Replacing the authentication service

To replace the authentication service with another authentication service, edit the following files based on the type of your new service.

When changing to another OpenID Connect service, follow those three steps:

1. Register an OpenID Connect client at the provider of your choice. You should be able to specify a redirect URI, which will be called after login was successful. Specify **src/views/login_callback.php** as redirect URI here.
2. Configure the login button in our frontend. Find all placeholder **span** elements for **oidc-button**. You can recognize them by their CSS class which is **oidc-button**.
   Set **data-name** to the name of the new OpenID Connect service
   Set **data-logo** to a URI to a logo of the new OpenID Connect service
   Set **data-client-id** to the Client ID you received when registering the client in step 1
3. Our server is implemented to be able to support multiple authentication services. Therefore, **signin_callback.js** sends a **service_type** parameter to our server to identify which service has been used to authenticate. The name of the **service_type** and all references should be renamed. (E.g. change from 'LearningLayers' to 'Google')

You can find the documentation of oidc-connect-button on github under
https://github.com/learning-layers/openid-connect-button

When changing to another authentication service. We assume, that the authentication service works token based:

1. Replace the existing login buttons by the new frontend component(s) for the new login. The documentation on https://github.com/learning-layers/openid-connect-button will help you find all frontend components which belong to the existing login button.
2. After completing the authentication with your new authentication service, you need to send a **POST** request to **'../php/create_user_session.php'** with two parameters **access_token** and **service_type**. This can be achieved with JQuery:
   jQuery.post('../php/create_user_session.php', {access_token:*token***,**
   service_type:*service_name***},** *optional_callback_function***);**
3. If your authentication service redirects users away from Anatomy2.0, but not back exactly to the page the user came from, you can use the following two helper to achieve a proper redirect:
   Before redirecting to the new authentication service, save the page the user currently is on:
   jQuery.post('../php/register_redirect_url.php', {requestUri:window.location.href},
   function(data) {});
   When returning from the new authentication service page, redirect to the old page with the following JavaScript:
   window.location.replace("../php/login_redirect.php");
4. Reimplement the two functions in **authentication.php** according to their JavaDoc description.

Adding another authentication service

To add another authentication service in addition to Learning Layers, do the following steps. It is assumed, that the new authentication service works token based:

1. Add frontend component(s) which allow using the new authentication service. All places which currently use **oidc-button** are to be considered. If the new authentication service is an OpenID Connect service, then consider using another **oidc-button** for the new login (see https://github.com/learning-layers/openid-connect-button).
2. After users log in with the new authentication service, you need to send a **POST** request to **'../php/create_user_session.php'** with two parameters **access_token** and **service_type**. This can be achieved with JQuery:
   jQuery.post('../php/create_user_session.php', {access_token:*token***, service_type:***service_name***},** *optional_callback_function***);**
3. If your authentication service redirects users away from Anatomy2.0, but not back exactly to the page the user came from, you can use the following two helper to achieve a proper redirect:
   Before redirecting to the new authentication service, save the page the user currently is on:
   jQuery.post('../php/register_redirect_url.php', {requestUri:window.location.href}, function(data) {});
   When returning from the new authentication service page, redirect to the old page with the following JavaScript:
   window.location.replace("../php/login_redirect.php");
4. Implement the two functions in **authentication.php** according to their JavaDoc description for your new **service_type** as specified in step 2.

