java_springboot_restful_08_OAuth2_Walkthrough

Swagger UI dependency in pom.xml

```
<!-- Adding OpenAPI support -->
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
   <version>2.6.0</version>
</dependency>
```

In folder -> config -> SwaggerConfig.java

```
@Configuration
@OpenAPIDefinition(info = @Info(title = "Demo API", version = "Verions 1.0",
contact = @Contact(name = "StudyEasy", email = "admin@studyeasy.org", url =
"https://studyeasy.org"), license = @License(name = "Apache 2.0", url =
"https://www.apache.org/licenses/LICENSE-2.0"), termsOfService =
"https://studyeasy.org/", description = "Spring Boot RestFul API Demo by
Chaand"))
public class SwaggerConfig {
}
```

@OpenAPIDefination -> Update the look and feel of the API.

This is were our application starts -> SpringRestdemoApplication.java

```
package org.studyeasy.SpringRestdemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import io.swagger.v3.oas.annotations.security.SecurityScheme;
import io.swagger.v3.oas.annotations.enums.*;

@SpringBootApplication
@SecurityScheme(name = "studyeasy-demo-api", scheme = "bearer", type =
SecuritySchemeType.HTTP, in = SecuritySchemeIn.HEADER)
public class SpringRestdemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringRestdemoApplication.class, args);
    }

}
```

In AccountController.java

```
    @GetMapping("/test")
    @Tag(name = "Test", description = "The Test API.")
    @SecurityRequirement(name = "studyeasy-demo-api")
    public String test(){
        return "Test Api";
    }
```

@Tag -> Name of the API in the frontend
@SecurityRequirement -> Security for the access from backend for that particular method

This was all about swagger

SecurityConfig.java

```
package org.studyeasy.SpringRestdemo.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.ProviderManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurit
y;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.security.oauth2.jwt.JwtEncoder;
import org.springframework.security.oauth2.jwt.NimbusJwtDecoder;
import org.springframework.security.oauth2.jwt.NimbusJwtEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

import com.nimbusds.jose.JOSEException;
import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.JWKSource;
import com.nimbusds.jose.proc.SecurityContext;

@Configuration
```

```java
@EnableWebSecurity
public class SecurityConfig {

    private RSAKey rsaKey;

    @Bean
    public JWKSource<SecurityContext> jwkSource() {
        rsaKey = Jwks.generateRsa();
        JWKSet jwkSet = new JWKSet(rsaKey);
        return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);
    }

    @Bean
    public InMemoryUserDetailsManager users() {
        return new InMemoryUserDetailsManager(
                User.withUsername("chaand")
                        .password("{noop}password")
                        .authorities("read")
                        .build());
    }

    @Bean
    public AuthenticationManager authManager(UserDetailsService
userDetailsService) {
        var authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        return new ProviderManager(authProvider);
    }

    @Bean
    JwtEncoder jwtEncoder(JWKSource<SecurityContext> jwks) {
        return new NimbusJwtEncoder(jwks);
    }

    @Bean
    JwtDecoder jwtDecoder() throws JOSEException {
        return NimbusJwtDecoder.withPublicKey(rsaKey.toRSAPublicKey()).build();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
                .authorizeHttpRequests(authorize -> authorize
                        .requestMatchers("/token").permitAll()
```

```
                        .requestMatchers("/").permitAll()
                        .requestMatchers("/swagger-ui/**").permitAll()
                        .requestMatchers("/v3/api-docs/**").permitAll()
                        .requestMatchers("/test").authenticated())
                .oauth2ResourceServer(oauth2 -> oauth2
                        .jwt(Customizer.withDefaults()))
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(csrf -> csrf.disable())
                .headers(headers -> headers
                        .frameOptions(frameOptions -> frameOptions.disable()));
        return http.build();
    }

}
```

Notes

In order to get this, you need to add dependency in the pom.xml

```
.oauth2ResourceServer(oauth2 -> oauth2
                        .jwt(Customizer.withDefaults()))
```

Dependency

```
<!-- Adding Spring Security OAuth2 Resource Server support -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

We are making changes and overriding Authentication Manager

```
 @Bean
    public AuthenticationManager authManager(UserDetailsService
userDetailsService) {
    var authProvider = new DaoAuthenticationProvider();
    authProvider.setUserDetailsService(userDetailsService);
    return new ProviderManager(authProvider);
}
```

TokenService.java

