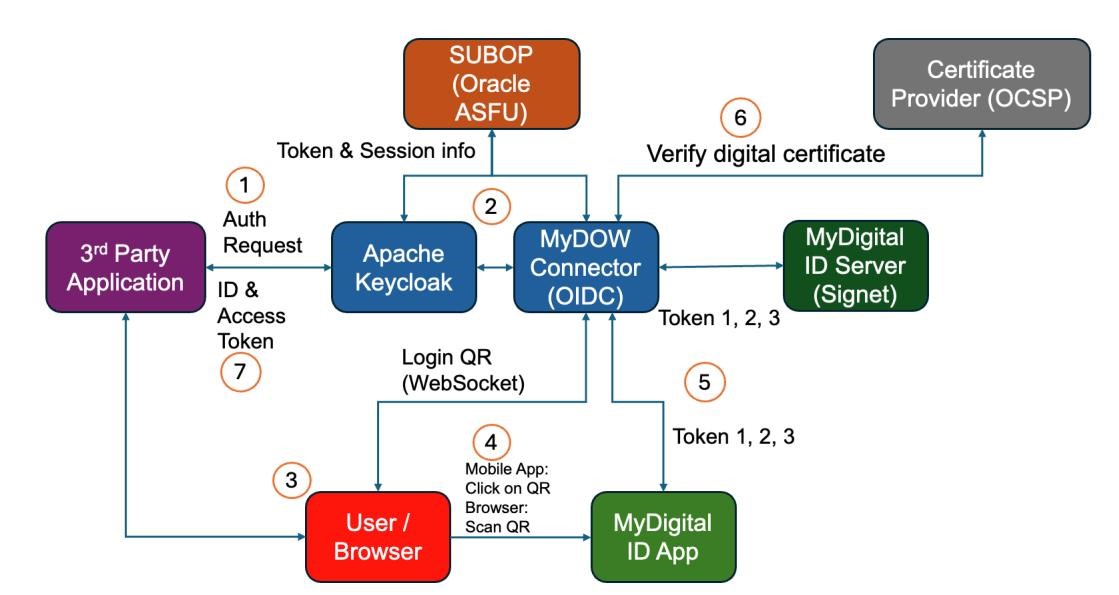
# 1.0 System Overview

## 1.1 SSO Integration

SSO Integration uses Keycloak, an open-source Identity and Access Management (IAM) for modern apps like single-page applications, mobile apps, and REST APIs. The services provided to facilitate the integration of SSO include:

* Authentication: Centralized login service to verify user identities
* Authorization: Role-based and implements detailed access control for applications
* Token Management: Issuance and validation of tokens using OpenID Connect protocol.
* User Federation: Integration with external user directories like LDAP, Active Directory and more.

## 1.2 System Architecture



**Figure 1: System Architecture**

### 1.2.1 System Architecture Flow

1. The user will go to the Login Page in the 3rd party application. The 3rd party application will be redirected to the Keycloak.
2. Keycloak then will be redirected to MyDOW Connector (OIDC).
3. MyDOW Connector will generate a unique QR Code and will display it to the user.
4. The user clicks the QR Code (mobile) or scans the QR Code (browser) and authenticates using the MyDigital ID application.
5. MyDigital ID application connects to MyDOW Connector and performs 3-way handshaking.
6. On the 3rd handshake, MyDOW Connector will receive the user certificate and check the certificate with OCSP.
7. MyDOW connector will return the auth token to Keycloak and Keycloak will return the token to the 3rd party application.

## 1.3 MyDigital ID SSO Platform

MyDigital ID SSO platform consists of the Identity and Access Management (IAM) platform on the frontend that supports OAuth 2.0 and OpenID Connect. Keycloak provides the functionality to manage authentication client secret, authentication flow behavior and other operational elements that is required by OAuth 2.0 and OIDC to function.

MyDOW OIDC Connector plugin is a purpose-built plugin that was developed to interface OIDC with the mechanism that performs 3-way handshaking of MyDigital ID (refer to MyDigital ID Integration Guideline Document). In essence, MyDOW OIDC connector is presented as an Identity Provider to Keycloak.

## 1.4 Keycloak SSO Protocol

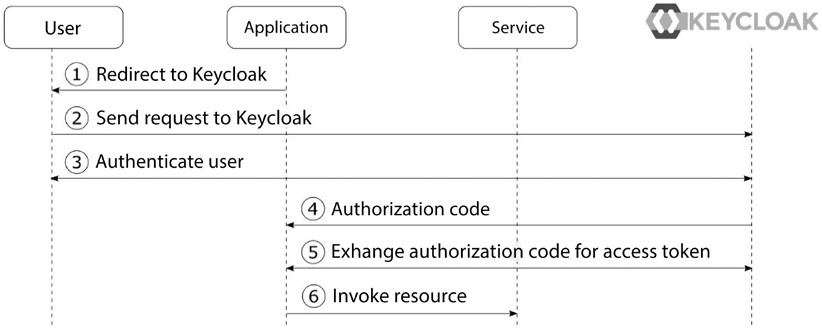
Keycloak uses industry-standard protocols like OAuth 2.0 and OpenID Connect secure authentication and authorization. Using industry-standard protocols is crucial not only for ensuring robust security but also for simplifying integration with both existing and new applications.

### 1.4.1 OAuth 2.0

With OAuth 2.0, sharing user data to third-party applications is easy, does not require sharing user credentials, and allows control over what data is shared. Four (4) roles defined in OAuth 2.0:

* **Resource owner**: The end user that owns resources that an application wants to access.
* **Resource server**: The service hosting the protected resources.
* **Client**: The application that would like to access the resource.
* **Authorization server**: The server issuing access to the client, which is the role of Keycloak.

