

## One Ring to Bring Them All

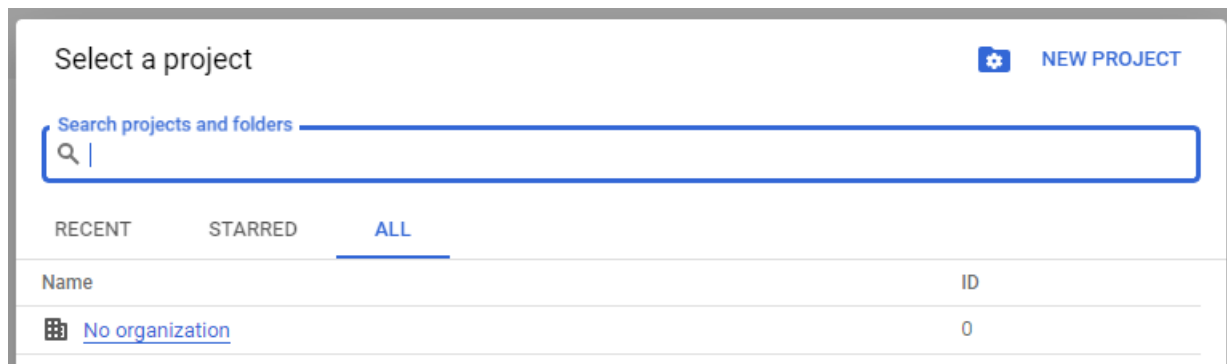
### Introduction

In the ever-evolving landscape of modern digital business operations, the need for robust and secure user authentication systems has become paramount. In response to this, Pandora Company Limited embarked on a strategic initiative to streamline user authentication processes across their applications. The project aimed to integrate a centralized user management system with OAuth2, a widely adopted and highly secure authentication protocol, offering users the convenience of logging in with their Google accounts. This report provides a comprehensive overview of the implementation, detailing the architecture, components, and security considerations, as well as outlining the key achievements and future directions of this transformative project. By enhancing user authentication through OAuth2 integration, Pandora Company Limited is poised to elevate user experience, strengthen security, and pave the way for further technological advancements in their application ecosystem.

### The Process

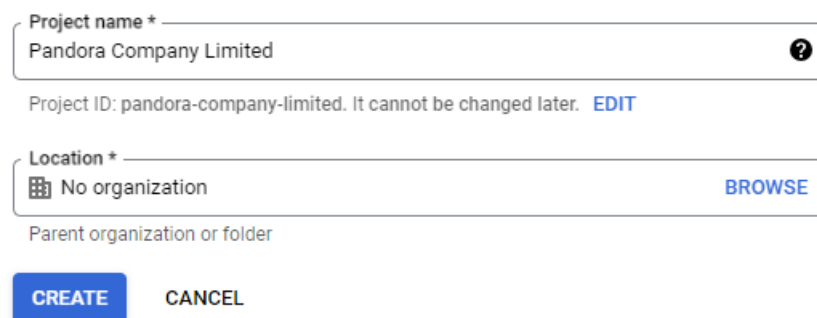
#### Set Up the Google API Project and Obtain OAuth2 Client Credentials

- Visit the Google Developers Console at <https://console.developers.google.com/>.
- Create a new project by clicking the project drop-down and selecting "New Project."



Name	ID
No organization	0

- Fill following details and create new project



Project name \*  
Pandora Company Limited

Project ID: pandora-company-limited. It cannot be changed later. [EDIT](#)

Location \*  
 No organization [BROWSE](#)

Parent organization or folder

[CREATE](#) [CANCEL](#)

- Then click on the "OAuth consent screen" tab in your project.
- Fill out the required information, including the application name and developer email. You can also add details like the application logo and user support email.

## App information

This shows in the consent screen, and helps end users know who you are and contact you

App name \*  
Pandora Company Limited

The name of the app asking for consent

User support email \*  
cmallawaarachchi0@gmail.com

For users to contact you with questions about their consent

## App logo

This is your logo. It helps people recognize your app and is displayed on the OAuth consent screen.

After you upload a logo, you will need to submit your app for verification unless the app is configured for internal use only or has a publishing status of "Testing". [Learn more](#)

Logo file to upload

BROWSE

Upload an image, not larger than 1MB on the consent screen that will help users recognize your app. Allowed image formats are JPG, PNG, and BMP. Logos should be square and 120px by 120px for the best results.

## App domain

To protect you and your users, Google only allows apps using OAuth to use Authorized Domains. The following information will be shown to your users on the consent screen.

- Likewise fill all four documentation steps and finish the configurations.
- After setting up your Google API Project, you need to obtain OAuth2 client credentials. This involves creating a set of **client ID** and **client secret** for your application.
- In the Google Developers Console, go to the "Credentials" tab for your project.
- Click the "Create Credentials" button and select "OAuth client ID."

+ CREATE CREDENTIALS

DELETE

RESTORE DELETED CRE

API key

Identifies your project using a simple API key to check quota and access

OAuth client ID

Requests user consent so your app can access the user's data

Service account

Enables server-to-server, app-level authentication using robot accounts

Help me choose

Asks a few questions to help you decide which type of credential to use

- Select "Web application" as the application type. This is suitable for a PHP-based web application.
- Add the authorized redirect URIs for your application. These are the URIs where Google will redirect the user after successful authentication. Typically, this would include the URL for your '**redirect.php**' page (e.g., 'http://localhost:3000/redirect.php').

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See [Setting up OAuth 2.0](#) for more information. [Learn more](#) about OAuth client types.

Application type \*  
Web application

Name \*  
Web client 1

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

**i** The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorized domains](#).

### Authorized JavaScript origins **?**

For use with requests from a browser

**+ ADD URI**

### Authorized redirect URIs **?**

For use with requests from a web server

URIs 1 \*  
http://localhost:3000/redirect.php

**+ ADD URI**

Note: It may take 5 minutes to a few hours for settings to take effect

**CREATE**

**CANCEL**

- After configuring the OAuth client ID, you will be provided with a "Client ID" and "Client Secret." These are the credentials you'll use in your PHP application. If you want you can download the **JSON** file also.

