COMP4801 Interim Report

Implementation of HKID/HKU
OAuth 2.0 Provider for Real Identity Verification

Name: Chiew Lik Seng (3035767487)

Supervisor: Dr. Chan, Hubert T. H.

Date of Submission: 21/01/2024

Acknowledgement

I would like to express my gratitude to the supervisor of this project, Dr. Chan, Hubert T. H.

Abstract

This report addresses the evolving landscape of online identity verification and proposes a solution using the OAuth 2.0 framework. The increasing demand for real-name verification in various online services, driven by concerns over security and public order, prompts the need for a more efficient and secure authentication system. The report explores the potential integration of OAuth 2.0 as an authentication platform, emphasizing its role in simplifying and unifying real-name verification processes. The research delves into the OAuth 2.0 protocol, examining its principles, grant types, and implementation in web and mobile applications. Security considerations and best practices, including potential weaknesses and countermeasures, are thoroughly investigated. The literature review focuses on the OAuth 2.0 framework, highlighting its deployment as an authorization framework for secure access to resources. The project's aims involve developing an OAuth 2.0 provider for real identity verification and contributing to the fields of authentication, cybersecurity, and delegated access control. Objectives include understanding the OAuth 2.0 protocol, selecting the most suitable grant type, integrating OAuth 2.0 into applications, and addressing security concerns. The report outlines the products and deliverables of the OAuth HKID provider, including the authorization server, resource server, and the authorization server login page. Technologies such as Node.js, Express.js, React.js, and MongoDB are chosen to implement the backend, frontend, and database components. Amazon Web Services (AWS) is selected as the cloud platform for deployment, ensuring scalability and reliability. The methodology section covers requirement analysis, system design, database design, implementation, testing, and production deployment. The project's current progress is ahead of schedule, with research completed, database schema designed, and the authorization server development underway. The frontend login page and resource server are finished in advance, providing visible results. Risk management involves identifying potential risks such as resource over-utilization, incorrect implementation of OAuth 2.0, and inappropriate database design. Mitigation strategies include using AWS Elastic Load Balancing, careful implementation following documentation, and consulting MongoDB experts. The report concludes by emphasizing the project's objective to deliver a secure and efficient OAuth provider for real identity verification. The team is confident in meeting project deadlines and overcoming upcoming challenges in implementing OAuth 2.0 with Proof Key for Code Exchange (PKCE) for enhanced security.

List of Tables

Table	Page number
Risk Assessment Matrix	12
Non-OHS Risk Assessment	12

List of Figures

Figure	Page number
Abstract Protocol Flow	3
Authorization Code Flow	4
Authorization Code Interception Attack	4
Abstract Authorization Code Flow with PKCE	5
Authorization Endpoint Code Snippet	8
Token Endpoint Code Snippet	9
Screencap of Authorization Server Login Page	10
Resource Server API Endpoint Code Snippet	10
Project Gantt Chart	11

Table of Contents

Acknowledgement i

Abstract i

List of Tables ii

List of Figures ii

1. Introduction 1
2. Aims & Objectives 1
   - 2.1. Research Question 1-2
   - 2.2. Aims 2
   - 2.3. Objectives 2
3. Literature Review 2
   - 3.1. OAuth 2.0 Framework 2-4
   - 3.2. Proof Key for Code Exchange 4-5
4. Products & Deliverables 6
   - 4.1. Authorization Server 6
   - 4.2. Resource Server 6
   - 4.3. Authorization Server Login Page 6
5. Technology Framework 6
   - 5.1. Backend 6
   - 5.2. Frontend 7
   - 5.3. Database 7
   - 5.4. Cloud Platform / Deployment 7
6. Methodology 7
   - 6.1. Requirement Analysis 7
   - 6.2. System Design 7
   - 6.3. Database Design 7
   - 6.4. Implementation 7
   - 6.5. Testing 7-8
   - 6.6. Production Deployment 8
7. Project Progress 8-11
   - 7.1. Preliminary Results and Discussion 8-10
     - 7.1.1. Development of Authorization Server 8-9
     - 7.1.2. Development of Resource Server 9-10

7.1.3. Development of Authorization Server Login Page 10
7.2. Limitations and Future Work 11
7.1.1. Limitations of Approaches 11
7.1.2. Future Work 11
8. Scope, Project Plan & Timeline 11
8.1. Scope 11
8.2. Project Plan & Timeline 11
9. Risk Management Plan 12
9.1. Risk Assessment Matrix 12
9.2. Non-OHS Risk Assessment 12
10. Conclusion 13
11. References 14