Understanding the authentication / authorization implementation

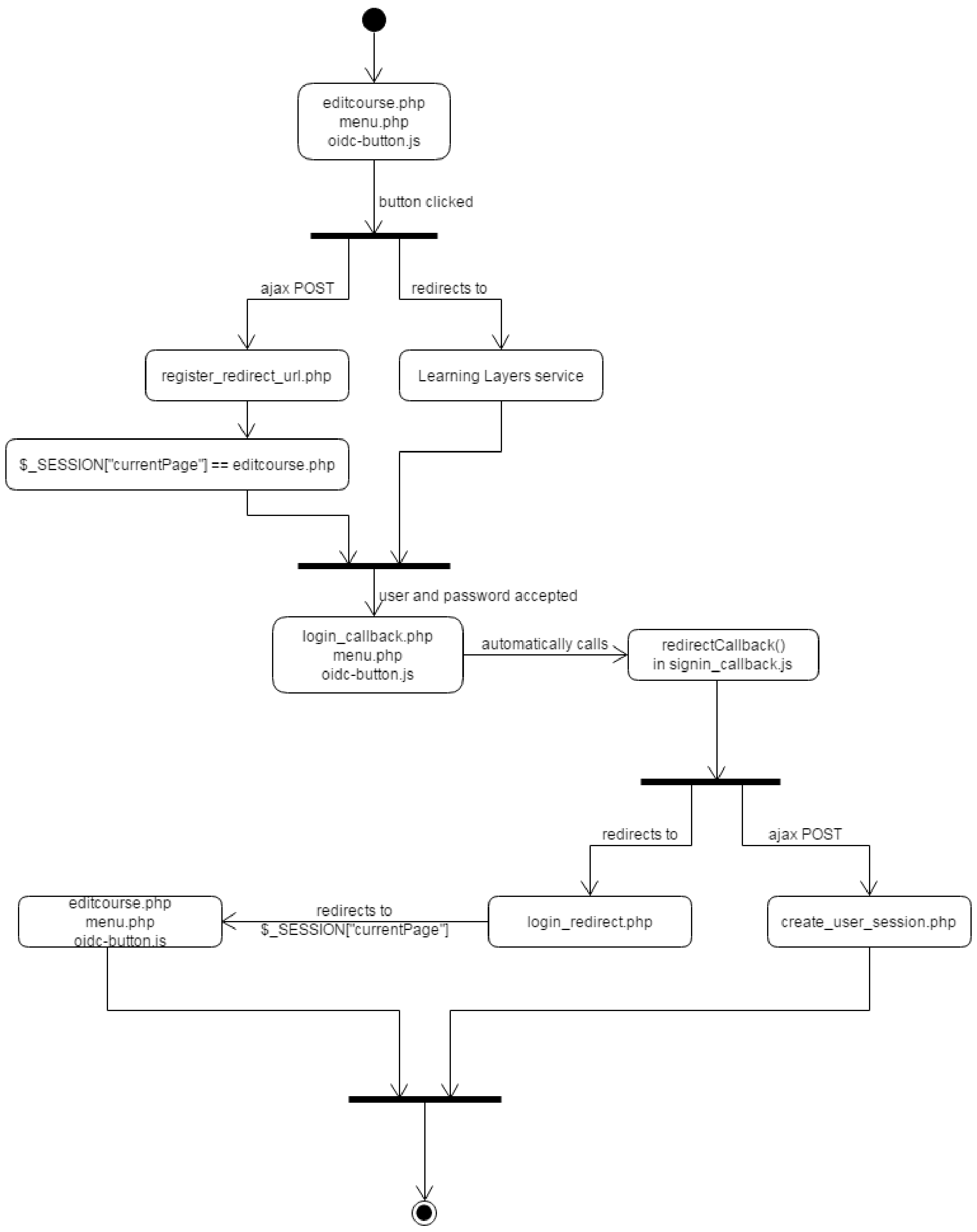
Login

The following diagram shows the login procedure. It is assumed, that the user currently views the editcourse.php page. The login works exactly the same way on all other pages.

The basic login with user name and password is handled by Learning Layers. Oidc-button also automatically shows the login status (the user name after the user has logged in).