```
package org.studyeasy.SpringRestdemo.service;

import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.stream.Collectors;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.oauth2.jwt.JwtClaimsSet;
import org.springframework.security.oauth2.jwt.JwtEncoder;
import org.springframework.security.oauth2.jwt.JwtEncoderParameters;
import org.springframework.stereotype.Service;

@Service
public class TokenService {

    private final JwtEncoder encoder;

    public TokenService(JwtEncoder encoder) {
        this.encoder = encoder;
    }

   //Encoding the token based on our authentication management
    public String generateToken(Authentication authentication) {
        Instant now = Instant.now();
        String scope = authentication.getAuthorities().stream()
                .map(GrantedAuthority::getAuthority)
                .collect(Collectors.joining(" "));
        JwtClaimsSet claims = JwtClaimsSet.builder()
                .issuer("self")
                .issuedAt(now)
                .expiresAt(now.plus(1, ChronoUnit.HOURS))
                .subject(authentication.getName())
                .claim("scope", scope)
                .build();
        return
this.encoder.encode(JwtEncoderParameters.from(claims)).getTokenValue();
    }

}
```

Authentication object -> This gives you the authentication object itself; the user.

In Order to use Token system, we need to use Encoder and Decoder

In Order to use in spring, it is recommended to make use of Asymmetric Key which is nothing but a pair.

Below is an encoder and decoder

```
 @Bean
  JwtEncoder jwtEncoder(JWKSource<SecurityContext> jwks) {
     return new NimbusJwtEncoder(jwks);
  }

  @Bean
  JwtDecoder jwtDecoder() throws JOSEException {
     return NimbusJwtDecoder.withPublicKey(rsaKey.toRSAPublicKey()).build();
  }
```

KeyGenerator.java

```
package org.studyeasy.SpringRestdemo.security;

import org.springframework.stereotype.Component;

import java.security.KeyPair;
import java.security.KeyPairGenerator;

@Component
final class KeyGeneratorUtils {

    private KeyGeneratorUtils() {}

    static KeyPair generateRsaKey() {
        KeyPair keyPair;
        try {
            KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(2048);
            keyPair = keyPairGenerator.generateKeyPair();
        } catch (Exception ex) {
            throw new IllegalStateException(ex);
        }
        return keyPair;
    }

}
```

This Key generator is inbuilt key generator.

Creating an instance of RSAKey

```
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
```

We have to extract the public and private key pair -> So we make use of Jwks.java

JWK -> Json/Jason Web Key -> This code will extract public and private key and it will return it.

```
package org.studyeasy.SpringRestdemo.security;

import com.nimbusds.jose.jwk.RSAKey;

import java.security.KeyPair;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.util.UUID;

public class Jwks {

    private Jwks() {}

    public static RSAKey generateRsa() {
        KeyPair keyPair = KeyGeneratorUtils.generateRsaKey();
        RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
        RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
        return new RSAKey.Builder(publicKey)
                .privateKey(privateKey)
                .keyID(UUID.randomUUID().toString())
                .build();
    }
}
```

In order to define, we need to define the source for using the keys. It will be input to the encoder and decoder.

```
 @Bean
 public JWKSource<SecurityContext> jwkSource() {
    rsaKey = Jwks.generateRsa();
    JWKSet jwkSet = new JWKSet(rsaKey);
    return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);
}
```

In Order to decode, we make use of public key. In Order to encode, we make use of private key.

It will make our application really secure.

```
 @Bean
    JwtEncoder jwtEncoder(JWKSource<SecurityContext> jwks) {
        return new NimbusJwtEncoder(jwks);
    }

    @Bean
    JwtDecoder jwtDecoder() throws JOSEException {
        return NimbusJwtDecoder.withPublicKey(rsaKey.toRSAPublicKey()).build();
    }
```

Below is the Security Chain

```
@Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
                .authorizeHttpRequests(authorize -> authorize
                        .requestMatchers("/token").permitAll()
                        .requestMatchers("/").permitAll()
                        .requestMatchers("/swagger-ui/**").permitAll()
                        .requestMatchers("/v3/api-docs/**").permitAll()
                        .requestMatchers("/test").authenticated())
                .oauth2ResourceServer(oauth2 -> oauth2
                        .jwt(Customizer.withDefaults()))
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(csrf -> csrf.disable())
                .headers(headers -> headers
                        .frameOptions(frameOptions -> frameOptions.disable()));
        return http.build();
    }
```