### OAuth client created

The client ID and secret can always be accessed from Credentials in APIs & Services

**i** OAuth access is restricted to the [test users](#) listed on your [OAuth consent screen](#)

Client ID	760852947507-9p0I2ihmiiqglb8k2mstmrqjcbvnfcb.apps.googleusercontent.com
Client secret	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
Creation date	October 22, 2023 at 12:21:49 AM GMT+5
Status	Enabled

**↓ DOWNLOAD JSON**

**OK**

## Setup the PHP Environment for the Project

- Install PHP and its necessary modules. On Ubuntu, you can install PHP with,

```
sudo apt install php
```

- You'll also want to install some common PHP extensions. For your OAuth2 implementation, you should make sure that the **'curl'** extension is enabled. You can install it with,

```
sudo apt install php-curl
```

- Then install the composer

```
sudo apt install composer
```

- After all these, initiate the project by giving relevant details, and packages that have to install and create the **composer.json** file.

```
composer init
```

- Then install the packages and relevant dependencies that are mentioned in the composer.json file by

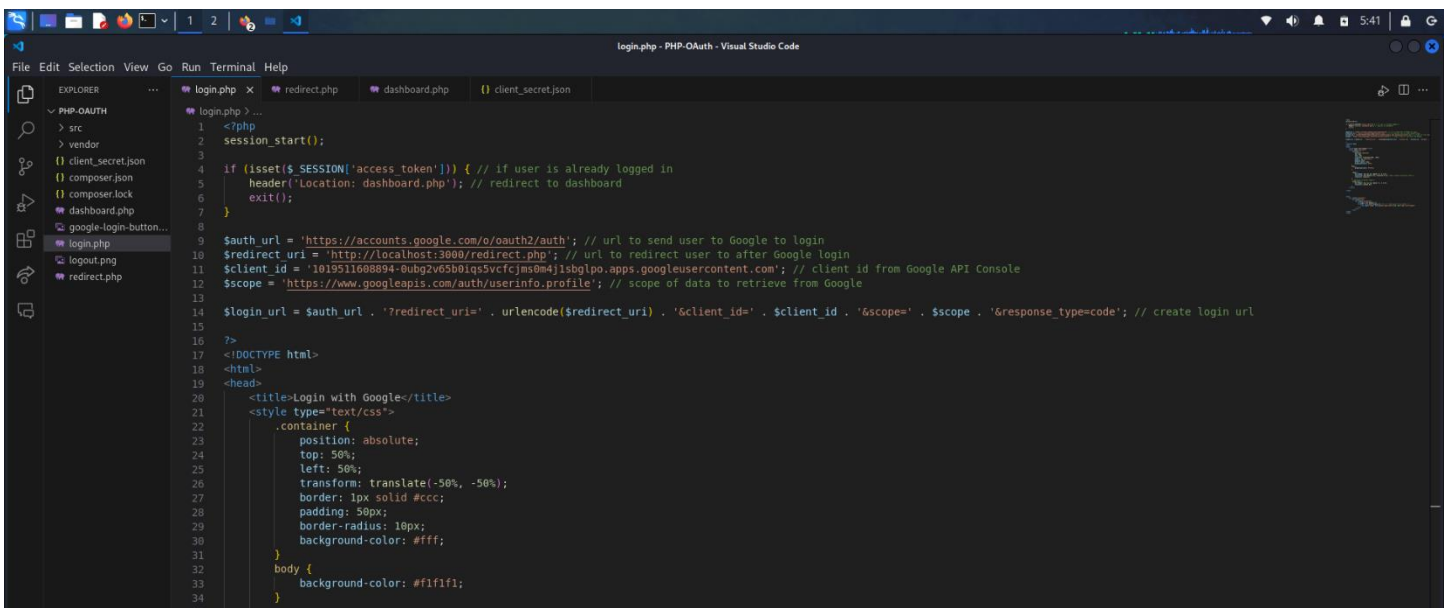
```
composer install
```

- After this process you can create the PHP environment that want to develop the application.
- Then you can run PHP server by,