In an OAuth 2.0 protocol flow, the client requests access to a resource on behalf of a resource owner from the authorization server. The authorization server issues limited access to the resource at the resource server by including access token in the request.



###### Figure 2: Simplified OAuth 2.0 Authorization Code Grant Type

The steps in the diagram are as follows:

1. The application sends an authentication request to the user's browser to be redirected to Keycloak for authorization.
2. The browser redirects the user to Keycloak’s authorization page.
3. If the user is not authenticated with Keycloak, Keycloak authenticates the user
4. The application gets an authorization code from Keycloak.
5. The application then exchanges the code for an access token from Keycloak.
6. The application uses the access token to access the protected resource.

Access tokens are passed around from the application to services, usually having a short lifetime. To get new access tokens without repeating the whole process, a refresh token is used.

### 1.4.2 OpenID Connect

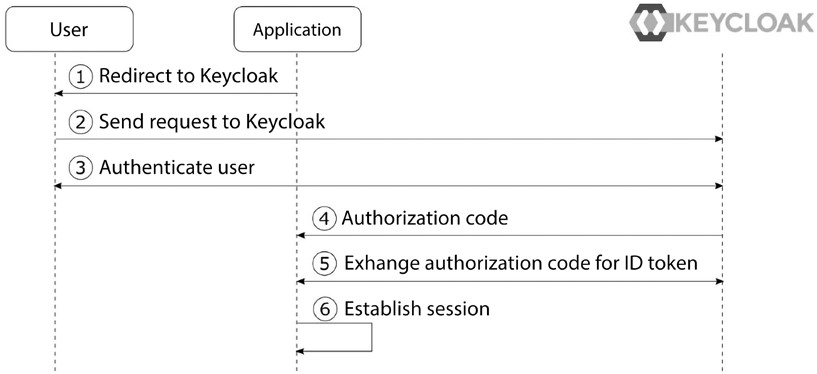
Since OAuth 2.0 is for authorization, but not authentication, OpenID Connect adds an authentication layer on top of OAuth 2.0. OpenID Connect has enabled a whole ecosystem of websites to no longer need to deal with user management and authenticating users. OpenID Connect is useful to have a centralized solution for authentication supporting single sign-on and increases security as applications do not have access to the user credentials directly. Furthermore, it enables the use of stronger authentication, such as OTP or WebAuthn, without the need to support it directly within applications.

OpenID Connect (OIDC) involves three (3) main roles in its protocol:

1. **End user**: It is the same as the resource owner in OAuth 2.0. This is simply the human being who is logging in or being authenticated.
2. **Relying Party (RP):** The application or website that needs to verify or authenticate the end user.
3. **OpenID Provider (OP):** The identity provider that is authenticating the user, which is the role of Keycloak.

In OpenID Connect, the RP asks the OP for the user's identity and can also obtain a token since it builds on OAuth 2.0. OpenID Connect uses OAuth 2.0's Authorization Code grant but adds *scope=openid* to make it an **authentication request** instead of just an **authorization request**. There are two (2) flows in OpenID Connect:

1. Authorization code flow: This follows the OAuth 2.0 Authorization Code flow, returning an authorization code that can be exchanged for an ID token, access token, and refresh token.
2. Hybrid flow: The ID token and authorization code are returned together in the initial request.

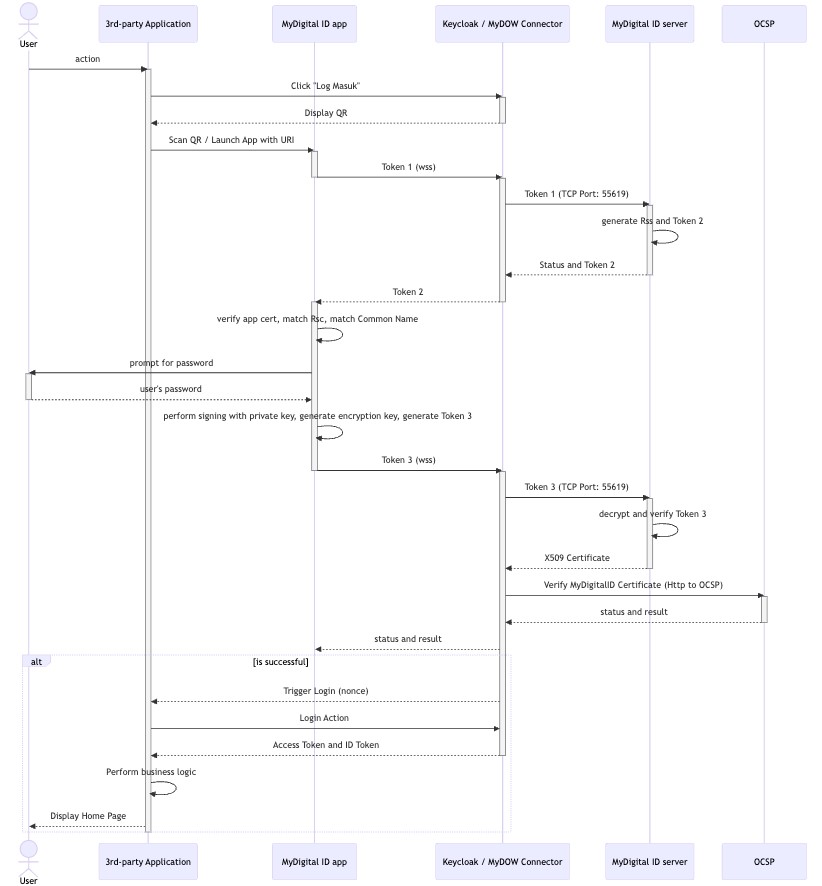


###### Figure 3: Simplified OpenID Connect Authorization Code Flow

The steps in the diagram are as follows:

1. The app prepares a request to authenticate and asks the user’s browser to be redirected to Keycloak.
2. The browser redirects the user to Keycloak’s login page (authorization endpoint).
3. If the user is not logged in yet, Keycloak will ask them to log in.
4. The app gets an authorization code from Keycloak as a response.
5. The app then exchanges this authorization code with Keycloak for an ID token and an access token.
6. The app uses the ID token to identify the user and start a logged-in session.

## 1.5 MyDigital ID SSO Protocol Sequence



**Figure 4: MyDigital ID SSO Protocol Sequence Diagram**

### 1.5.1 MyDigital ID SSO Protocol Sequence Components

**Components:**

1. **User**: The end-user requesting access to resources.
2. **3rd-Party Application (App)**: The client application consists of a mobile application or web application that the user interacts with.
3. **MyDigitalID App**: An app responsible for verifying users and generating tokens.
4. **Keycloak/MyDOW Cluster (OIDC Provider)**: The OpenID Connect (OIDC) provider for handling authentication.
5. **WebSocket Cluster (WSS)**: Manages WebSocket connections for real-time communication.
6. **Signet Cluster**: Handles secure token verification and status updates.
7. **Flow Explanation:**

###### 1. User Request

• The user makes a request to the application for a resource, but there is no prior authentication.

###### 2. Redirect to Keycloak

* The application redirects the user to Keycloak for authentication.
* Keycloak detects the Mydigital ID Identity Provider as the IDP stated in the authentication flow of the client.
* Keycloak redirects requests to MyDigital MyDOW IDP which then produces the unique nonce and generates the MyDigital ID QR code to be scanned and application url link to Signet server if the QR is clicked.

###### 3. Action and Token Generation

* The user performs an action by scanning using MyDigital ID app

(Desktop web app) or click on the QR code (Mobile app or Mobile PWA)

* MyDigitalID commence the MyDigital ID authentication protocol (3way handshaking)

###### 4. WebSocket Initiation

* While the MyDigital ID app is bound to the websocket server, the 3rd party application is also connected with the Websocket Connections. The connections from the 3rd party application will join a channel determined by the nonce.
* The Two apps, 3rd party login and my Digital ID websocket login are bound by the nonce websocket channel.

###### 5. Final Token Generation and Authentication

* Once the 3-way handshaking process is accomplished (includes the password verification), a JWT Token is responded to Keycloak.
* The JWT Token responded by MyDOW OIDC to Keycloak will consist of full name and IC Number.
* Once Keycloak receives the JWT Token, Keycloak will take the claims (fullname and IC Number) and incorporate the info into a newly generated JWT token to be returned to the initiator client.

###### 6. Resource Access

* The user accesses the application with the JWT token.
* The application validates the token with Keycloak.
* Upon successful validation, Keycloak informs the application, and the application provides access to the requested resources.

## 1.6 MyDigital ID SSO Keycloak Client

A Keycloak client is an application or service that interacts with Keycloak for authentication and authorization. In Keycloak, a "client" can be any system that integrates with Keycloak to use its identity management capabilities, such as single sign-on (SSO), user authentication, and role-based access control (RBAC).

Keycloak clients are typically web applications, mobile applications, or even backend services that use Keycloak to authenticate users and obtain access tokens, or they could be microservices that need to secure their APIs.

### 1.6.1 MyDigital ID SSO Keycloak Client Configuration

When configuring a Keycloak client, you usually specify the following settings:

1. **Client ID**: A unique identifier for the client within Keycloak. This is how Keycloak knows which client is making a request.
2. **Client Secret**: A secret key used for secure communication between the client and Keycloak (in the case of confidential clients).
3. **Redirect URI**: The URL to which Keycloak will redirect the user after successful authentication. This is typically the URL of the application that the user is trying to access.
4. **Protocol**: The authentication protocol used by the client. Common protocols include:
   * **OpenID Connect** (OIDC) – A modern protocol based on OAuth 2.0, commonly used for web and mobile applications.
   * **OAuth 2.0** – A framework for access delegation, widely used for authorization.
5. **Access Type**:
   * **Confidential**: Clients that can securely store credentials (e.g., server-side applications). These clients authenticate themselves to Keycloak using client secrets.
   * **Public**: Clients that cannot securely store credentials (e.g., single-page applications, mobile apps).
   * **Bearer-only**: Clients that only accept access tokens and cannot initiate authentication themselves. Common for APIs or resource servers.
6. **Roles and Scopes**: The permissions or access levels that the client can request, often used to limit or specify access to certain resources or actions.

### 1.6.2 Client Flow in MyDigital ID SSO Keycloak

1. A user accesses a client application and is redirected to Keycloak for authentication.
2. Keycloak authenticates the user (e.g., through login credentials, social login, etc.).
3. After successful authentication, Keycloak sends an authorization code or access token back to the client application (depending on the flow used).
4. The client can use the token to make authorized API calls, or the token can be stored for the duration of the session.