1 Introduction

In this era of globalisation and rapid technology advances, Internet is accessible to most of the world's total population [1]. The widespread proliferation of internet access throughout many societies worldwide has facilitated not only socially positive practices, but also detrimental ones. There are countless users on the web that remain their anonymity and participate in online discussions without fear of reprisal in the physical world. It is just the nature of Internet. But in recent years, the norm has gradually been broken. It can be observed that more and more applications or services require real name verification before we can access the service for instance the registration process of student Octopus card, booking process of community sports centre, etc. Besides that, countries like China even imposed the real-name registration rules to stabilize public order [2]. The Republic of Korea, the nation with the highest rate of high-speed internet access in the world, created its own "Real Name Verification System" as well by forcing Korean citizens to verify their identities on a public institution website [3].

In Hong Kong, users are generally required to scan their HKID (Hong Kong Identity Card) by their phone camera in these real identity verification procedures. It is annoying that the process is not essentially accurate in times since factors like lighting, shadow and camera resolution could affect the outcome of the verification process. Other than that, the same process is repeated whenever real name verification is required on any other web services. Thus, some users just quit attempting to use certain services.

To better facilitate the real identity verification process, OAuth framework has come across my mind as an auth account basically represents a user's identity in the scope of client application and HKID is no different from an auth account in real life. Each citizen would have his/her own user account on a resource server regulated by a trusted authority which would be the government and all the personal information regarding the account owner is contained within the resource server. Whenever users log into any third part web applications via OAuth 2.0 protocol, by providing the correct login credentials to the government server, an acknowledgement would be returned by the server and it could serve the same purpose as the HKID and extend to even more possibilities. The real-name verification processes could also be simplified and unified by one authentication service platform.

2 Aims & Objectives

In this section, the questions that are to be kept in mind during the research process are covered in subsection 2.1 Research Question. As for the aims and objectives of project, they are being further discussed in subsection 2.2 Aims and subsection 2.3 Objectives accordingly.

2.1 Research Question

The research questions for this project are as follows:

1. What is OAuth 2.0 protocol and how does it work in principle?
2. What are the differences between different OAuth 2.0 grant type and which one would be the best choice for our situation?
3. How can OAuth 2.0 protocol be incorporated into web and mobile applications to act as the authentication system?
4. What are the common security weaknesses in implementation of OAuth 2.0 server and what are the countermeasures?

2.2 Aims

This project aims to develop an OAuth 2.0 provider for real identity verification and investigate the possible extension and contribute to the advancement of knowledge in the field of authentication, cybersecurity, and delegated access control.

2.3 Objectives

The objectives of this project are:

1. To understand the OAuth 2.0 protocol and its underlying principles.
2. To study different OAuth 2.0 grant type (e.g. Authorization Code Flow, Implicit Flow, Resource Owner Password Credentials Flow, and Client Credentials Flow) and pick the most suitable one for the project.
3. To explore the integration of OAuth 2.0 with existing web and mobile applications by incorporating SSO capabilities and delegated access.
4. To inspect the security considerations and best practices in OAuth 2.0, for instance secure token handling, client authentication and protection against common attacks.

3 Literature Review

As the authentication system is built on top of a mature, secure authorization framework, this project relies heavily on analysis and implementation of the OAuth 2.0 Framework. Thus, a brief introduction of the OAuth protocol is discussed in subsection 3.1 OAuth 2.0 Framework along with a more detailed approach of OAuth implementation being discussed in subsection 3.2 Proof Key for Code Exchange.

3.1 OAuth 2.0 Framework

Open Authorization (OAuth) is a widely deployed authorization framework that allows users (known as “resource owner”) to grant limited access to their resources on one website (known as the “resource server”) to another website or application (known as the “client”) without sharing their credentials [4]. In this way, users do not need to create an auth account for every single new client application. The risk of leaking users’ login credentials is also minimized as OAuth relies on short-lived access tokens and third-party client application will only be communicating with the resource server via an access token. In general, there will be not less than 2 endpoints in the authorization server. The two mandatory endpoints are authorization endpoint

and token endpoint. The authorization endpoint communicates with the resource owner and returns an authorization grant/code whereas the token endpoint interacts with the client application where client presents the authorization grant to exchange for the access token. The scope of resource access is embedded onto the access token which means third-party client application has restricted access to the resource data of the resource owner.

The basic OAuth protocol flow is illustrated in Fig.1 below.

