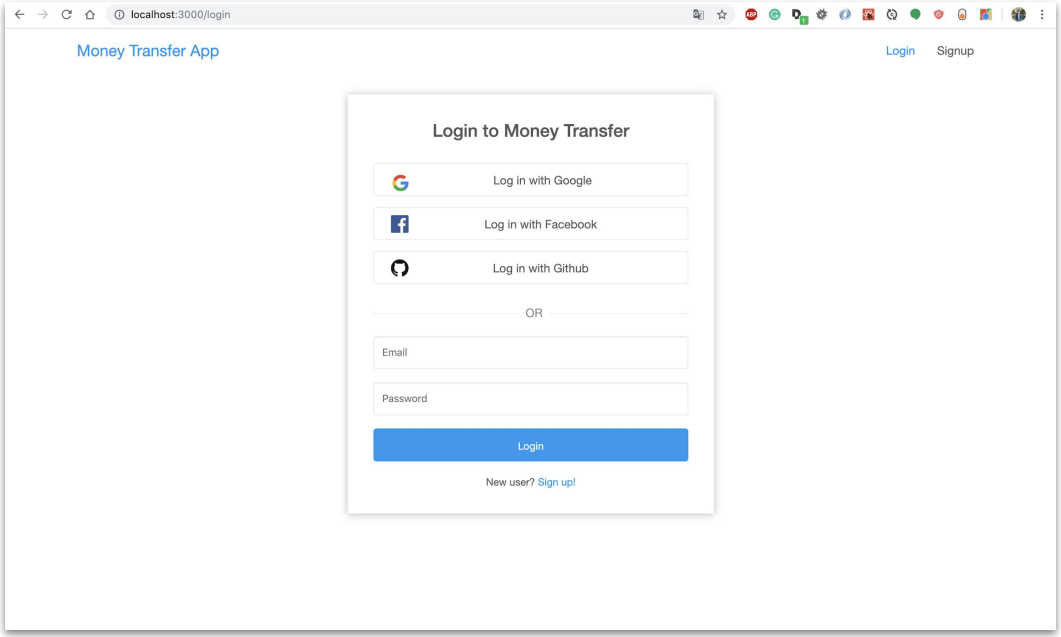
OAuth2 Social Login Facebook

Demo First



http://localhost:3000/

Maven Dependency

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

Configuration

```yaml
security:
 oauth2:
  client:
   registration:
    facebook:
     clientId: 2395114657235495
     clientSecret: d1d915ed8d18972081cb192be8682bce
     redirectUriTemplate: "{baseUrl}/oauth2/callback/{registrationId}"
     scope:
      - email
      - public_profile
   provider:
    facebook:
     authorizationUri: https://www.facebook.com/v5.0/dialog/oauth
     tokenUri: https://graph.facebook.com/v5.0/oauth/access_token
     userInfoUri:
https://graph.facebook.com/v5.0/me?fields=id,first_name,middle_name,last_name,name,email,verified,is_verified,picture.width(250).height(250)
```

Configuration 2

```yaml
app:
 auth:
  tokenSecret: 926D96C90030DD58429D2751AC1BDBBC
  tokenExpirationMsec: 864000000
 oauth2:
  authorizedRedirectUris:
   - http://localhost:3000/oauth2/redirect
```

CORS Configuration

```
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {

  private final long MAX_AGE_SECS = 3600;

  @Override
  public void addCorsMappings(CorsRegistry registry) {
     registry.addMapping("/**")
     .allowedOrigins("*")
     .allowedMethods("GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS")
     .allowedHeaders("*")
     .allowCredentials(true)
     .maxAge(MAX_AGE_SECS);
  }
}
```

User Entity Changes

```
@Entity
@EntityListeners(AuditingEntityListener.class)
@Table(name = "users")
public class User {


  private String imageUrl;

  @NotNull
  @Enumerated(EnumType.STRING)
  private AuthProvider provider;

  private String providerId;
}
```

Entity AuthProvider

```
package com.nursultanturdaliev.moneytransferapp.model;

public enum AuthProvider {
  local,
  facebook,
}
```

