, 'userName': 'def', 'passWord': 'def'}
Admin created student def

no response
Student couldn't create record abc

{'id': 3, 'recordName': 'def', 'recordDate': '01-01-2000', 'recordOrigin': 'def', 'recordValue': 'def'}
Admin created record def

[{'id': 1, 'userName': 'abc', 'passWord': 'abc'}, {'id': 2, 'userName': 'def', 'passWord': 'def'}]
Student fetched all students

[{'id': 1, 'userName': 'abc', 'passWord': 'abc'}, {'id': 2, 'userName': 'def', 'passWord': 'def'}]
Admin fetched all students

no response
Student couldn't fetch all records

[{'id': 3, 'recordName': 'def', 'recordDate': '01-01-2000', 'recordOrigin': 'def', 'recordValue': 'def'}]
Admin fetched all records
```