### 1.6.3 Example of Client Flow in MyDigital ID SSO Keycloak Use Cases

* **Web Application**: A client in Keycloak could represent a React or Angular app that interacts with a backend API to fetch user data after authentication.
* **Mobile App**: A client could represent a mobile app that needs to authenticate users via Keycloak and interact with a REST API.
* **Service-to-Service Communication**: A client could also be a backend service that needs to authenticate itself when calling other services or APIs.

In summary, a MyDigital ID SSO Keycloak client is a representation of an application or service that uses Keycloak for handling authentication and authorization, with various settings and protocols to define its interaction with Keycloak's identity management features.

# SECTION 2: WEB LOGIN, MOBILE APP SSO INTEGRATION AND DEVELOPER GUIDE

# 2.0 3rd Party Application Integration with MyDigital ID Keycloak SSO Integration

Integration with MyDigital ID Keycloak SSO is usually done by incorporating standard open source libraries that are readily available on various platforms. These libraries are usually maintained by the open source community, most often up to date with features and security patches. This method is recommended because it will reduce development time and be more stable due to the fact that they are being used widely by the community. The following section will outline the necessary task required to accomplish such integration with examples towards the end.

## 2.1 MyDigital ID SSO Integration Checklist

Below is a list of information(s) that is needed for us to assess the compatibility and support feature of the current 3rd party application with the MyDigital ID SSO platform. Assessment is primarily needed to assess if the current web or application framework has a standard Keycloak library that will reduce development time or custom development for the authentication protocols aforementioned which will extend development time.

**Table 1: Application Survey Form**

|  |  |  |
| --- | --- | --- |
| **#** | **Item** | **Description** |
| 1 | Application Survey Form | A third-party application product owner MUST provide us the details of the software framework used and send Application form email to sso@myid.my to request the integration.    The Application Survey Form should include the following details:   * Apps: * Language: * Language Version: * Framework: * Framework Version: * Os: * Os Version: * DB: * DB Version: * Authentication Type: * Public App/Internal App: Warganegara /Bukan Warganegara / Staff/Orang Awam     **Please refer Appendix A** |

## 2.2 Library for Keycloak SSO Integration

Below is a list of libraries that supports Keycloak SSO Integration with the suitable Programming Language and framework. The website or application environment must meet the required language and framework versions before installing the library.

**Table 2: List of Libraries for SSO Integration**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Language** | **Language Version** | **Framework** | **Framework Version** | **Library** | **Library Version** |
| Node.js | 12.x, 14.x | Express.js | 4.x | keycloak-connect | 11.x |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Language** | **Language Version** | **Framework** | **Framework Version** | **Library** | **Library Version** |
|  |  |  |  |  |  |
| 16.x+ | Express.js | 4.x | keycloak-connect | 12.x |
| 14.x+ | NestJS | 7.x to 9.x | nest-keycloak-connect | 2.x |
| 16.x+ | NestJS | 10.x | nest-keycloak-connect | 3.x |
| Java | 8+ | Spring Boot | 2.x | spring-boot-starteroauth2-client | 2.x |
| 11+ | Spring Boot | 2.x | spring-security-oauth2 | 5.x |
| 8+ | Java EE  (Jakarta  EE) | 8.x | keycloak-java-adapter | 1.x |
| 4+ | Quarkus | 1.x | keycloak-authorization | 1.x |
| Python | 3.6+ | Flask | 1.x to 2.x | flask-oauthlib | 0.9 |
| 3.6+ | Django | 3.x | django-oauth-toolkit, | 1.x |
| 3.6+ | FastAPI | 0.68+ | fastapi-keycloak | 1.x |
| PHP | 8.1+ | Laravel | 10.x | laravel-keycloakguard | 3.x |
| 8.0+ | Laravel | 9.x | laravel-keycloakguard | 2.x |
| 7.4+ | Laravel | 7.x to 8.x | laravel-keycloakguard | 2.x |
| 7.0+ | Codeigniter | 3.x | oauth2-client | 1.x |
| 7.4+ | Codeigniter | 3.x | oauth2-client | 2.x |
| 8.0+ | Codeigniter | 4.x | openid-connect-php | 3.x |
| 7.4+ | Symfony | 4.x | knpuniversity/oauth2client | 2.x |
| React | 17.x+ | Next.js | 10.x+ | next-auth (Keycloak provider) | 3.x+ |
|  | 16.x+ | Gatsby | 3.x+ | gatsby-pluginkeycloak | 1.x |
| .NET  MAUI | 6.x+ | Xamarin | 5.x+ | Xamarin.Auth | 1.x+ |
| Ionic | 6.x+ | Angular | 11.x+ | angular-oauth2-oidc | 11.x+ |
| Swift | 5.x+ | SwiftUI | 2.x+ | OAuthSwift | 2.x+ |
| **Language** | **Language Version** | **Framework** | **Framework Version** | **Library** | **Library Version** |
| Kotlin | 1.4+ | Spring Boot | 2.x+ | spring-securityoauth2-client | 5.x+ |
| 1.4+ | Ktor | 2.x+ | ktor-client-auth | 2.x+ |
| Dart | 2.x+ | Flutter | 2.x+ | flutter\_appauth | 0.9.x+ |
| 2.x+ | AngularDart | 6.x+ | angular\_oauth2 | 6.x+ |
| 2.x+ | Aqueduct | 4.x+ | aqueduct\_auth | 4.x+ |

## 2.3 Keycloak Configuration

The configuration of Keycloak for an application or website is managed through the Keycloak Admin Console, which provides a user-friendly interface for setting up and maintaining Keycloak's integration. This console allows administrators to configure various components necessary for the secure authentication and authorization of users. Below is a comprehensive list of configurations required to integrate Keycloak into the application or website, ensuring that user access is properly controlled and authenticated in accordance with the desired security policies.