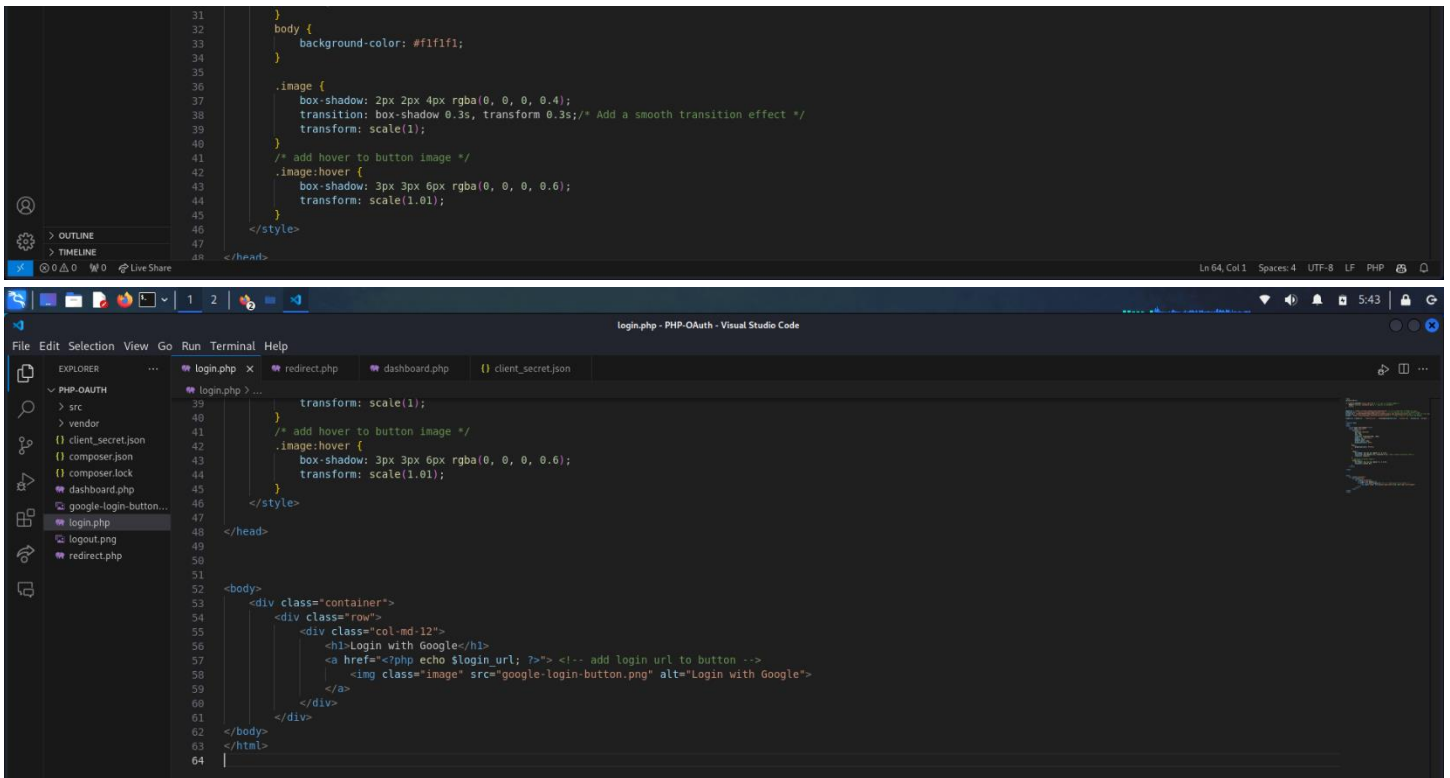
```
php -S localhost:3000 -t .
```

## Develop the PHP-Based Application Components and Implement "Login with Google" Functionality

- Create the following PHP-based application files:
  - **login.php**: The entry point with a "Login with Google" button.



```
login.php - PHP-OAuth - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  PHP-OAUTH
    src
    vendor
    client_secret.json
    composer.json
    composer.lock
    dashboard.php
    google-login-button...
    login.php
    logout.png
    redirect.php
login.php
1 <?php
2 session_start();
3
4 if (isset($_SESSION['access token'])) { // If user is already logged in
5     header('Location: dashboard.php'); // redirect to dashboard
6     exit();
7 }
8
9 $auth_url = 'https://accounts.google.com/o/oauth2/auth'; // url to send user to Google to login
10 $redirect_url = 'http://localhost:3000/redirect.php'; // url to redirect user to after Google login
11 $client_id = '1019511608894-0ubg2v65b0iq55vcfcjms0m41sbglpo.apps.googleusercontent.com'; // client_id from Google API Console
12 $scope = 'https://www.googleapis.com/auth/userinfo.profile'; // scope of data to retrieve from Google
13
14 $login_url = $auth_url . '?redirect_url=' . urlencode($redirect_url) . '&client_id=' . $client_id . '&scope=' . $scope . '&response_type=code'; // create login url
15
16 ?>
17 <!DOCTYPE html>
18 <html>
19 <head>
20 <title>Login with Google</title>
21 <style type="text/css">
22 .container {
23     position: absolute;
24     top: 50%;
25     left: 50%;
26     transform: translate(-50%, -50%);
27     border: 1px solid #ccc;
28     padding: 50px;
29     border-radius: 10px;
30     background-color: #fff;
31 }
32 body {
33     background-color: #f1f1f1;
34 }
35
```

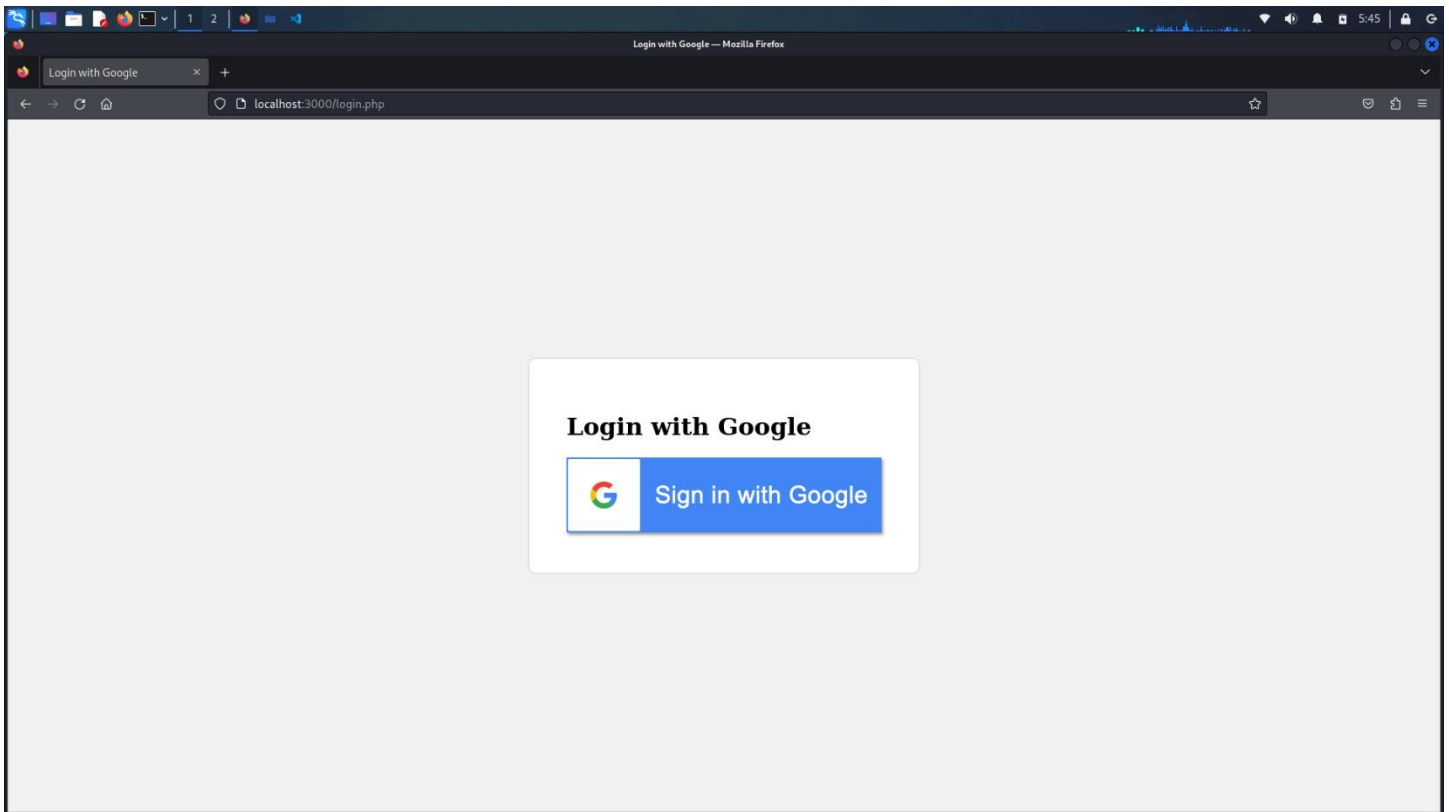


The screenshot shows the Visual Studio Code editor with two files open: `login.php` and `login.html`. The `login.php` file contains PHP code for session management and OAuth2 authentication. The `login.html` file contains the HTML structure for the login page, including a centered container with a login button that has a hover effect.