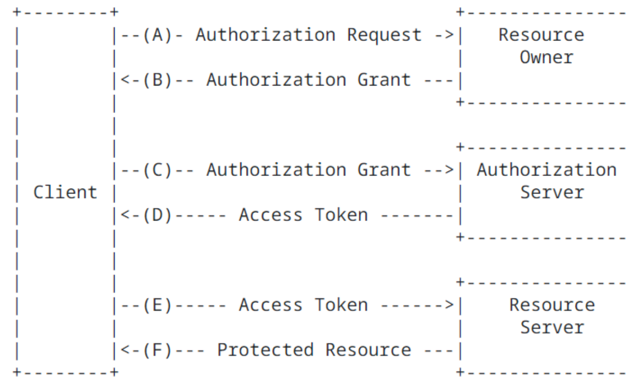

Fig. 1: Abstract Protocol Flow. [4]

OAuth 2.0 defines four main authorization grant types: authorization code, implicit, resource owner password credentials, and client credentials. As for now, authorization code flow would be our choice of authorization grant as it provides an additional layer of security compared to other grant types. It involves an extra step of exchanging authorization code between the client and the authorization server which better enhance the security of the process [4]. Besides that, the user is redirected to the authorization server's login page as they are providing their login credentials. This makes sure that users' login credentials are never exposed to the client application and in turn reassure users and earn their trusts to select the OAuth login method.

The Authorization Code Flow (see Fig. 2 below) below includes the following steps:

(A) The flow starts with the client directing the resource owner' user-agent (in this case the browser) to the authorization endpoint in authorization server. The client identifier, requested scope, local state and a redirection URI is sent along.

(B) The resource owner is authenticated by the authorization server via the browser and resource owner is asked for its permission to grant the client's access request.

(C) Once access is granted by the owner, the user-agent is redirected back to client following the redirection URI mentioned earlier and the authorization code is included in the URI.

(D) The client then makes a request for the access token by presenting the appropriate authorization code and the redirection URI to the authorization server.

(E) The authorization server authenticates the client by validating the authorization code and verifying that the redirection URI is the one used to exchange for authorization code in the previous steps. The authorization server returns an access token if the authentication is valid.

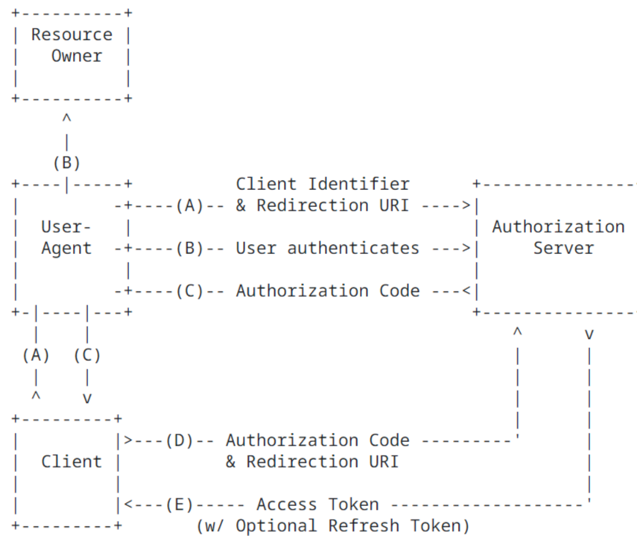

Fig. 2. Authorization Code Flow. [4]

The communication route to the authorization server is built upon Transport Layer Security (TLS) hence it greatly enhances the security of the framework.

3.2 Proof Key for Code Exchange

As illustrated in the previous section, the correct implementation of Authorization Code Flow is sufficiently secure, but there is still a common security weakness for OAuth 2.0 public clients signing in with Authorization Code Grant such that it is prone to authorization code interception attack [5].

An illustration of a possible cyberattack, specifically called "Authorization Code Interception Attack" is given in Fig. 3.