**Table 3: Keycloak Configuration Example**

|  |  |
| --- | --- |
| **Field** | **Description** |
| KEYCLOAK\_CLIENT\_ID | <Client\_ID> |
| KEYCLOACK\_CLIENT\_SECRET | NUxxxxxxxxxxxxxx |
| KEYCLOACK\_REDIRECT\_URI | http://127.0.0.1:8000/callback |
| KEYCLOACK\_BASE\_URL | <MyDigital ID SSO DNS host> |
| KEYCLOCK\_REALM | mydid |
| KEYCLOACK\_URL\_AUTHORIZE | https:/ <MyDigital ID SSO DNS host> /realms/mydid/protocol/openid-connect/auth |
| KEYCLOACK\_URL\_ACCESS\_TOKEN | https:// <MyDigital ID SSO DNS host> /realms/mydid/protocol/openid-token |
| KEYCLOACK\_URL\_USER\_INFO | https:// <MyDigital ID SSO DNS host> /realms/mydid/protocol/openidconnect/userinfo |

### 2.3.1 Endpoint

**Table 4: Endpoint List**

|  |  |  |
| --- | --- | --- |
| **Endpoint Name** | **Description** | **Example URL:** **MyDigital ID SSO DNS host** |
| Authorization Endpoint | Used to obtain an authorization code or implicit access token via user  authentication | https:// <MyDigital ID SSO DNS host> /auth/realms/{realm}/protocol/openidconnect/auth |
| Token  Endpoint | Used to exchange authorization codes, refresh tokens, or client credentials for access tokens | https:// <MyDigital ID SSO DNS host> /auth/realms/{realm}/protocol/openidconnect/token |
| Userinfo Endpoint | Returns claims (user profile information) about the authenticated user. | https:// <MyDigital ID SSO DNS host> /auth/realms/{realm}/protocol/openidconnect/userinfo |
| Logout Endpoint | Allows users to log out and revoke their session. | https:// <MyDigital ID SSO DNS host> /auth/realms/{realm}/protocol/openidconnect/logout |
| JWKS  Endpoint | Provides the public keys used to verify JSON Web Tokens (JWTs) | https:// <MyDigital ID SSO DNS host> /auth/realms/{realm}/protocol/openidconnect/certs |

## 2.4 Developer Guide for SSO Integration

### 2.4.1 PHP 7 with Laravel Framework

This configuration guide is specifically applicable to **PHP v7.4.33** and **Laravel framework v8.83.29**. It is important to note that this guide may not be universally compatible with all PHP or Laravel versions. Developers should verify that their environment, including libraries, programming language, and framework versions, aligns with these requirements to ensure successful implementation.

#### 2.4.1.1 Environment Configuration

Keycloak is configured in the *.***env** file. This allows easy management of Keycloak configuration without hardcoding sensitive information in the application code:

A computer screen with white text and numbers

Description automatically generated

**Figure 5: Environment Configuration in PHP Laravel**

#### 2.4.1.2 Library Installation

Install the Keycloak Guard library in the Laravel project folder.

A black background with white text

Description automatically generated

**Figure 6: Keycloak Guard Library Installation**

#### 2.4.1.3 Add Keycloak Driver

Add the configuration for the Keycloak driver in the **auth.php** file to define Keycloak as an authentication provider for the application.

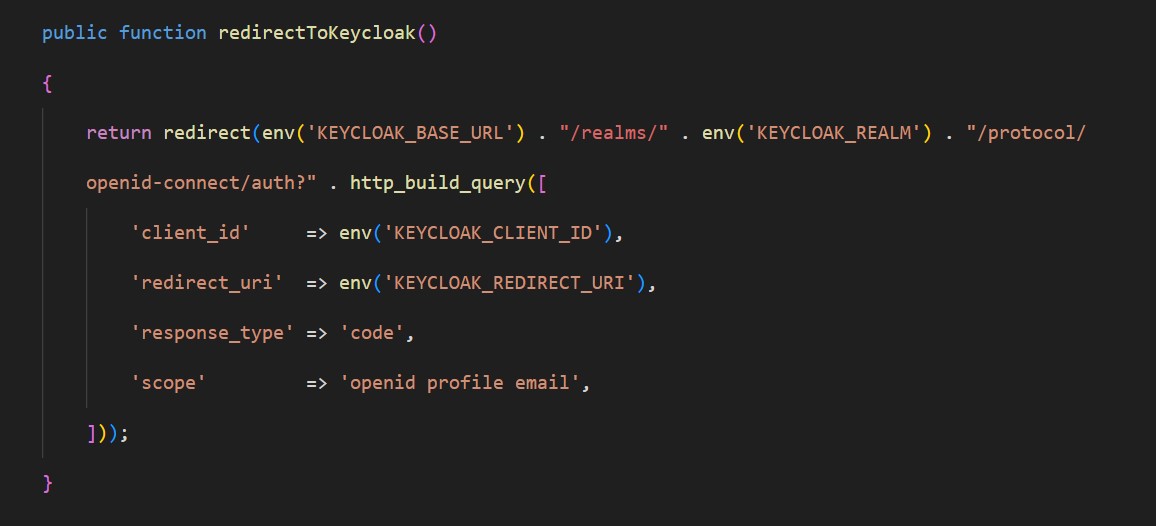
A computer screen shot of text

Description automatically generated

**Figure 7: Keycloak Driver**

#### 2.4.1.4 Create redirectToKeycloak Function

In the **LoginController.php** file, create the redirectToKeycloak function. This function redirects users to the Keycloak Authorization Endpoint to start the login process. It is a key step in integrating Keycloak with a PHP application using the OAuth2 or OpenID Connect protocols, ensuring secure and seamless user authentication.