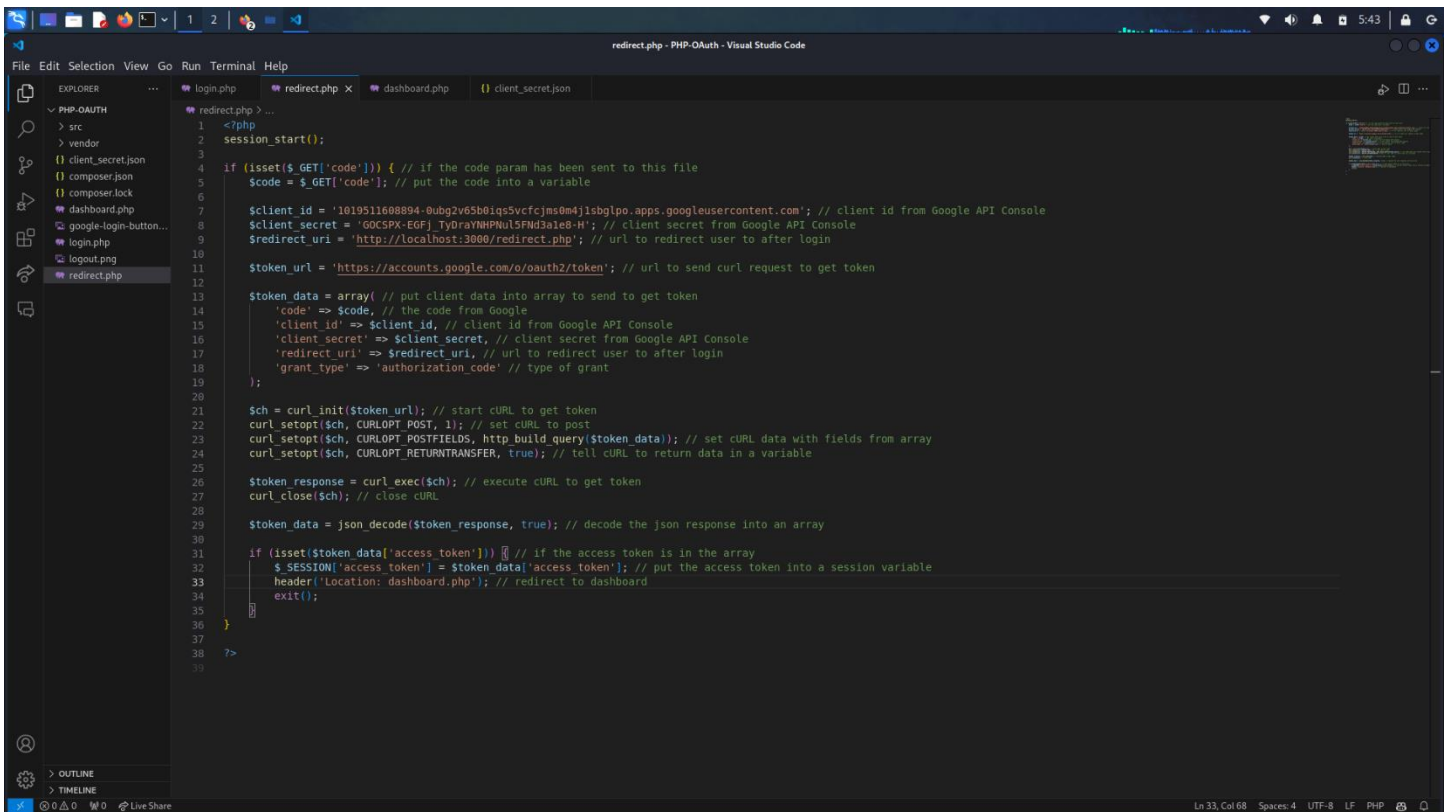
```
31 }
32 body {
33     background-color: #f1f1f1;
34 }
35
36 .image {
37     box-shadow: 2px 2px 4px rgba(0, 0, 0, 0.4);
38     transition: box-shadow 0.3s, transform 0.3s; /* Add a smooth transition effect */
39     transform: scale(1);
40 }
41 /* add hover to button image */
42 .image:hover {
43     box-shadow: 3px 3px 6px rgba(0, 0, 0, 0.6);
44     transform: scale(1.01);
45 }
46
47 </style>
48 </head>
49
50 <body>
51
52 <div class="container">
53     <div class="row">
54         <div class="col-md-12">
55             <h1>Login with Google</h1>
56             <a href="/php echo $login_url; ?"> <!-- add login url to button -->
57                 
59         </div>
60     </div>
61 </body>
62 </html>
63
64
```

1. 'session\_start();' initiates a session to store user data such as the access token, allowing users to remain logged in across different pages of the application.
2. It checks if the user is already logged in by verifying if the 'access\_token' session variable is set. If the user is already logged in, it redirects them to the 'dashboard.php' page to bypass the login process.
3. Then define several variables used in the OAuth2 authentication process:
  - a. '\$auth\_url' is the URL where users are sent to log in with their Google accounts.
  - b. '\$redirect\_uri' is the URL where Google will redirect the user after successful authentication.
  - c. '\$client\_id' is the client ID obtained from the Google API Console.
  - d. '\$scope' specifies the scope of data that will be retrieved from Google. In this case, it's the user's profile information.
4. It constructs the '\$login\_url' by combining the above variables to create a URL that users can click to initiate the OAuth2 authentication process.
5. The HTML part of the code defines the structure of the login page:
  - a. It sets the page title to "Login with Google."
  - b. It includes some inline CSS for styling, creating a centered container for the login button.
  - c. The login button is an image (with hover effect) that, when clicked, redirects users to the Google login page using the constructed '\$login\_url'.

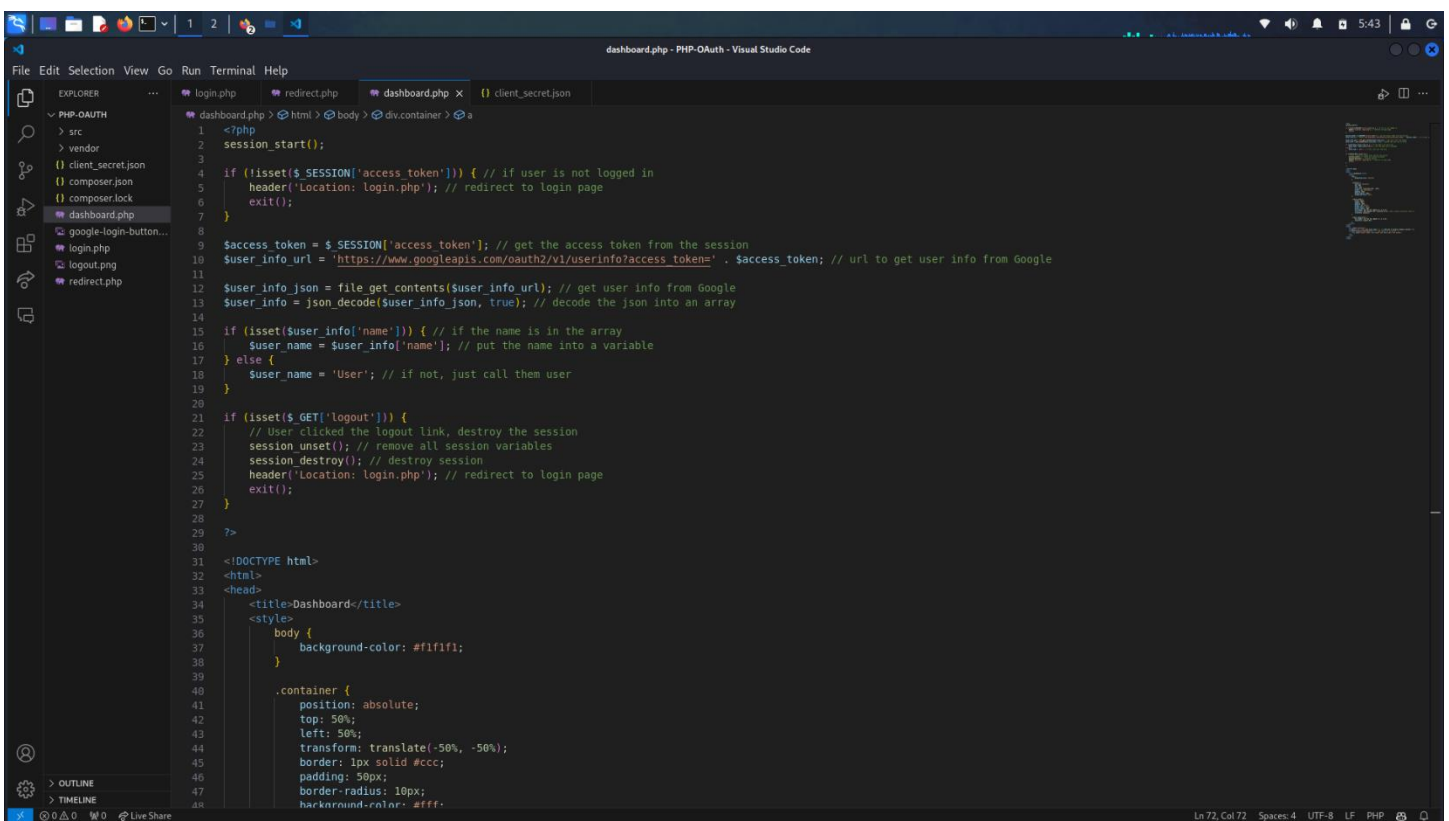
- This is how the **login.php** looks like,



- **redirect.php**: Handles the Google OAuth2 redirect and exchanges the authorization code for an access token.



