OAuth 2 Client Steps - Google

OAuth2 Client Authentication with Google in Spring Boot

1. Add OAuth2 Client Dependency

Add the following dependency in pom.xml:

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

2. Google Console Setup

1. **Create a new project** in [Google Cloud Console](). Name it **Spring Boot Security Project**.
2. **Go to Dashboard** → Navigate to **API & Services** → Select **Credentials**.
3. Click **Create Credentials** → Select **OAuth2 Client ID**.
4. Configure **OAuth2 Consent Screen**:
    - Fill in the required details and save.
    - Click **Add or Remove Scopes**, select the first three, and update.
    - Add test users and proceed.
    - Publish the app by clicking **Publish App**.
5. **Obtain Credentials**:
    - Navigate to **Credentials** → **Create OAuth2 Client ID**.
    - Choose **Web Application**, name it **Spring Security App**.
    - Configure **Authorized JavaScript Origins**:
        - http://localhost:8080
        - http://localhost
    - Configure **Authorized Redirect URIs**:
        - http://localhost:8080/login/oauth2/code/google
    - Click **Create** → Copy **Client ID** and **Client Secret**.

3. Configure application.yml or application.properties also fine

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id:    your-client-id
            client-secret:    your-client-secret
```

4. Web Security Configuration

Create a configuration class in the configuration package:

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
            .authorizeHttpRequests(auth ->
                    auth.requestMatchers("/post/all", "/auth/**").permitAll()
                        .anyRequest().authenticated())
            .csrf(csrfConfig -> csrfConfig.disable())
            .sessionManagement(sessionConfig ->
                    sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
            .addFilterBefore(loggingFilter, JwtAuthFilter.class)
            .oauth2Login(oauth2config ->
                    oauth2config.failureUrl("/login?error=true"));
    return httpSecurity.build();
}
```

5. Running the Application

Start the project and navigate to http://localhost:8080/login. Click **Google Login** to authenticate and receive user information.

6. Create OAuth2 Success Handler

Inside a new package handler, create OAuth2SuccessHandler:

```
@Component
@Slf4j
public class OAuth2SuccessHandler extends SimpleUrlAuthenticationSuccessHandler {
    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws IOException,
ServletException {
        OAuth2AuthenticationToken token = (OAuth2AuthenticationToken) authentication;
        DefaultOAuth2User oAuth2User = (DefaultOAuth2User) token.getPrincipal();
        log.info("User Email: " + oAuth2User.getAttribute("email"));
    }
}
```

7. Update Web Security Configuration

Modify securityFilterChain to include the success handler:

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
```

```
        .authorizeHttpRequests(auth ->
                auth.requestMatchers("/post/all", "/auth/**").permitAll()
                    .anyRequest().authenticated())
        .csrf(csrfConfig -> csrfConfig.disable())
        .sessionManagement(sessionConfig ->
                sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
        .addFilterBefore(loggingFilter, JwtAuthFilter.class)
        .oauth2Login(oauth2config ->
                oauth2config.failureUrl("/login?error=true")
                    .successHandler(oAuth2SuccessHandler));
    return httpSecurity.build();
}
```

Run the application again. On login, the email should be logged in the console:

```
INFO  [OAuth2SuccessHandler]: User Email: example@gmail.com
```

8. Check and Save User in Database

Modify OAuth2SuccessHandler:

```
@Value("${deploy.env}")
private String deployenv;

@Override
public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse
response, Authentication authentication) throws IOException, ServletException {
    OAuth2AuthenticationToken token = (OAuth2AuthenticationToken) authentication;
    DefaultOAuth2User oAuth2User = (DefaultOAuth2User) token.getPrincipal();
    String email = oAuth2User.getAttribute("email");

    UserEntity user = userService.findByEmail(email);
    if (user == null) {
        UserEntity userEntity = UserEntity.builder()
                .name(oAuth2User.getName())
                .email(email)
                .build();
        user = userService.save(userEntity);
    }

    String accessToken = jwtServices.generateAccessToken(user);
    String refreshToken = jwtServices.generateRefreshToken(user);
    Cookie cookie = new Cookie("refreshToken", refreshToken);
    cookie.setHttpOnly(true);
    cookie.setSecure("production".equals(deployenv));
    response.addCookie(cookie);

    String frontEndUrl = "http://localhost:8080/home.html?token=" + accessToken;
    response.sendRedirect(frontEndUrl);
}
```

9. Create a Frontend Page

Create home.html to display user details after successful login. Run the application, login with Google, and see authentication details redirected to home.html.

Conclusion

By following these steps, we have successfully implemented OAuth2 authentication with Google in Spring Boot, securely handling user login, storing user data, and generating JWT tokens for authentication.

By Aron