**Figure 8: *redirectToKeycloak* Function**

#### 2.4.1.5 Create handleKeycloakCallback Function

In the **LoginController.php** file, create a handleKeycloakCallback function. This function processes the Keycloak callback by exchanging the authorization code for an access token, enabling secure user authentication and access to protected resources.

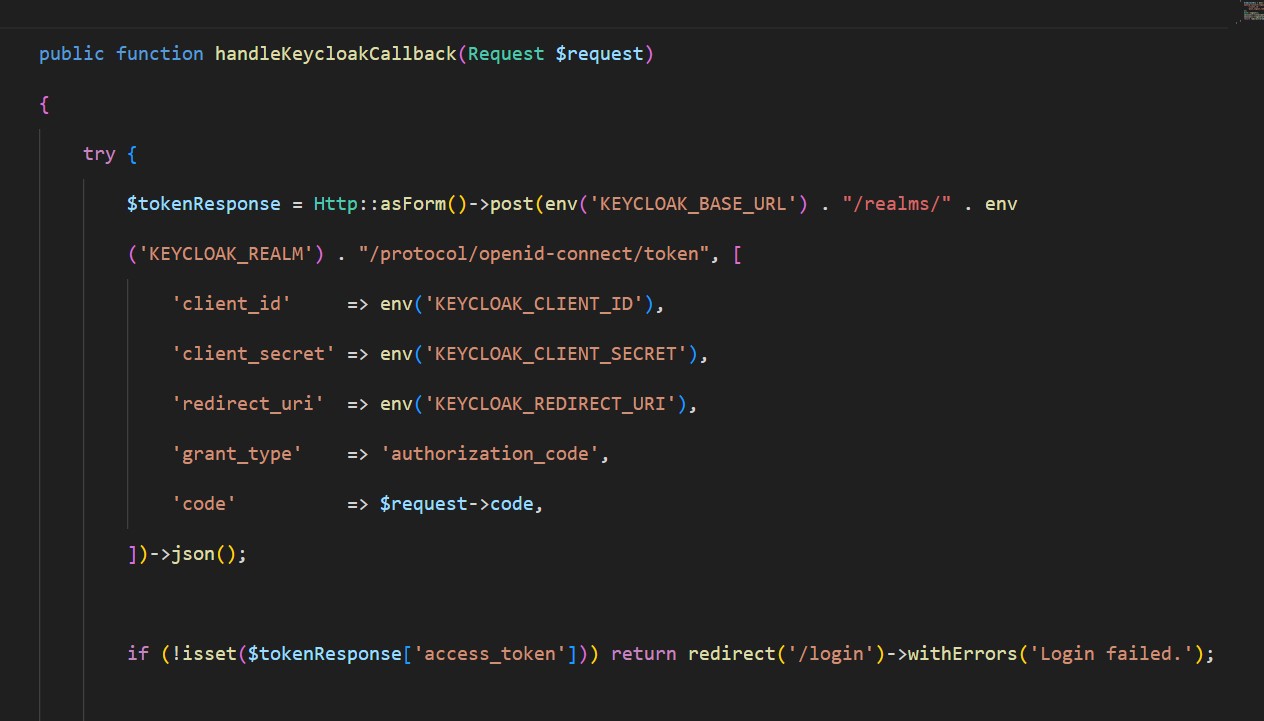


Figure 9: handleKeycloakCallback Function

Retrieve the user information using the access token by querying the Keycloak User Info endpoint. Extract the IC number and name from the response to securely access and utilize these details within the application for user identification and processing.



**Figure 10: Retrieve User Information**

Creates or updates the user in the database. Finds a user based on IC number and if the IC number is found, proceed to log the user in. If the IC number is not found or does not match, handle redirect to the login page with login error message.



**Figure 11: Finds IC Number and Stored in The Database**

#### 2.4.1.6 Add IC Number Column in Model

If the application or website does not store the IC number in the database, the developer can add the IC number column to the protected fillable section in the **User.php** file. This approach ensures that the IC number is securely managed and can be updated within the user model while maintaining data integrity.



**Figure 12: Add IC Number Column**

#### 2.4.1.7 Add New Routes

Add the new routes for all the newly created functions in the **web.php** file to ensure proper routing for actions such as account login and callback.



**Figure 13: Add New Routes**

### 2.4.2 PHP 8 with Laravel Framework

This configuration guide is specifically applicable to **PHP v8.1.25** and **Laravel framework v10.48.12**. It is important to note that this guide may not be universally compatible with all PHP or Laravel versions. Developers should verify that their environment, including libraries, programming language, and framework versions, aligns with these requirements to ensure successful implementation.

#### 2.4.2.1 Environment Configuration

Keycloak is configured in the *.***env** file. This allows easy management of Keycloak configuration without hardcoding sensitive information in the application code:

A computer screen with white text and numbers

Description automatically generated

**Figure 14: Environment Configuration in PHP Laravel**

#### 2.4.2.2 Library Installation

Install the Keycloak Guard library in the Laravel project folder.

A black background with white text

Description automatically generated

**Figure 15: Keycloak Guard Library Installation**

#### 2.4.2.3 Configure Return Variable

Configure the return variables in the **keycloak-web.php** file by retrieving the necessary values from the environment file.

