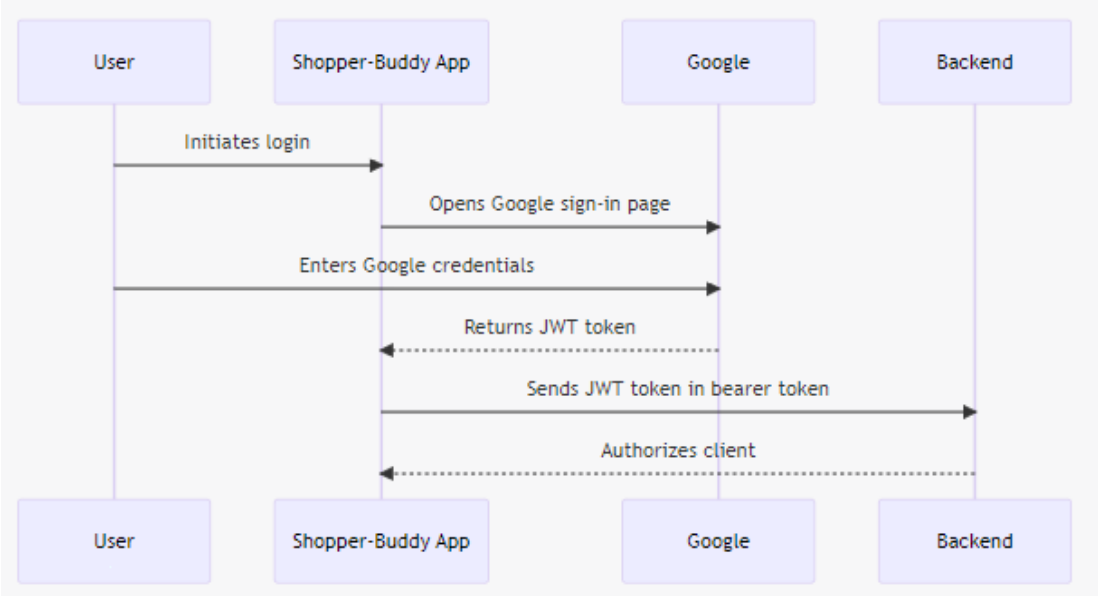
Google OAuth2 Android/Spring Boot Integration

OAuth2 Authorization Code flow

Basic User is not authenticated scenario:

1. User opens app on Android
2. User is shown Google sign in Activity
3. User signs in with Google
4. User is redirected back to the Main App
5. Main App gets JWT token from the signin result
6. App uses JWT token to make requests to the backend server (Spring boot)
7. Spring Authorizes requests based on presence of JWT token in Authorization header.
    a. Requests that do not present a JWT token are redirected to google for authentication by the backend server.

We integrate authentication in an Android app.



Prerequisites:

- Basic computer literacy
- Basic understanding of Java ecosystem
    - Maven
    - Spring boot
- Have a Spring boot project set up

- Basic understanding of Android ecosystem
    - Gradle
- Have a simple android system setup

This guide applies to :

spring.boot version: 3.0.4
springboot.security version: 6.0.3
google-api-client version: 2.2.0
google-auth-library-oauth2-http version: 1.16.0
android sdk 33
Play-services-auth:20.5.0

Links to the project:
Frontend: https://github.com/SirMeows/shopper-buddy-app
Backend: https://github.com/SirMeows/shopper-buddy

Here are the steps for implementing Google OAuth2 in this Spring Boot project:

Setup Steps

Google setup

1. **Create Google OAuth Credentials**

    - Go to Google Cloud Console
    - Create a new project
    - Enable the Google+ API
    - Configure the OAuth consent screen
    - Create OAuth client ID credentials
    - Set authorized redirect URIs (likely something like `http://localhost:8080/login/oauth2/code/google`)
    - Save the client ID and client secret

Spring boot setup:

2. **Setup environment variables**
    - Assign the saved client ID from step 1 to GOOGLE_CLIENT_ID
    - Assign the saved client secret from step 1 to GOOGLE_CLIENT_SECRET

3. **Configure Application Properties**

    - Add OAuth2 client registration properties in application.properties:

```
spring.security.oauth2.client.registration.google.client-id=GOOGLE_CLIENT_ID
spring.security.oauth2.client.registration.google.client-secret=GOOGLE_CLIENT_SECRET
spring.security.oauth2.client.registration.google.scope=email,profile
```

3. **Add maven dependencies**
    - See appendix 1.
4. **Configure Spring Security**
    - See appendix 2.
    - Creates `SecurityConfig`
5. **Create a Custom Class to load the User**
    - See appendix 3.
    - Creates `CustomOAuth2UserService`

Android Setup:

6. **Create client for App**
    - Use the Google cloud console to create a client id specific the the android app
    - Remember this id
7. **Add Gradle dependencies**
    - `'com.google.android.gms:play-services-auth:20.5.0'`
8. **Integrate google authentication into MainActivity**
    - See appendix 4
9. **Configure clientId**
    - Add a string resource with key serverClientId
    - Assign value of the clientId we remembered in step 6.
    - Creates line in res/values/strings.xml
        i. `<string name="serverClientId">24257 … </string>`
10. **Get JWT Token**
    - You can get the JwtToken after user signed in with `GoogleSignIn.getLastSignedInAccount(this).getIdToken()`
11. **Authorize HTTP requests**
    - Add authorization header with jwtToken when making request to the backend server
      `.header("Authorization", "Bearer " + idToken)`

Appendix

Appendix 1 : Maven dependencies

```
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-core</artifactId>
   <version>${springboot.security.version}</version>
</dependency>
```

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
   <version>${springboot.version}</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-config</artifactId>
   <version>${springboot.security.version}</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-web</artifactId>
   <version>${springboot.security.version}</version>
</dependency>

<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-oauth2-client</artifactId>
   <version>${springboot.security.version}</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-oauth2-jose</artifactId>
   <version>${springboot.security.version}</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-oauth2-resource-server</artifactId>
   <version>${springboot.security.version}</version>
</dependency>

<dependency>
   <groupId>com.google.auth</groupId>
   <artifactId>google-auth-library-oauth2-http</artifactId>
   <version>1.16.0</version>
</dependency>
<dependency>
   <groupId>com.google.api-client</groupId>
   <artifactId>google-api-client</artifactId>
   <version>2.2.0</version>
</dependency>
```

Appendix 2: spring boot security config

```
@Configuration(
        proxyBeanMethods = false
)
public class SecurityConfig {
    @Order(1)
    @Bean
    SecurityFilterChain jwtSecurityFilterChain(HttpSecurity http)
throws Exception {
        http.securityMatcher(new BearerTokenRequestMatcher()) // only
requests that present a bearer token
            .authorizeHttpRequests((requests) -> {
                ((AuthorizeHttpRequestsConfigurer.AuthorizedUrl)
requests.anyRequest()).authenticated();
            });

        http.oauth2ResourceServer(OAuth2ResourceServerConfigurer::jwt);
        return (SecurityFilterChain) http.build();
    }

    @Order(2)
    @Bean
    public SecurityFilterChain
oauthLoginSecurityFilterChain(HttpSecurity http) throws Exception {
        http.authorizeRequests(authorizeRequests ->
                        authorizeRequests.anyRequest().authenticated()
                )
                .oauth2Login();
        return http.build();
    }
}

class BearerTokenRequestMatcher implements RequestMatcher {
    @Override
    public boolean matches(HttpServletRequest request) {
        String auth = request.getHeader("Authorization");
        return (auth != null && auth.startsWith("Bearer"));
    }
}
```

Appendix 3: Custom Spring OAuth2UserService

```
@Service
```

```
public class CustomOAuth2UserService extends DefaultOAuth2UserService
{

   @Override
   public OAuth2User loadUser(OAuth2UserRequest userRequest) throws
OAuth2AuthenticationException {
       var oAuth2User =  super.loadUser(userRequest);
        // add code to register the user in the local database if
desired.


       return googleOAuth2User;
   }
}
```

Appendix 4:

```
private GoogleSignInClient googleSignInClient;

ActivityResultLauncher<Intent> signInResultLauncher;

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   googleSignInClient = getSignInClient();
   if (isUserLoggedIn()) {
       // continue show application
   } else {
       startSignInActivity();
   }
}

private void startSignInActivity() {
   signInBinding = ActivitySignInBinding.inflate(getLayoutInflater());
   setContentView(signInBinding.getRoot());
   signInResultLauncher = registerForActivityResult(
           new ActivityResultContracts.StartActivityForResult(),
           result -> {
               if (result.getResultCode() == Activity.RESULT_OK) {
                   Intent data = result.getData();
                   Task<GoogleSignInAccount> task =
GoogleSignIn.getSignedInAccountFromIntent(data);
                   handleSignInResult(task);
```

```
                }
            }
    );

    signInBinding.signInButton.setOnClickListener(v -> signIn());
}

private GoogleSignInClient getSignInClient() {
    GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestEmail()
            .requestId()
            .requestProfile()
            .requestIdToken(getString(R.string.serverClientId))
            .build();

    return GoogleSignIn.getClient(this, gso);
}

private boolean isUserLoggedIn() {
    GoogleSignInAccount account =
GoogleSignIn.getLastSignedInAccount(this);
    return account != null && !account.isExpired() &&
account.getIdToken() != null;
}

private void signIn() {
    Intent signInIntent = googleSignInClient.getSignInIntent();
    signInResultLauncher.launch(signInIntent);
}

private void handleSignInResult(Task<GoogleSignInAccount>
completedTask) {
    try {
        GoogleSignInAccount account =
completedTask.getResult(ApiException.class);
        // continue and show main application
    } catch (ApiException e) {
        Log.e(TAG, "signInResult:failed code=" + e.getStatusCode());
    }
}
```