1. 'session\_start();' initiates a session to store the user's access token for future use.
2. Then checks if the 'code' parameter is present in the URL. This parameter is sent by Google after the user successfully logs in and consents to the application's access request.
3. If the 'code' parameter is present, it retrieves the code from the URL and assigns it to the '\$code' variable. This code is a one-time use authorization code generated by Google.
4. Then defines several variables necessary for the OAuth2 token exchange:
  - a. '\$client\_id' is the client ID obtained from the Google API Console.
  - b. '\$client\_secret' is the client secret from the Google API Console.
  - c. '\$redirect\_uri' is the URL where Google will redirect the user after successful authentication.
  - d. '\$token\_url' is the URL where a cURL request will be sent to obtain the access token.
5. Then creates an array named '\$token\_data' to hold the parameters required to obtain the access token. These parameters include the 'code', 'client\_id', 'client\_secret', 'redirect\_uri', and 'grant\_type' ('authorization\_code').
6. After initializes a cURL session using the '\$token\_url' and configures it to perform an HTTP POST request, passing the '\$token\_data' as the request body. This is how it sends the authorization code to Google to request an access token.
7. The cURL request is configured to return the response from the token request into the '\$token\_response' variable.
8. The cURL request is executed with 'curl\_exec(\$ch)'.
9. After obtaining the response, the cURL session is closed with 'curl\_close(\$ch)'.
10. The '\$token\_response', which is in JSON format, is decoded into an array using 'json\_decode'.
11. Then checks if the array contains an 'access\_token'. If the access token is present, it means the OAuth2 authentication was successful.
12. If an access token is found, it is stored in the user's session using '\$\_SESSION['access\_token']'.
13. Finally, the user is redirected to the 'dashboard.php' page, and the script exits.
  - **dashboard.php:** The landing page after successful login, displaying a personalized greeting to the user.

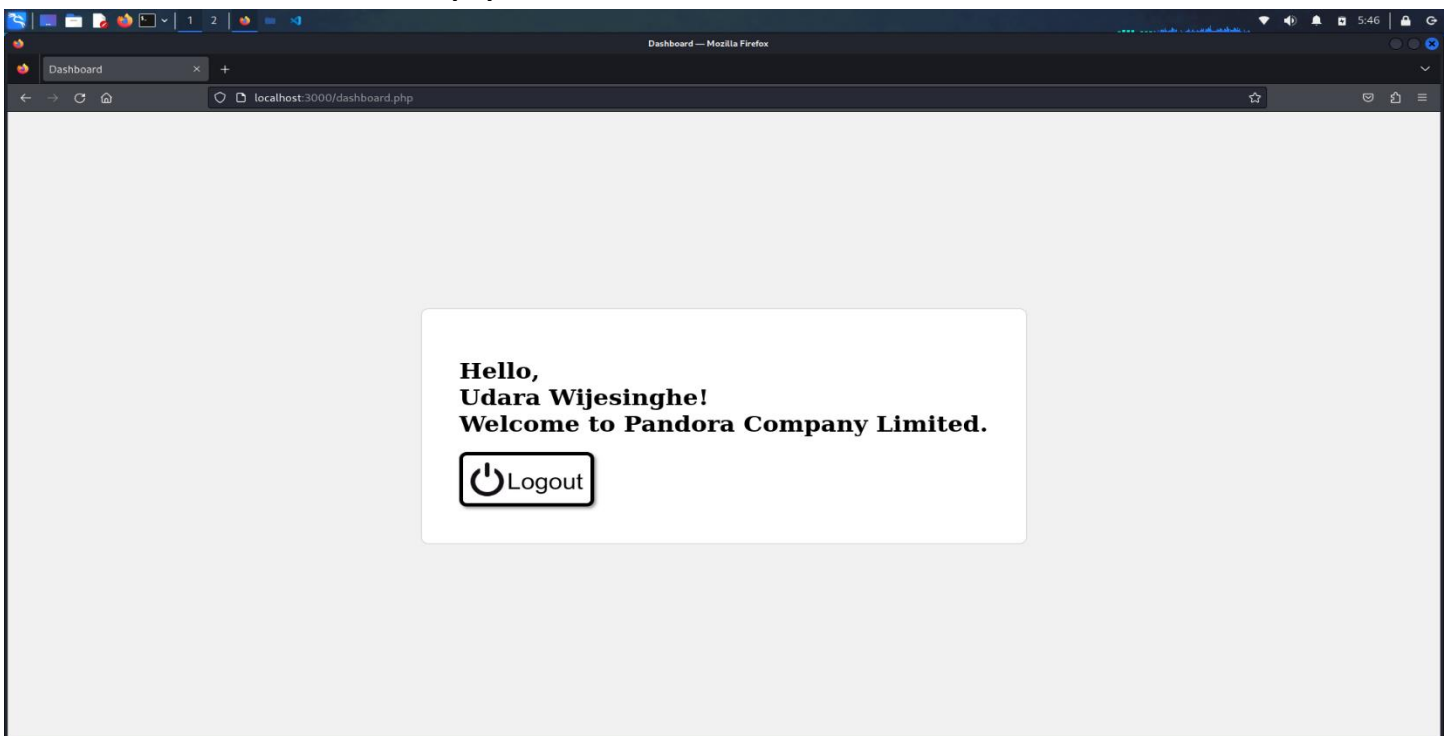




```
46         padding: 50px;
47         border-radius: 10px;
48         background-color: #fff;
49     }
50
51     .logout-image {
52         width: 150px;
53         height: 50px;
54         padding: 10px;
55         border: 5px solid;
56         border-color: black;
57         border-radius: 10px;
58         box-shadow: 2px 2px 4px rgba(0, 0, 0, 0.4);
59         transition: box-shadow 0.3s, transform 0.3s; /* Add a smooth transition effect */
60         transform: scale(1);
61     }
62
63     .logout-image:hover {
64         box-shadow: 3px 3px 6px rgba(0, 0, 0, 0.6);
65         transform: scale(1.01);
66     }
67 }
68 </style>
69 </head>
70 <body>
71     <div class="container">
72         <h1>Hello,</br> <?php echo $user_name; ?>! </br>Welcome to Pandora Company Limited.</h1>
73         <a href="dashboard.php?logout=1"> <!-- add logout url to button -->
74             
75         </a>
76     </div>
77 </body>
78 </html>
```