Security Config

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  Http
        ...
        .oauth2Login()
           .authorizationEndpoint()
              .baseUri("/oauth2/authorize")
              .authorizationRequestRepository(cookieAuthorizationRequestRepository())
              .and()
           .redirectionEndpoint()
              .baseUri("/oauth2/callback/*")
              .and()
           .userInfoEndpoint()
              .userService(customOAuth2UserService)
              .and()
           .successHandler(oAuth2AuthenticationSuccessHandler)
           .failureHandler(oAuth2AuthenticationFailureHandler);

  http.addFilterBefore(tokenAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);
}
```

OAuth2 Login Flow

- The OAuth2 login flow will be initiated by the frontend client by sending the user to the endpoint `http://localhost:8080/oauth2/authorize/{provider}?redirect_uri=<redirect_uri_after_login>`.
  The `provider` path parameter is one of `google`, `facebook`, or `github`. The `redirect_uri` is the URI to which the user will be redirected once the authentication with the OAuth2 provider is successful. This is different from the OAuth2 redirectUri.

OAuth2 Login Flow 2

- On receiving the authorization request, Spring Security's OAuth2 client will redirect the user to the AuthorizationUrl of the supplied `provider`.
- All the state related to the authorization request is saved using the `authorizationRequestRepository` specified in the SecurityConfig.
- The user now allows/denies permission to your app on the provider's page. If the user allows permission to the app, the provider will redirect the user to the callback url `http://localhost:8080/oauth2/callback/{provider}` with an authorization code. If the user denies the permission, he will be redirected to the same callbackUrl but with an `error`.

OAuth2 Login Flow 3

- If the OAuth2 callback results in an error, Spring security will invoke the `oAuth2AuthenticationFailureHandler` specified in the above `SecurityConfig`.
- If the OAuth2 callback is successful and it contains the authorization code, Spring Security will exchange the `authorization_code` for an `access_token` and invoke the `customOAuth2UserService` specified in the above SecurityConfig.

OAuth2 Login Flow 4

- The `customOAuth2UserService` retrieves the details of the authenticated user and creates a new entry in the database or updates the existing entry with the same email.
- Finally, the `oAuth2AuthenticationSuccessHandler` is invoked. It creates a JWT authentication token for the user and sends the user to the `redirect_uri` along with the JWT token in a query string.

HttpCookieOAuth2AuthorizationRequestRepository

The OAuth2 protocol recommends using a `state` parameter to prevent CSRF attacks. During authentication, the application sends this parameter in the authorization request, and the OAuth2 provider returns this parameter unchanged in the OAuth2 callback.

The application compares the value of the `state` parameter returned from the OAuth2 provider with the value that it had sent initially. If they don’t match then it denies the authentication request.

To achieve this flow, the application needs to store the `state` parameter somewhere so that it can later compare it with the `state` returned from the OAuth2 provider.

HttpCookieOAuth2AuthorizationRequestRepository

```java
@Component
public class HttpCookieOAuth2AuthorizationRequestRepository implements AuthorizationRequestRepository<OAuth2AuthorizationRequest> {
  public static final String OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME = "oauth2_auth_request";
  public static final String REDIRECT_URI_PARAM_COOKIE_NAME = "redirect_uri";
  private static final int cookieExpireSeconds = 180;
  @Override
  public OAuth2AuthorizationRequest loadAuthorizationRequest(HttpServletRequest request) {
    return CookieUtils.getCookie(request, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME)
        .map(cookie -> CookieUtils.deserialize(cookie, OAuth2AuthorizationRequest.class))
        .orElse(null);
  }
  @Override
  public void saveAuthorizationRequest(OAuth2AuthorizationRequest authorizationRequest, HttpServletRequest request, HttpServletResponse response) {
    if (authorizationRequest == null) {
      CookieUtils.deleteCookie(request, response, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME);
      CookieUtils.deleteCookie(request, response, REDIRECT_URI_PARAM_COOKIE_NAME);
      return;
    }
    CookieUtils.addCookie(response, OAUTH2_AUTHORIZATION_REQUEST_COOKIE_NAME, CookieUtils.serialize(authorizationRequest), cookieExpireSeconds);
    String redirectUriAfterLogin = request.getParameter(REDIRECT_URI_PARAM_COOKIE_NAME);
    if (StringUtils.isNotBlank(redirectUriAfterLogin)) {
      CookieUtils.addCookie(response, REDIRECT_URI_PARAM_COOKIE_NAME, redirectUriAfterLogin, cookieExpireSeconds);
    }
  }