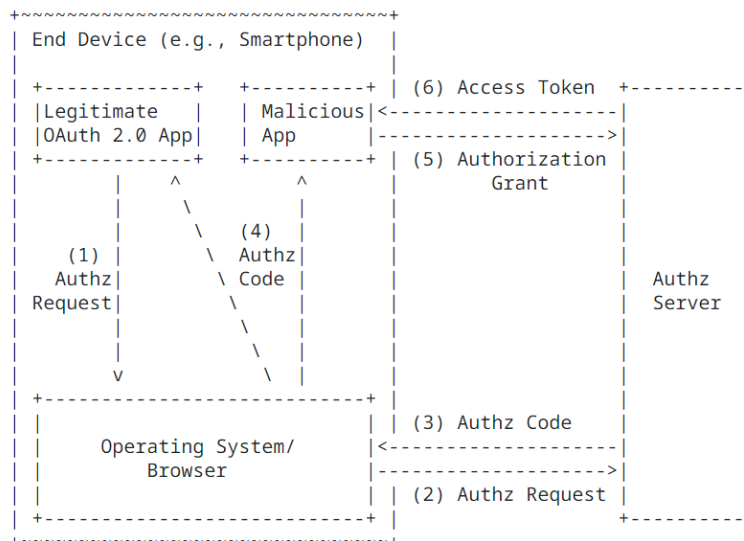

Fig. 3. Authorization Code Interception Attack. [5]

If a malicious application is planted on the client device, when the authorization code is sent back to the client application via the browser in step (4), the malicious application could intercept the authorization code and in turn request and obtain an access token by going through steps (5) and (6) respectively. As the communication path in steps (1) and (2) is directed to the authorization server, if OAuth Protocol is implemented correctly with TLS version, the transport channel is secure from any attempts to intercept [6].

To counter against this kind of attack, an extension to the Authorization Code flow called Proof Key for Code Exchange (PKCE), pronounced as 'pixey' is utilised. The abstract flow utilizing PKCE practice is visualized in Fig.4.

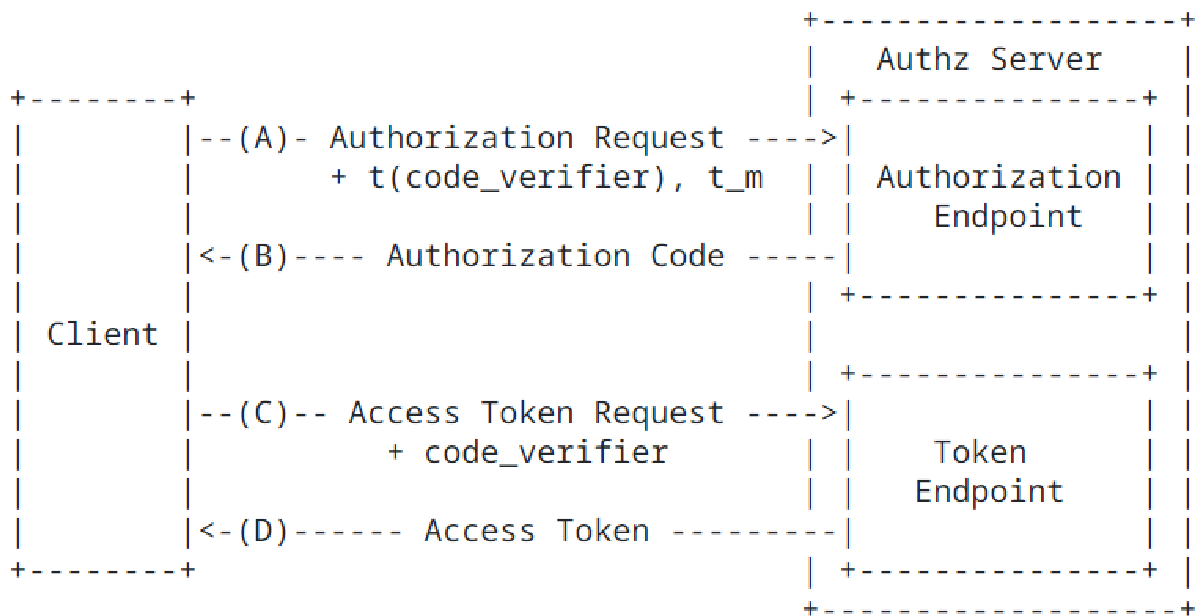

Fig. 4. Abstract Authorization Code Flow with PKCE. [5]

PKCE basically creates a randomly-generated cryptographical key called "code verifier" for every authorization request and then the code verifier is transformed into a value called "code challenge" [5]. The "code challenge" derived from the "code verifier" is then sent to the authorization endpoint of the authorization server along with the request for the authorization code. The authorization code is then sent to the token endpoint along with the "code verifier". Afterwards, the authorization server compares the "code verifier" and "code challenge" to verify that the client requesting for access token is the same as the one asking for the authorization code. This could effectively mitigate the interception attack as the attacker would not know this one-time key as it is sent over TLS.