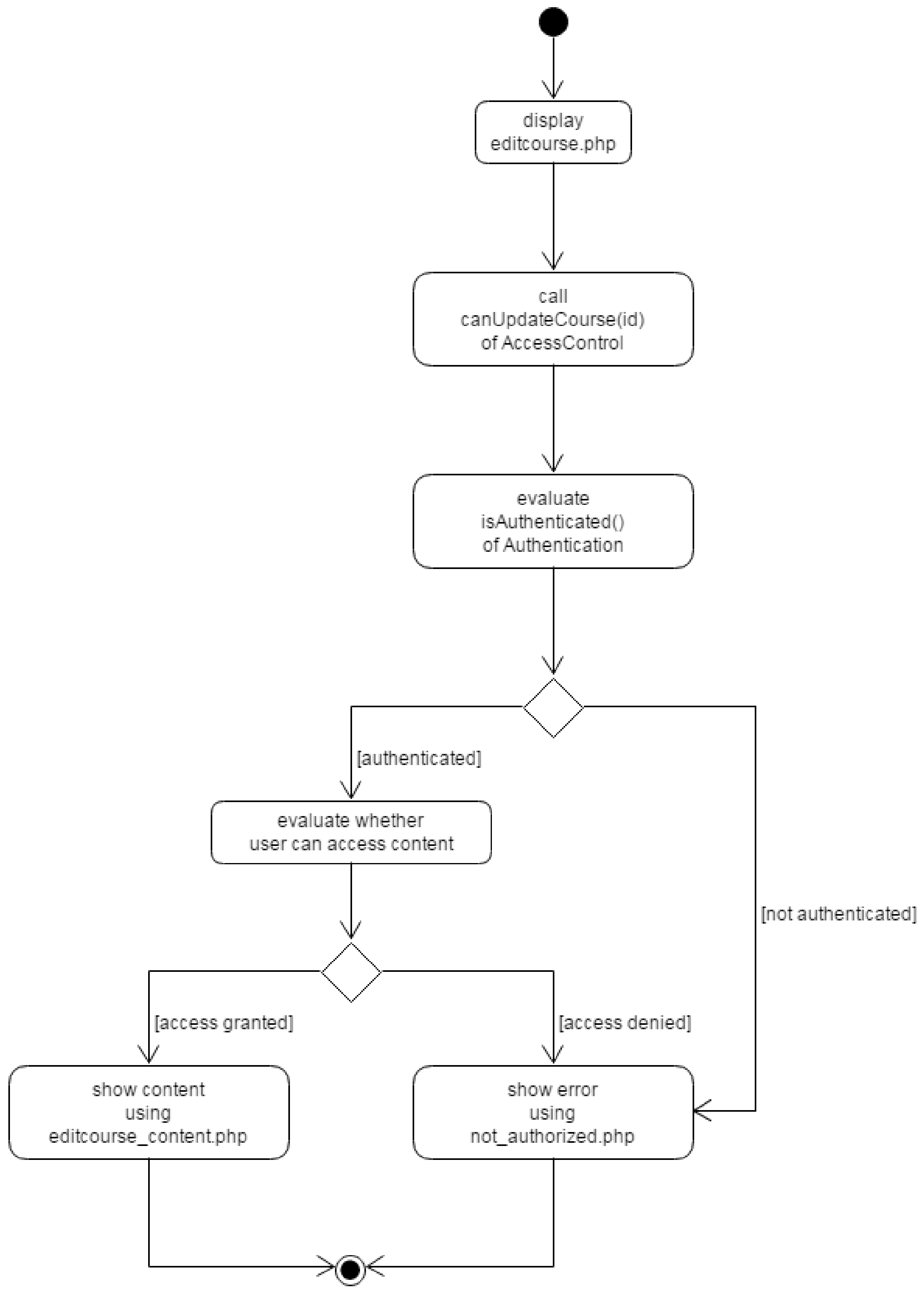
Two other features are implemented by us:

1. Learning Layers always redirects to only one fixed page after the user provided user name and password. But we want to redirect to the page where the user was on (which can be any page). Therefore, when a user clicks the login button, we save the current page on our server via session storage. This is done in **register_redirect_url.php.**
   Learning Layers will redirect the user to **login_callback.php**, which will redirect the user again by calling **login_redirect.php**. **login_redirect.php** has access to session storage and can therefore redirect to the page the user came from.
2. Anatomy2.0 saves additional properties of users which the Learning Layers service cannot store. Therefore, a database entry needs to be created for each user of our application. This database entry also needs to store email and name of users (although they can be received from Learning Layers). This is, because email and name of course creators or model uploaders should be shown to other users, which cannot access this data from Learning Layers. To save this data, **create_user_session.php** creates and updates database entries.