}
```

This class provides functionality for storing the authorization request in cookies and retrieving it

CustomOAuth2UserService

The `CustomOAuth2UserService` extends Spring Security's `DefaultOAuth2UserService` and implements its `loadUser()` method. This method is called after an access token is obtained from the OAuth2 provider.

In this method, we first fetch the user's details from the OAuth2 provider. If a user with the same email already exists in our database then we update his details, otherwise, we register a new user.

CustomOAuth2UserService

```
private OAuth2User processOAuth2User(OAuth2UserRequest oAuth2UserRequest, OAuth2User oAuth2User) {
  OAuth2UserInfo oAuth2UserInfo = OAuth2UserInfoFactory.getOAuth2UserInfo(oAuth2UserRequest.getClientRegistration().getRegistrationId(), oAuth2User.getAttributes());
  if(StringUtils.isEmpty(oAuth2UserInfo.getEmail())) {
    throw new OAuth2AuthenticationProcessingException("Email not found from OAuth2 provider");
  }

  Optional<User> userOptional = userRepository.findByEmail(oAuth2UserInfo.getEmail());
  User user;
  if(userOptional.isPresent()) {
    user = userOptional.get();
    if(!user.getProvider().equals(AuthProvider.valueOf(oAuth2UserRequest.getClientRegistration().getRegistrationId()))) {
      throw new OAuth2AuthenticationProcessingException("Looks like you're signed up with " +
            user.getProvider() + " account. Please use your " + user.getProvider() +
            " account to login.");
    }
    user = updateExistingUser(user, oAuth2UserInfo);
  } else {
    user = registerNewUser(oAuth2UserRequest, oAuth2UserInfo);
  }

  return UserPrincipal.create(user, oAuth2User.getAttributes());
}
```

OAuth2UserInfo

Every OAuth2 provider returns a different JSON response when we fetch the authenticated user's details. Spring security parses the response in the form of a generic `map` of key-value pairs.

```
public abstract class OAuth2UserInfo {
  protected Map<String, Object> attributes;

  public OAuth2UserInfo(Map<String, Object> attributes) {
    this.attributes = attributes;
  }

  public Map<String, Object> getAttributes() {
    return attributes;
  }

  public abstract String getId();
  public abstract String getName();
  public abstract String getEmail();
  public abstract String getImageUrl();
}
```

FacebookOAuth2UserInfo

```java
public class FacebookOAuth2UserInfo extends OAuth2UserInfo {
  public FacebookOAuth2UserInfo(Map<String, Object> attributes) {
    super(attributes);
  }
  @Override
  public String getId() {
    return (String) attributes.get("id");
  }
  @Override
  public String getName() {
    return (String) attributes.get("name");
  }
  @Override
  public String getEmail() {
    return (String) attributes.get("email");
  }
  @Override
  public String getImageUrl() {
    if(attributes.containsKey("picture")) {
      Map<String, Object> pictureObj = (Map<String, Object>) attributes.get("picture");
      if(pictureObj.containsKey("data")) {
        Map<String, Object>  dataObj = (Map<String, Object>) pictureObj.get("data");
        if(dataObj.containsKey("url")) {
          return (String) dataObj.get("url");
        }
      }
    }
    return null;
  }
}
```

OAuth2UserInfoFactory

```
public class OAuth2UserInfoFactory {

  public static OAuth2UserInfo getOAuth2UserInfo(String registrationId, Map<String, Object> attributes) {
    if (registrationId.equalsIgnoreCase(AuthProvider.facebook.toString())) {
      return new FacebookOAuth2UserInfo(attributes);
    } else {
      throw new OAuth2AuthenticationProcessingException("Sorry! Login with " + registrationId + " is not supported yet.");
    }
  }
}
```

OAuth2AuthenticationSuccessHandler

On successful authentication, Spring security invokes the `onAuthenticationSuccess()` method of the `OAuth2AuthenticationSuccessHandler` configured in `SecurityConfig`.

In this method, we perform some validations, create a JWT authentication token, and redirect the user to the `redirect_uri` specified by the client with the JWT token added in the query string.

OAuth2AuthenticationSuccessHandler