4 Products & Deliverables

There are mainly three products that build up the OAuth HKID provider. The authorization server that handles the authentication logic is discussed in subsection 4.1 whereas the resource server that stores resource data and responds to clients' data retrieval requests is explained in detail in subsection 4.2 Resource Server. As for the visible user interface component in the project, it will be further discussed in subsection 4.3 Authorization Server Login Page.

4.1 Authorization Server

The Authorization Server is the main component of this project as it contains the two authorization endpoints, authorization endpoint and token endpoint, to handle all the authentication processes. It will be a backend server that can handle RESTful APIs in the form of HTTP request and return appropriate HTTP responses.

4.2 Resource Server

This original goal of this project is to have the resource server containing personal information of all Hong Kong citizens but it is impossible to obtain access these data from government at the moment hence an alternative "HKU students" resource server is created to replace the "HKID" resource server as they are interchangeable components. The resource server allows retrieval of data upon receiving a valid access token according to the scope and permissions associated with the access token by setting up routes to handle HTTP requests.

4.3 Authorization Server Login Page

A login page is written for the ease of client applications developers to incorporate the OAuth provider into their applications. It serves as an interface for users to bypass the third-party client application and interact with our authorization server directly. The login page fires authentication requests to the authorization server upon receiving login credentials from the user.

5 Technology Framework

The content below covers the technology framework and tools chosen for various parts of the project. The detailed reasoning of the backend framework is discussed in subsection 5.1 while the frontend development framework is discussed in subsection 5.2. The database that are obligated to store the resource data is covered in subsection 5.3 Database. Lastly, the cloud platform that is chosen for deploying and hosting our webpages and web servers is further discussed in subsection 5.4 Cloud Platform/ Deployment.

5.1 Backend

The two servers will be implemented in Node.js/Express.js Framework as Node.js is asynchronous and non-blocking in nature. This allows the server to handle many concurrent connections and requests efficiently. In OAuth, it is common that multiple clients are making simultaneous authorization requests and Node.js performs well in these situations, making our OAuth project scalable and responsive.

5.2 Frontend

As for the authorization server login page, React.js will be used to create the webpages because it is also a JavaScript framework and thus it works well with the backend servers. Plus, React.js supports dynamic rendering of webpage, making the user interface more interactive and responsive.

5.3 Database

Our choice of database would be MongoDB because it is designed to scale horizontally, allowing developers to distribute the OAuth server across multiple machines to handle high traffic loads. As we are expecting to have every Hong Kong citizen as our users, the fact that MongoDB allows database to scale horizontally makes it a decent choice to handle millions of database connection requests well at a time.

5.4 Cloud Platform / Deployment

AWS (Amazon Web Services) is best known for its scalability and elasticity so it very suitable for us to easily scale our OAuth server based on demand. It is also easy to deploy Node.js servers on the AWS EC2 instances. AWS also offers a highly reliable infrastructure with multiple availability zones and redundant components that minimize the downtime and provides high availability for our authorization service.

6 Methodology

This section covers the main steps of building out the OAuth provider including subsection 6.1 Requirement Analysis, subsection 6.2 System Design, subsection 6.3 Database Design, subsection 6.4 Implementations, subsection 6.5 Testing and subsection 6.6 Production Deployment.

6.1 Requirement Analysis

Firstly, a comprehensive analysis of the requirements for the OAuth server must be conducted. The core functionalities, security measures and the OAuth workflow are first identified.

6.2 System Design

Based on the analysis, system design is then carried out to define the architecture, components, and interactions of the OAuth server. It includes deciding the tech stack, selecting appropriate frameworks or libraries.

6.3 Database Design

Database and tables are to be designed and tailored for the OAuth system to efficiently store access tokens, authorization grant, and users' information. Good database design ensures the scalability of the application when the userbase grows to certain extent.

6.4 Implementation

This phase involved coding out the OAuth server using Node.js, importing the chosen frameworks and libraries. The OAuth protocols are implemented by accurately following the documentation, ensuring compliance with OAuth 2.0 and related standards such as PKCE and TSL.

6.5 Testing

Unit testing is conducted simultaneously during development process to ensure the individual component work as expected in the early stages. Integration testing is carried out at the final stage of development to ensure each functional component interact correctly with each other.

6.6 Production Deployment

This phase involved deploying the OAuth server on the Amazon Web Services (AWS) cloud platform, deploying a production MongoDB database, and utilizing suitable DevOps tools to manage the project.