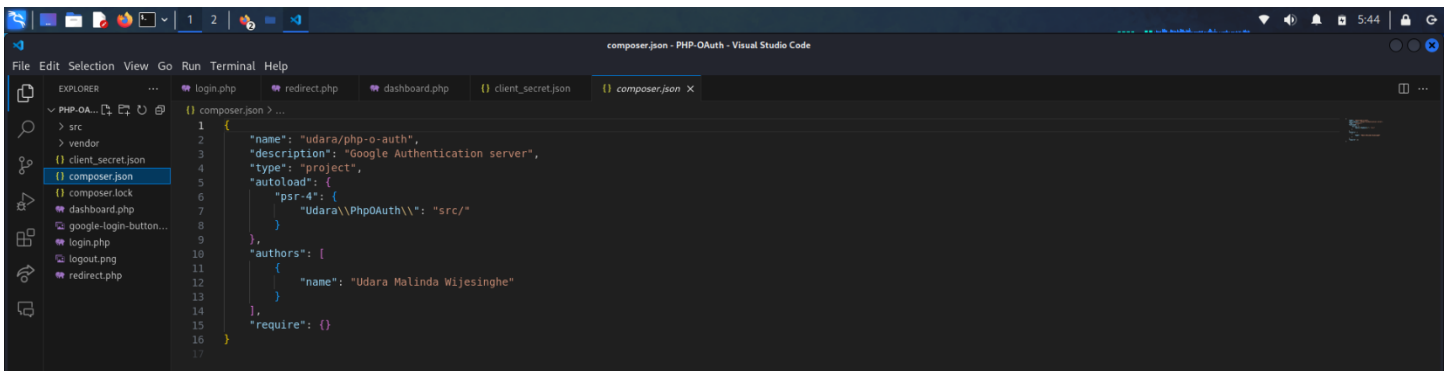
1. First checks if the user is not logged in (no access token in the session). If not logged in, it redirects to the login page ('login.php').
2. If the user is logged in, it retrieves the access token from the session and constructs a URL to fetch the user's information from Google's OAuth2 API.
3. Then uses 'file\_get\_contents' to make an HTTP request to that URL, fetching the user information in JSON format.
4. The JSON response is decoded into an array, and it checks if the user's name is present in the array. If the name is found, it's assigned to the '\$user\_name' variable. If not, the user is addressed as "User."
5. If the user clicks the "Logout" button, it clears the session (removes all session variables) and destroys the session, ensuring the user is logged out. The user is then redirected to the login page.
6. The HTML part defines the structure of the dashboard page, including styles for layout and the appearance of the "Logout" button.
7. A personalized greeting is displayed, addressing the user by name.
8. Users can log out by clicking a button, and the button has a hover effect for better user interaction.

- This is how the **dashboard.php** looks like,





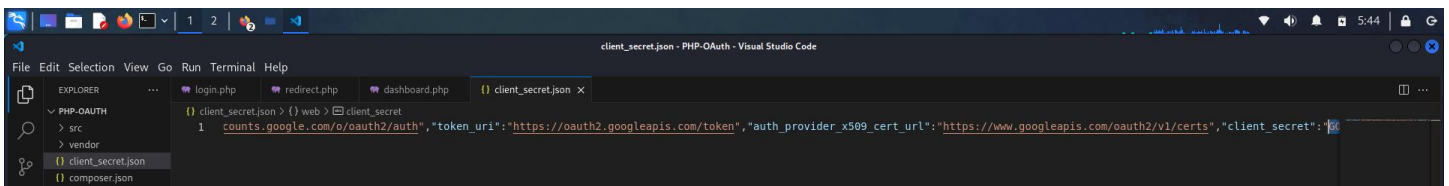
- This is the 'composer.json' file



The screenshot shows the Visual Studio Code editor with the 'composer.json' file open. The file contains the following JSON content:

```
{
  "name": "udara/php-o-auth",
  "description": "Google Authentication server",
  "type": "project",
  "autoload": {
    "psr-4": {
      "Udara\\PhpOAuth\\": "src/"
    }
  },
  "authors": [
    {
      "name": "Udara Malinda Wijesinghe"
    }
  ],
  "require": {}
}
```

- This is the 'client\_secret.json' file

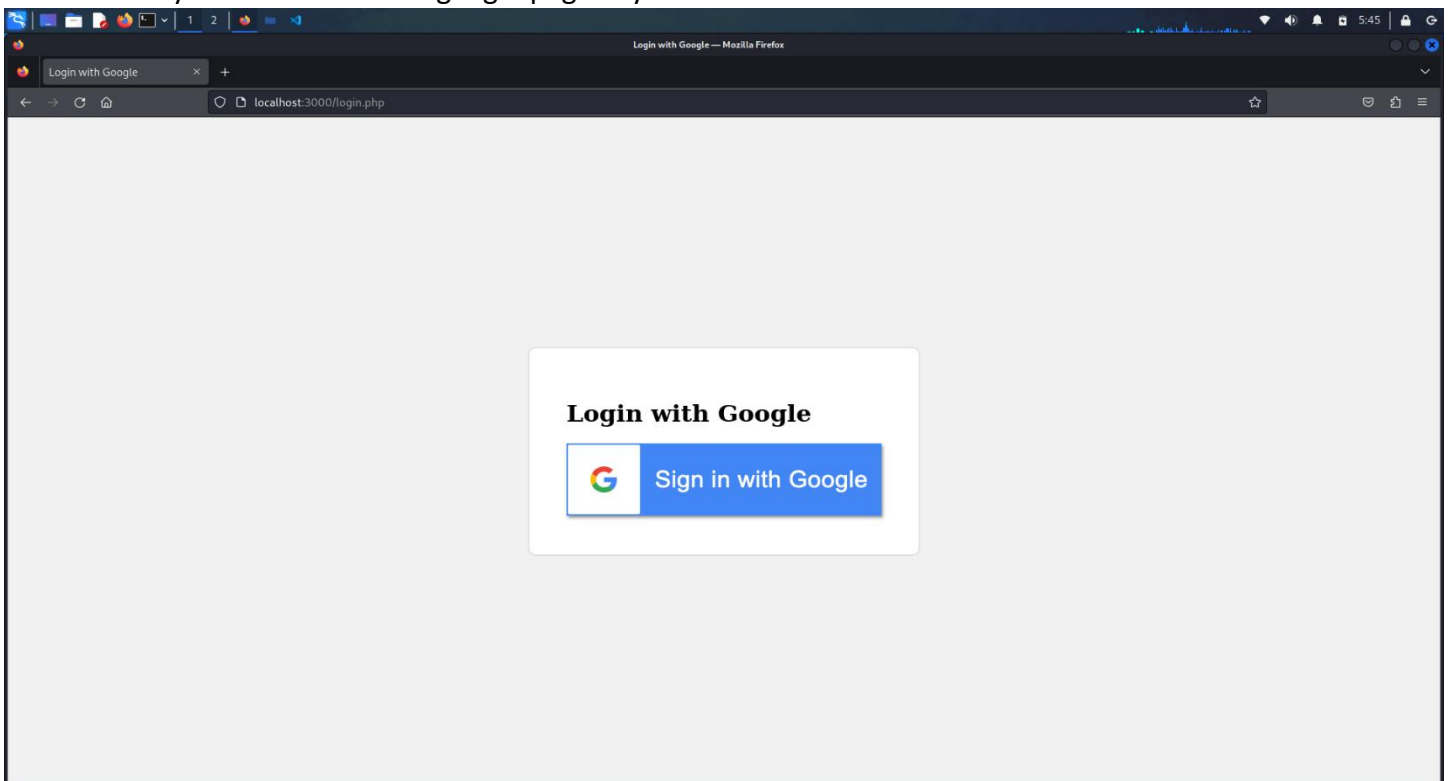


The screenshot shows the Visual Studio Code editor with the 'client\_secret.json' file open. The file contains the following JSON content:

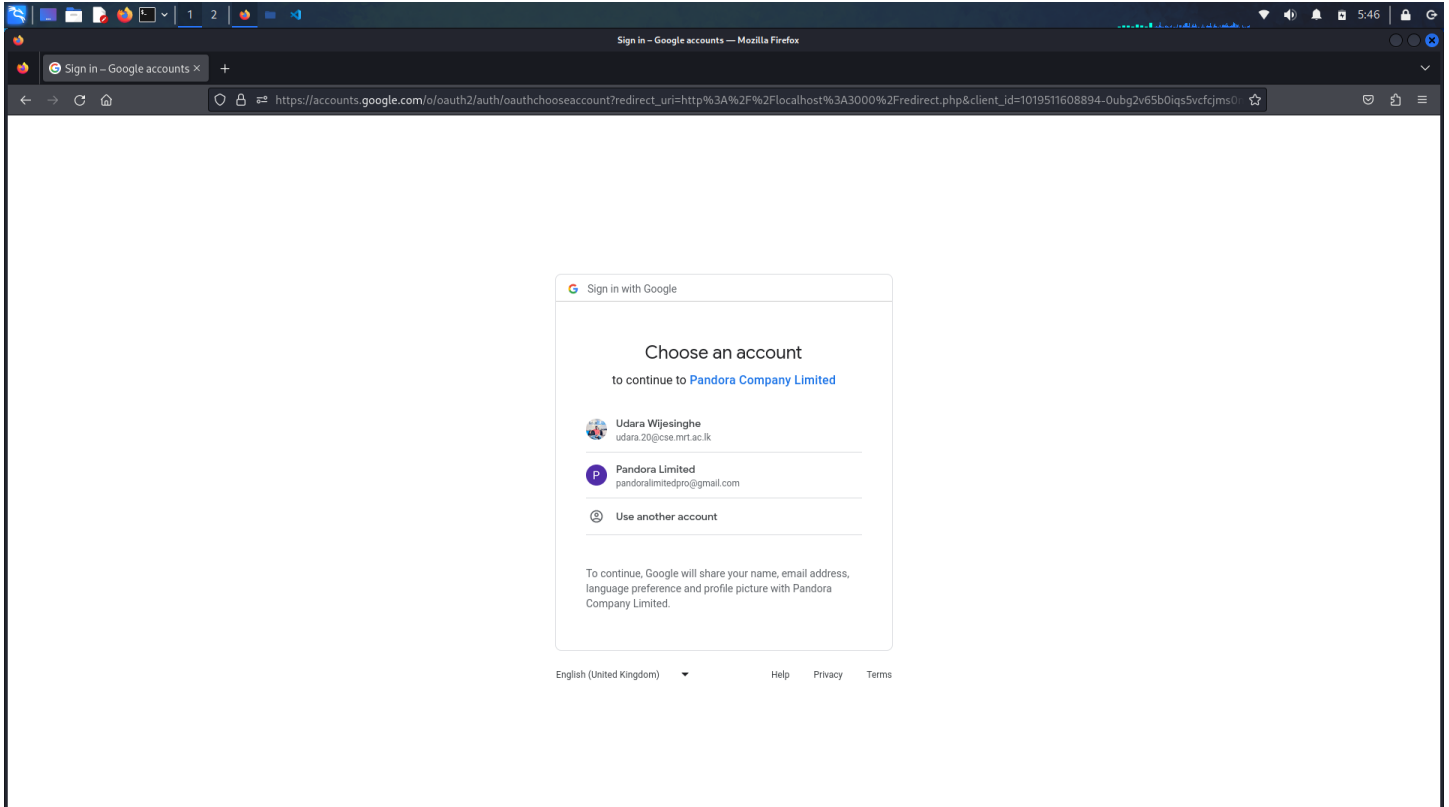
```
{
  "web": {
    "redirect_uri": "http://localhost:3000/login.php",
    "token_uri": "https://accounts.google.com/o/oauth2/token",
    "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
    "client_secret": "YOUR_CLIENT_SECRET"
  }
}
```

## The Demonstration Process

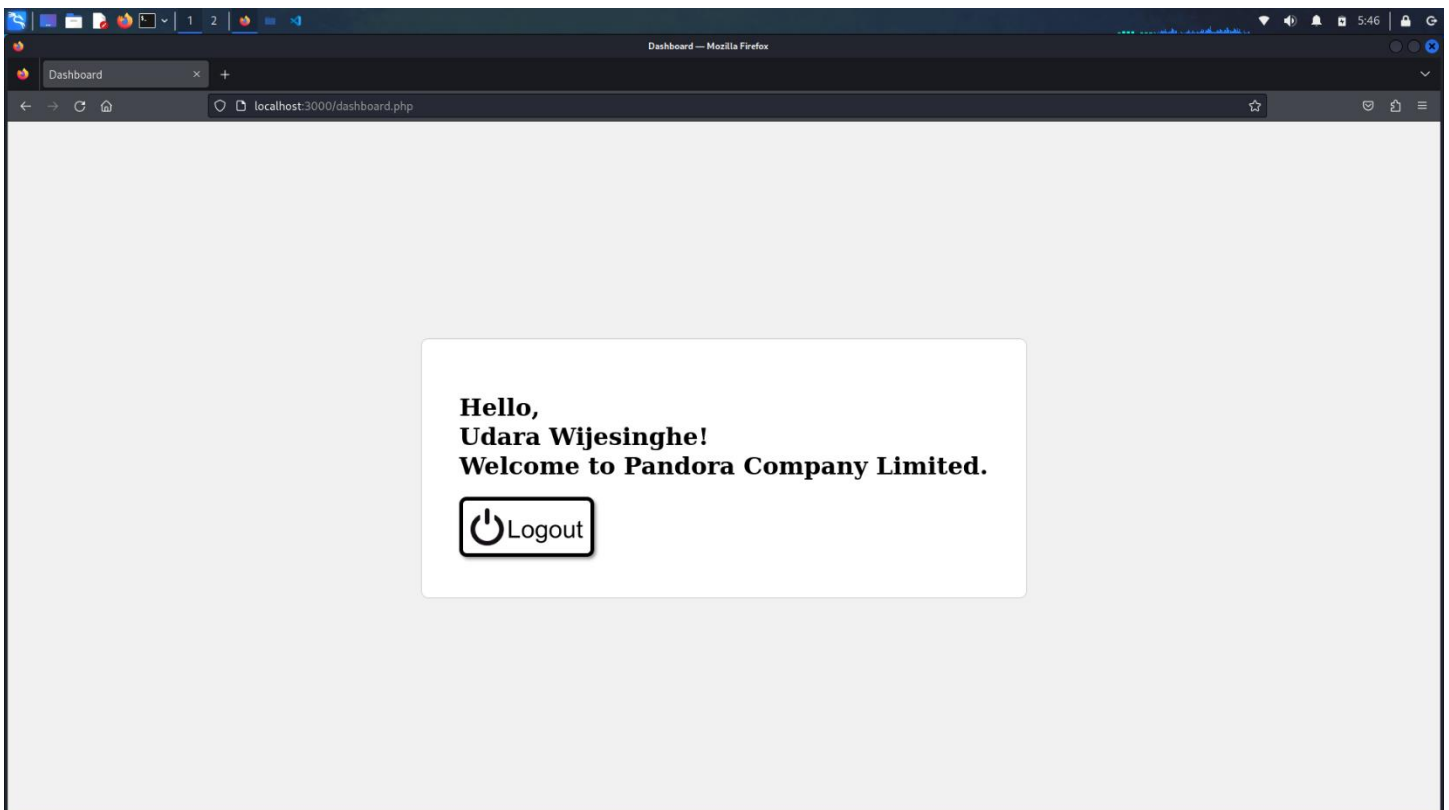
1. First install the **php, curl, and composer** in your computer. The installation process is mentioned above.
2. Then go to the relevant directory that all the files located and give following command to install the wanted dependencies.  
**composer install**
3. Then give the following command to start the php server.  
**php -S localhost:3000 -t .**
4. Then goto your browser and go to following link, <http://localhost:3000/login.php>
5. Then you can see following login page in your browser.