A screen shot of a computer program

Description automatically generated

**Figure 16: Configure the Return Variable**

#### 2.4.2.4 Add Keycloak Driver

Add the configuration for the Keycloak driver in the **auth.php** file to define Keycloak as an authentication provider for the application.

A computer screen shot of text

Description automatically generated

**Figure 17: Keycloak Driver**

#### 2.4.2.5 Create redirectToKeycloak Function

In the **LoginController.php** file, create the redirectToKeycloak function. This function redirects users to the Keycloak Authorization Endpoint to start the login process. It is a key step in integrating Keycloak with a PHP application using the OAuth2 or OpenID Connect protocols, ensuring secure and seamless user authentication.

A screen shot of a computer program

Description automatically generated

**Figure 18: *redirectToKeycloak* Function**

#### 2.4.2.6 Create getAccessTokenFromKeycloak Function

In the LoginController.php file, implement the getAccessTokenFromKeycloak function. This function is responsible for exchanging the authorization code obtained during the authentication process with Keycloak for an access token.

A screen shot of a computer code

Description automatically generated

**Figure 19: *getAccessTokenFromKeycloak* Function**

#### 2.4.2.7 Create getUserInfoFromKeycloak Function

Create a function called getUserInfoFromKeycloak in the **LoginController.php** file. This method is to retrieve user information using an access token.

A screen shot of a computer screen

Description automatically generated

**Figure 20: *getUserInfoFromKeycloak* Function**

#### 2.4.2.8 Create handleKeycloakCallback Function

In the **LoginController.php** file, create a handleKeycloakCallback function. This function processes the Keycloak callback by exchanging the authorization code for an access token, enabling secure user authentication and access to protected resources.

A screen shot of a computer program

Description automatically generated

Figure 21: handleKeycloakCallback Function

Retrieve the user information using the access token by querying the Keycloak User Info endpoint. Extract the IC number and name from the response to securely access and utilize these details within the application for user identification and processing.

A screen shot of a computer program

Description automatically generated

**Figure 22: Retrieve User Information**

Verify if the IC number exists in the users table by checking for a matching record. Ensure that the provided IC number is properly hashed and matches the stored hashed value for accurate user identification and secure authentication.

A computer code with colorful text

Description automatically generated

**Figure 23: Verify IC Number**

Verify if the provided IC number matches the hashed value stored in the database. If a match is found, proceed to log the user in. If the IC number is not found or does not match, handle the necessary steps for linking the user's account or initiating the registration process.

A screen shot of a computer program

Description automatically generated

**Figure 24: Verify IC Number with The Hashed Value Stored in The Database**

#### 2.4.2.9 Add IC Number Column in Model

If the application or website does not store the IC number in the database, the developer can add the IC number column to the protected fillable section in the **User.php** file. This approach ensures that the IC number is securely managed and can be updated within the user model while maintaining data integrity.

A screen shot of a computer

Description automatically generated

**Figure 25: Add IC Number Column**

#### 2.4.2.10 Create updateICNumber Function

Create an updateICNumber function in the **LoginController.php** file to retrieve the IC number from the session. This function will identify users who have not yet linked their IC number, update the user's record with the IC number, log the user in, and then redirect them to the dashboard.

A computer screen with text on it

Description automatically generated

**Figure 26: *updateICNumber* Function**

#### 2.4.2.11 Create verifyAndLinkAccount Function

Create a verifyAndLinkAccount function in the **LoginController.php** file to validate the email and password input. The function will attempt to authenticate the user using the provided credentials. If authentication is successful, the IC number will be updated and linked to the user’s account.

A screen shot of a computer program

Description automatically generated

**Figure 27: *verifyAndLinkAccount* Function**

#### 2.4.2.12 Create registerNewAccount Function

Create a registerNewAccount function in the **LoginController.php** file to check if the IC number and name are in the session. If no user is found, a new account will be created, and the user will be redirected to the dashboard.

A computer screen shot of a program

Description automatically generated

###### Figure 28: *registerNewAccount* Function

#### 2.4.2.13 Add New Routes

Add the new routes for all the newly created functions in the **web.php** file to ensure proper routing for actions such as account registration, login, and IC number linking.

A screen shot of a computer screen

Description automatically generated

**Figure 29: Add New Routes**

### 2.4.3 PHP 8 with CodeIgniter Framework

This configuration guide is specifically applicable to **PHP v8.1.25** and **CodeIgniter framework v4.2.1**. It is important to note that this guide may not be universally compatible with all PHP and CodeIgniter versions. Developers should verify that their environment, including libraries, programming language, and framework versions, aligns with these requirements to ensure successful implementation.

#### 2.4.3.1 Environment Configuration

Configure the keycloak configuration in the *.***env** file:

A computer screen with white text and numbers

Description automatically generated

**Figure 30: Environment Configuration in PHP Laravel**

#### 2.4.3.2 Library Installation

Install ***openid-connect-php*** library in CodeIgniter project folder.

A black background with white text

Description automatically generated

**Figure 31: *openid-connect-php* Library Installation**

#### 2.4.3.3 Create *loginWithKeycloak* Function

Create a loginWithKeycloak function in the **Auth.php** file controller. Initialize the OpenID Connect client and set redirect URI. Authenticate the user and save user data in session to handle login logic.

A screen shot of a computer program

Description automatically generated

**Figure 32: *loginWithKeycloak* Function**

#### 2.4.3.4 Create *getAccessToken* Function

Create a *getAccessToken* function in the **Auth.php** file controller. This method is to exchange authorization code for access token.