7 Project Progress

7.1 Preliminary Results and Discussion

7.1.1 Development of Authorization Server

As discussed in section 4, Authorization Server is undoubtedly the most crucial component of the OAuth provider as it handles the authentication and authorization logic of the provider. The development progress of the Authorization Server has been closely monitored and focused during semester 1. The Authorization Server consists of two significant API endpoints including the Authorization Endpoint which returns the authorization code and the Token Endpoint which returns the access token to the client application. Up until now, the basic structure and logic of the two API endpoints have been completed as shown in Figure 5 and Figure 6 and a typical OAuth 2.0 protocol workflow could be successfully functioning right now.

```
21  router
22    .route("/code")
23    .get(projectMiddleware, verifyAuthToken, async function (req, res) {
24      try {
25        var code = await req.user.generateOAuthCode(req.project);
26        redirectURL = `${req.query.redirectURL}?code=${code}`;
27        return res.send({ redirectURL });
28      } catch (e) {
29        console.log(e);
30        res.status(500).send({ message: "Unknown Error", code: 500 });
31      }
32    });
```

Fig. 5. Authorization Endpoint Code Snippet

```
33  router
34    .route("/token")
35    .get(projectMiddleware, verifyOAuthCode, async function (req, res) {
36      if (req.project.projectSecret != req.query.projectSecret) {
37        return res
38          .status(400)
39          .send({ code: 400, message: "Mismatch ProjectID and Secret" });
40      }
41      user = req.user;
42
43      user
44        .generateAccessToken(req.decoded.scope)
45        .then((token) => {
46          return user.removeToken(req.token).then((e) => {
47            return token;
48          });
49        })
50        .then((token) => {
51          res.send({ access_token: token });
52        })
53        .catch((e) => {
54          res
55            .status(400)
56            .send({ message: "Error while generating access token" });
57        });
58    });
```

Fig. 6. Token Endpoint Code Snippet

The remaining parts to be added on to the Authorization Server would be the implementation of the logic of PKCE. It requires the Authorization Endpoint to receive the code challenge (the hashed code verifier) and store it inside the server. On the other hand, the Token Endpoint is required to receive the code verifier and compare it with the code challenge previously stored. Hashing algorithm is essential in this case as the security of the code challenge relies on the cryptographical strength of the hashing algorithm.

7.1.2 Development of Resource Server

As for the resource server, upon critical assessment of the existing design, the resource server logic is to be included in the server machine where the authorization server logic is located instead of deploying it on a separate cloud machine. This is to reduce the complexity of server and database deployment as one server suffices to meet the requirements of our requirements. Essentially, the resource server has to allow for retrieval of users' resources according to the scope embedded in the access token upon presented a valid access token. Hence only a single API endpoint that dynamically reacts to different scope of the access token and an authentication middleware that validates the access token are necessary. Figure 7 displays the complete development of the resource server. Here I utilized a dictionary to map between the scope and the array of data fields that the token is allowed to access. By iterating through the array, we would be able to return the list of values that correspond to the data field.

```
const scopeMapping = {
  full: ["_id", "name", "email", "phone"],
  default: ["_id", "name"],
  email: ["_id", "name", "email"],
  phone: ["_id", "name", "phone"],
};

router
  .route("/userinfo")
  .get(verifyAccessToken, async function (req, res) {
    token = req.decoded;
    user = req.user;
    res.send(R.pick(scopeMapping[token.scope], user));
  });

module.exports = router;
```

Fig. 7. Resource Server API Endpoint Code Snippet

7.1.3 Development of Authorization Server Login Page

The static layout of login page is finished building as well as show in Figure 8. It comprises of two text fields to let users input their credentials (e.g. email and password) and a button that triggers communication with the authorization server once it is clicked.