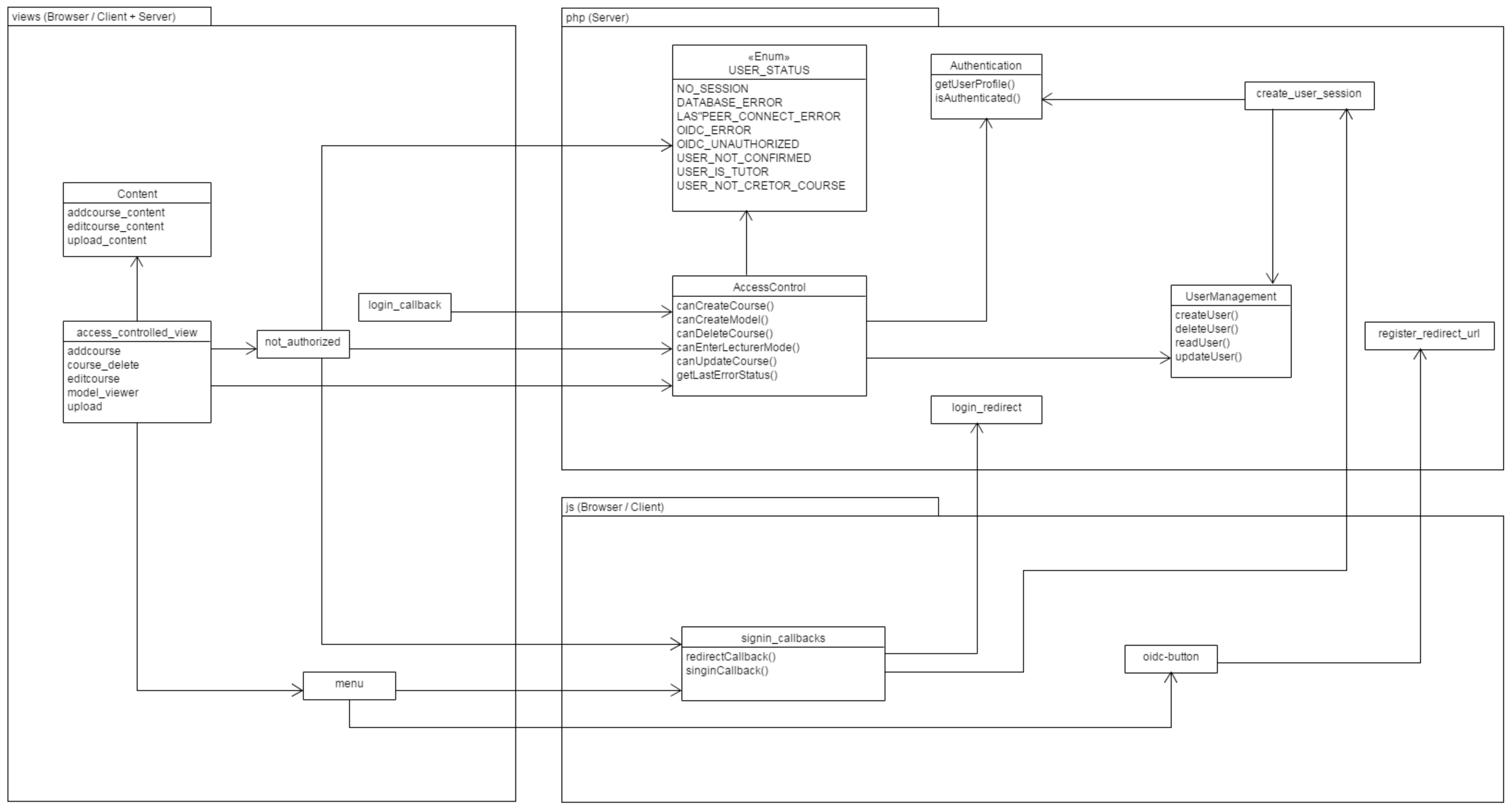
Authentication and Access Control

To make sure users access (especially change and delete) only those objects that they are allowed to, Anatomy 2.0 implements authentication and access control mechanisms. They are handled in **authentication.php** and **access_control.php**. The control flow is described in the picture below.


display editcourse.php
call canUpdateCourse(id) of AccessControl
evaluate isAuthenticated() of Authentication
[authenticated]
[not authenticated]
evaluate whether user can access content
[access granted]
[access denied]
show content using editcourse_content.php
show error using not_authorized.php

User management classes

To give an overview of the dependencies between all classes which are concerned with user management, the following diagram has been created. Please note, that most classes in the diagram are not real classes, but just files.


views (Browser / Client + Server)
Content
addcourse_content
editcourse_content
upload_content
access_controlled_view
addcourse
course_delete
editcourse
model_viewer
upload
not_authorized
login_callback
menu
php (Server)
«Enum»
USER_STATUS
NO_SESSION
DATABASE_ERROR
LAS"PEER_CONNECT_ERROR
OIDC_ERROR
OIDC_UNAUTHORIZED
USER_NOT_CONFIRMED
USER_IS_TUTOR
USER_NOT_CRETOR_COURSE
Authentication
getUserProfile()
isAuthenticated()
create_user_session
AccessControl
canCreateCourse()
canCreateModel()
canDeleteCourse()
canEnterLecturerMode()
canUpdateCourse()
getLastErrorStatus()
UserManagement
createUser()
deleteUser()
readUser()
updateUser()
register_redirect_url
login_redirect
js (Browser / Client)
signin_callbacks
redirectCallback()
singinCallback()
oidc-button