6. Then click **“Sign in with Google”** button.
7. Then you redirect the sign in with google page. Choose your any google account that want login into system

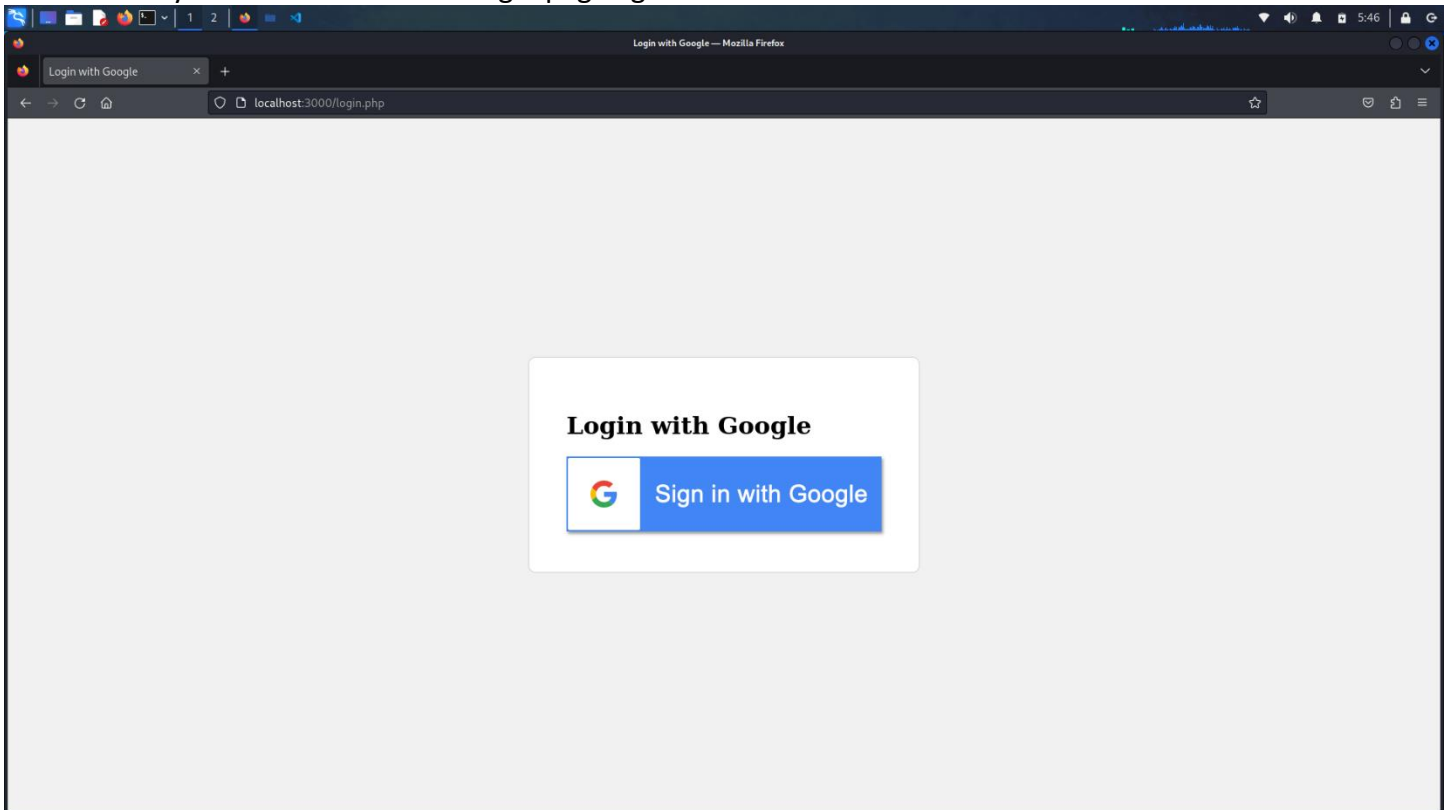


8. Then you can go to the dashboard of the pandora limited company.



9. Then if you want to logout your system then press the logout button in the page.

10. Then you will redirect to the login page again.



### **Challengers that are faced during the process**

- Setting up OAuth2 correctly is challenging, especially since it's my first time working with it. Configuring client IDs, secrets, and callback URLs causes some error-prone.
- Handling user authentication and access tokens securely is crucial. I have to protect user data and tokens from potential security threats.
- Dealing with the redirection flow, especially ensuring that the redirect URIs match, is the source of issues. Incorrect or misconfigured URIs lead to failed logins.
- Safely storing and managing access tokens in applications is very complex. I need to ensure that tokens are not exposed or compromised.
- Maintaining user sessions, especially when it comes to handling login and logout operations, is challenging. Proper session handling is vital for user experience and security.
- Dealing with errors, such as when the user denies access or when the OAuth2 provider returns an error handling is a difficult task.

## **Summary**

The project at hand involves the integration of OAuth2 for user authentication in PHP-based applications, with a specific focus on enabling "Login with Google" functionality. The goal is to create a secure and user-friendly authentication system that centralizes user management for Pandora Company Limited. The project comprises several key components: a login page with a "Login with Google" button, a redirect page to handle Google's OAuth2 authentication, and a user dashboard for displaying personalized content upon successful login. This initiative not only enhances security but also streamlines the user experience, making it convenient for users to access Pandora's services using their Google credentials. The project's steps involve OAuth2 implementation, user dashboard customization, and the management of user sessions. Challenges include correctly configuring OAuth2, handling security considerations, managing user sessions, and ensuring a smooth user experience. In the end, the project aims to provide Pandora Company Limited with a robust, user-friendly, and secure authentication solution, reinforcing the company's commitment to efficient user management and data protection.

## **References**

- [1] I get the support of 'chatgpt' when occur some error in project.
- [2] php-oauth2-example - <https://github.com/XeroAPI/php-oauth2-example/tree/master>