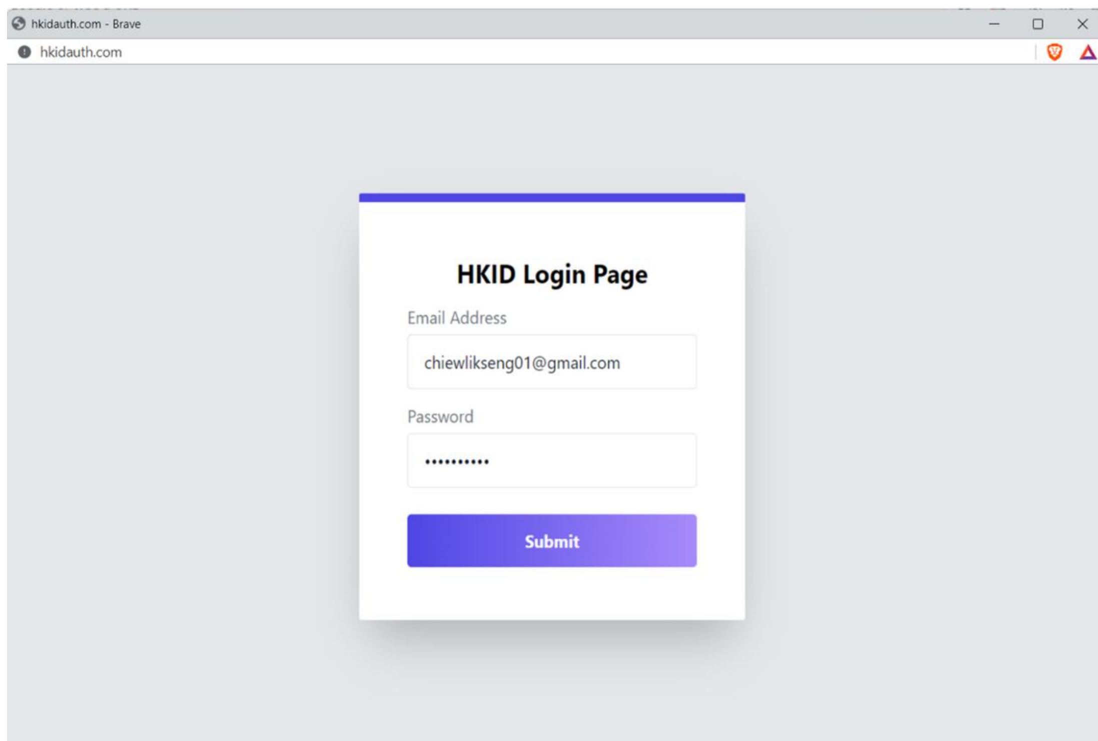

Fig. 8. Screencap of Authorization Server Login Page

7.2 Limitations and Future Work

7.2.1 Limitations of Approaches

As per mentioned by Dr. Hubert Chan during the first presentation, there exists a potential risk of token leakage as the PKCE protocol only protects the system from authorization code interception attack. If an attacker manages to compromise the client application or the user's device, they might still gain access to the access token. Therefore, it is essential to implement other security measures, such as secure token storage and token expiration policies to mitigate the impact of token leakage.

Besides that, OAuth 2.0 with PKCE requires client applications to support the PKCE extension. While most modern client libraries and frameworks have built-in support for PKCE, older or custom-built clients may not. This can limit the compatibility of our OAuth provider with certain client applications.

7.2.2 Future Work

The first and most significant task would be the implementation and addition of PKCE logic into the authorization server API endpoints. Once it is done, all individual components in the OAuth provider are built and set for the integration testing. The progress is then followed by stress testing and user acceptance testing. Considerations of possible security measures to improve the security of the authentication system will be taken. Overall, 90% of the project has been done and we are greatly ahead of schedule.

8 Project Schedule & Current Progress

8.1 Project Scope

The scope of this project includes:

- Development of an OAuth 2.0 with PKCE server
- Development of a resource server that stores the personal information of the resource owner

The scope of this project does not include:

- Client registration process with the authorization server

8.2 Project Plan & Timeline

A Gannt Chart is produced to ensure that the project progresses smoothly and can be monitored easily. The tasks allocated as well as the timelines are defined in the Gantt Chart show in Figure 9.

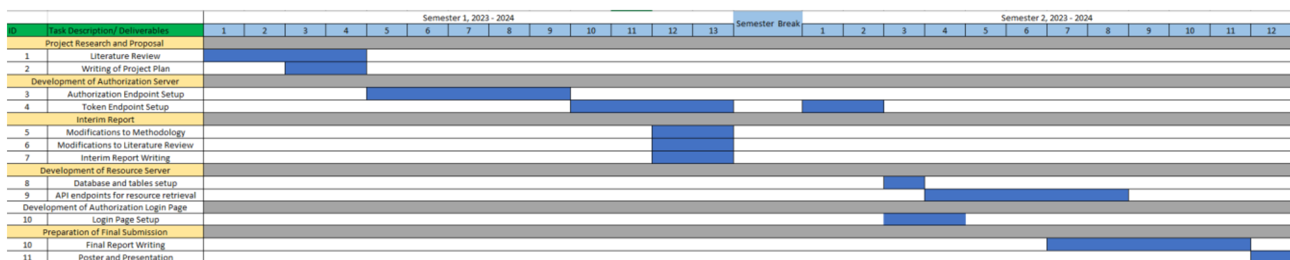

Fig. 9. Project Gantt Chart that displays the timeline of the project in a simple and more readable format

9 Risk Management Plan

Table 1 from subsection 8.1 and Table 2 from subsection 8.2 below combine to provide an intuitive way for audience to understand the possible underlying risks and consequences of the subsequent risks.