A screenshot of a computer program

Description automatically generated

**Figure 33: getAccessToken Function**

#### 2.4.3.5 Create getUserInfo Function

Create a getUserInfo function in the Auth.php file controller. This method to retrieve user info using access token.

A computer screen shot of text

Description automatically generated

**Figure 34: *getUserInfo* Function**

#### 2.4.3.6 Create *keycloakCallback* Function

Create a keycloakCallback function in the **Auth.php** file controller. Fetch the authorization code and access token.

A screen shot of a computer program

Description automatically generated

###### Figure 35: keycloakCallback Function

Fetch user info and validate user information using the access token.

A computer screen with text on it

Description automatically generated

**Figure 36: Fetch userInfo**

Store user data such as IC number and name in session.

A computer screen shot of a black background

Description automatically generated

**Figure 37: Store User Data**

Verify if the IC number matches in the database. If the IC number is found and matches, log the user in. Then, if the IC number is not found or does not match, handle linking or registration.

A computer screen with text

Description automatically generated

**Figure 38: Verify IC Number**

#### 2.4.3.7 Add IC Number Column in Model

Add IC number column in the protected allowed field section in **Auth.php** file model.

A screen shot of a computer program

Description automatically generated

**Figure 39: Add IC Number Column**

#### 2.4.3.8 Create *update\_user* Function

Create an update\_user function in the **Auth.php** file controller. Get IC number from session and find the user who has not linked their IC number yet. Update the user's IC number and log the user in and redirect to Main.

A screenshot of a computer program

Description automatically generated

**Figure 40: *update\_user* Function**

Retrieve the user information using the access token by querying the Keycloak User Info endpoint. Extract the IC number and name from the response to securely access and utilize these details within the application for user identification and processing.

#### 2.4.3.9 Create *linkAccount* Function

Create a linkAccount function in the **Auth.php** file controller. Validate the email and password input. Attempt to authenticate the user using email and password. If the user is authenticated, update the IC number.

A computer screen shot of a program

Description automatically generated

**Figure 41: linkAccount Function**

#### 2.4.3.10 Create *registerWithKeycloak* Function

Create a registerWithKeycloak function in the **Auth.php** file controller. Check if both IC number and name are available in the session. Create a new user if no existing user was found and redirect to the Main.

A screenshot of a computer program

Description automatically generated

**Figure 42: *registerWithKeycloak* Function**

#### 2.4.3.11 Add New Routes

Add new routes for all new functions related in **Routes.php** file.

A computer screen shot of text

Description automatically generated

**Figure 43: Add New Routes**

### 2.4.4 PHP 7 with CodeIgniter Framework

This configuration guide is specifically applicable to **PHP v7.4.33** and **CodeIgniter framework v3.1.11**. It is important to note that this guide may not be universally compatible with all PHP and CodeIgniter versions. Developers should verify that their environment, including libraries, programming language, and framework versions, aligns with these requirements to ensure successful implementation.

#### 2.4.4.1 Configuration Setup

Setup keycloak configuration in **config.php** file in Codeigniter 3 project folder.

A computer screen shot of a black screen with white text

Description automatically generated

**Figure 44: Setup Configuration**

#### 2.4.4.2 Library Installation

Install ***oauth2-client*** library in CodeIgniter 3 project folder.

A black screen with white text

Description automatically generated

**Figure 45: *oauth2-client* Library Installation**

#### 2.4.4.3 Create New Construct

Add a new construct in the **Oauth.php** file controller. Initialize OAuth 2.0 Client and set to return keycloak configuration.

A screen shot of a computer program

Description automatically generated

**Figure 46: Add New Construct**

#### 2.4.4.4 Create *login* Function

Create a login function in the **Oauth.php** file controller. This method to get authorization URL and save the state for validation then redirect to Keycloak for login.

A computer screen with text on it

Description automatically generated

**Figure 47: *login* Function**

#### 2.4.4.5 Create *callback* Function

Create a callback function in **Oauth.php** file controller. Verify state parameters and get an access token using the authorization code grant.

A computer screen with text on it

Description automatically generated

**Figure 48: *callback* Function**

Fetch user information and check if hashed IC number exists in the administrators table.

A computer screen with text on it

Description automatically generated

**Figure 49: Fetch User Information**

Verify if the hashed IC number matches in the database. If the IC number is found and matches, log the user in. Then, if the IC number is not found or does not match, handle linking or registration.

A computer screen shot of text

Description automatically generated

**Figure 50: Verify IC Number**

#### 2.4.4.6 Create *show\_modal\_options* Function

Create show\_modal\_options function in **Oauth.php** file controller to load header, modal, and footer views.

A computer screen shot of text

Description automatically generated

**Figure 51: show\_modal\_options Functions**

#### 2.4.4.7 Create *link\_existing\_account* Function

Create link\_existing\_account function in **Oauth.php** file controller. Retrieve user info stored in session, validate the email and password input. Attempt to authenticate the user using email and password. If the user is authenticated, update the IC number.

A screen shot of a computer screen

Description automatically generated

**Figure 52: *link\_existing\_account* Function**

#### 2.4.4.8 Create *register\_new\_account* Function

Create register\_new\_account function in **Oauth.php** file controller. Check if both IC number and name are available in the session. Create a new user if no existing user was found and redirected to the dashboard.

A screen shot of a computer program

Description automatically generated

###### Figure 53: *register\_new\_account* Function Part 1

A screen shot of a computer program

Description automatically generated

**Figure 54: *register\_new\_account* Function Part 2**