```java
@Component
public class OAuth2AuthenticationSuccessHandler extends SimpleUrlAuthenticationSuccessHandler {
  @Override
  public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) throws IOException, ServletException {
    String targetUrl = determineTargetUrl(request, response, authentication);
    if (response.isCommitted()) {
      logger.debug("Response has already been committed. Unable to redirect to " + targetUrl);
      return;
    }

    clearAuthenticationAttributes(request, response);
    getRedirectStrategy().sendRedirect(request, response, targetUrl);
  }

  protected String determineTargetUrl(HttpServletRequest request, HttpServletResponse response, Authentication authentication) {
    Optional<String> redirectUri = CookieUtils.getCookie(request, REDIRECT_URI_PARAM_COOKIE_NAME)
        .map(Cookie::getValue);
    if (redirectUri.isPresent() && !isAuthorizedRedirectUri(redirectUri.get())) {
      throw new BadRequestException("Sorry! We've got an Unauthorized Redirect URI and can't proceed with the authentication");
    }

    String targetUrl = redirectUri.orElse(getDefaultTargetUrl());
    String token = tokenProvider.createToken(authentication);
    return UriComponentsBuilder.fromUriString(targetUrl)
        .queryParam("token", token)
        .build().toUriString();
  }

}
```

OAuth2AuthenticationFailureHandler

In case of any error during OAuth2 authentication, Spring Security invokes the `onAuthenticationFailure()` method of the `OAuth2AuthenticationFailureHandler` that we have configured in `SecurityConfig`.

It sends the user to the frontend client with an error message added to the query string

OAuth2AuthenticationFailureHandler

```
@Component
public class OAuth2AuthenticationFailureHandler extends SimpleUrlAuthenticationFailureHandler {

  @Autowired
  HttpCookieOAuth2AuthorizationRequestRepository httpCookieOAuth2AuthorizationRequestRepository;

  @Override
  public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response, AuthenticationException exception) throws IOException, ServletException {
    String targetUrl = CookieUtils.getCookie(request, REDIRECT_URI_PARAM_COOKIE_NAME)
            .map(Cookie::getValue)
            .orElse(("/"));
    targetUrl = UriComponentsBuilder.fromUriString(targetUrl)
            .queryParam("error", exception.getLocalizedMessage())
            .build().toUriString();

    httpCookieOAuth2AuthorizationRequestRepository.removeAuthorizationRequestCookies(request, response);

    getRedirectStrategy().sendRedirect(request, response, targetUrl);
  }
}
```

Utility Class CookieUtils

```
public class CookieUtils {
  public static Optional<Cookie> getCookie(HttpServletRequest request, String name) {
    Cookie[] cookies = request.getCookies();

    if (cookies != null && cookies.length > 0) {
      for (Cookie cookie : cookies) {
        if (cookie.getName().equals(name)) {
          return Optional.of(cookie);
        }
      }
    }
    return Optional.empty();
  }
  public static void addCookie(HttpServletResponse response, String name, String value, int maxAge) {
    Cookie cookie = new Cookie(name, value);
    cookie.setPath("/");
    cookie.setHttpOnly(true);
    cookie.setMaxAge(maxAge);
    response.addCookie(cookie);
  }

}
```

Utility Class CookieUtils

```
public class CookieUtils {

  public static void deleteCookie(HttpServletRequest request, HttpServletResponse response, String name) {
    Cookie[] cookies = request.getCookies();
    if (cookies != null && cookies.length > 0) {
      for (Cookie cookie : cookies) {
        if (cookie.getName().equals(name)) {
          cookie.setValue("");
          cookie.setPath("/");
          cookie.setMaxAge(0);
          response.addCookie(cookie);
        }
      }
    }
  }

}
```