9.1 Risk Assessment Matrix

Consequence and Likelihood	Insignificant	Minor	Serious	Disastrous	Catastrophic
Rare	L	L	L	L	M
Unlikely	L	L	M	M	S
Possible	L	M	M	S	H
Likely	L	M	S	H	E
Almost Certain	M	S	H	E	E

Table 1: Risk Assessment Matrix which provides a scale to assess the risk of the project based on the likelihood and the serious level of the risks

L = Low Risk, M = Medium Rish, S = Substantial Risk, H = High Risk, E = Extreme Risk

9.2 Non-OHS Risk Assessment

Project Risk	Risk	Likelihood	Consequence	Risk Level	Mitigation	Residual Risk
Resource over-utilization in AWS EC2 instance	Users fail to be authenticated	Possible	Serious	M	Utilize Elastic Load Balancing (ELB) Service to allow flexible adaptions to intensive CPU usage	Lost users' trust in our OAuth application
Incorrect implementation of OAuth 2.0 protocol	Weak security, susceptible to malicious attack	Possible	Disastrous	S	Careful implementation following research documentation and thorough integration testing	Users' data leaked and reputation harm
Inappropriate database design	Non-scalable when userbase grows, server operations become irresponsive	Possible	Disastrous	S	Consulting MongoDB experts to better model the tables	All existing database structures are required to be erased

Table 2: Non-OHS Risk Assessment which lists out the potential project risks, its corresponding risk level, and the mitigation techniques

10 Conclusion

This project aims to deliver a secure and convenient OAuth provider by correct implementation of the OAuth 2.0 protocol from scratch to simplify the real identity verification process as the current method of doing the verification by capturing photo of the HKID card using the phone camera is too inefficient. To do so, we first study the OAuth 2.0 protocol and its underlying principles. Different approaches to implement the OAuth workflow are analysed to find out the best practice judging from the security considerations, overall performance, and implementation complexity. The OAuth provider consists of three main deliverables including a frontend login page, an authorization server, and a resource server. Modern and bleeding edges technology frameworks such as React.js and Node.js are utilized in our project to cater to our project requirements. By far, the frontend login page and the resource server are completed in ahead of schedule. We are in the final step of developing the authorization server by embedding the PKCE logic into the authentication and authorization workflow. Greater challenges are lying ahead during the phase of implementing the details of OAuth 2.0 with PKCE protocol as PKCE brings more complexity to the project but higher complexity brings better security to our project. More possibilities of security measures are to be explored to reduce the risk of illegal and malicious access to users' personal data so as to encourage users and developers to leverage our OAuth provider as a trusted and reliable solution.

The project progress is currently ahead of schedule and it can be confidently said that the complete project can be delivered on time.

11 References

[1] “Digital Around the World”, https://datareportal.com/global-digital-overview

[2] J. Lee, C. Liu, “Real-Name Registration Rules and the Fading Digital Anonymity in China,” Washington International Law Journal, Vol 25, No. 1, 2016.
https://digitalcommons.law.uw.edu/cgi/viewcontent.cgi?article=1717&context=wilj

[3] L. John, “Identifying the Problem: Korea’s Initial Experience with Mandatory Real Name Verification on Internet Portals,” Journal of Korean Law, Vol. 9, 83-108, December 2009. https://s-space.snu.ac.kr/bitstream/10371/85159/1/4.%20Identifying%20the%20Problem%20Korea%E2%80%99s%20Initial%20Experience%20with%20Mandatory%20Real%20Name%20Verification%20on%20Internet%20Portals.pdf

[4] D. Hardt, “RFC 6749 The OAuth 2.0 Authorization Framework,” Internet Engineering Task Force (IETF), October. https://datatracker.ietf.org/doc/html/rfc6749

[5] N. Sakimura, J.Bradley, and N. Agarwal, “RFC 6749 The OAuth 2.0 Authorization Framework,” Internet Engineering Task Force (IETF), October. https://datatracker.ietf.org/doc/html/rfc7636

[6] T. Lodderstedt, M. McGloin, P. Hunt, “OAuth 2.0 Threat Model and Security Considerations”, RFC 6819, DOI 10.17487/RFC6819, January 2013, http://www.rfc-editor.org/info/rfc6819.