# Final Report

## Introduction

In this report you will find explanations on the micro service, explanations on the design and how the project was planned out, I will outline the legal, social, ethical, and professional issues of the project. Finally, I will discuss the implementation and evaluate the effectiveness of the final product.

[Github link](https://github.com/Zedster82/Comp-2001)

[Swagger API Link](https://web.socem.plymouth.ac.uk/COMP2001/OMaynard/Swagger/index.html)

## Background

The micro service that I have implemented is the profile service, it is used use CRUD procedures on the profiles of the user, while also managing things such as followers and favourite activities.

This will all be done via the use of a Swagger interface to show the structure of the API.

I have controllers for: Activities, ArchiveUsers, FavouriteActivities, Follow, FollowersCount, FollowingCount, Login, Logout and finally Users.

Each of these controllers defines the CRUD interactions that can be done in the database and the authentication API.

## Design

A white paper with black text

Description automatically generatedLogical ERD:

A diagram of a software company

Description automatically generated with medium confidence

UML Diagram of the controllers and how they interact with the tables.

## Legal, Social, Ethical and Professional (LSEP)

My implementation makes sure to implement information security, privacy, integrity and preservation. The way that it does this is via encryption, and the permissions that are given to users to help protect unauthorised access to sensitive information.

Encryption helps with information security by ensuring that sensitive data remains confidential and secure from unauthorized access.

It helps with privacy in that it provides users with control over their data as it is well safeguarded and the user doesn’t have to trust other people with their data as much.

It helps with integrity as it helps to prevent unauthorised modifications of data.

It helps with preservation as it is less sought after and is less likely to be modified via unauthorised modifications.

The method that I used to implement the encryption is by double-hashing the passwords that the user enters, along with a randomly generated salt, the hashed password and salt are then stored in the database instead of a plain text password. When the user logs in, the hashed password and salt are retrieved, using the new password with the salt, and checking if the result is the same. This means that the user has entered the correct password.

My data items are designed in a way that enforces integrity, privacy and security as the controllers have limited access to sensitive information and is designed in a way that maintains data integrity via good use of normalisation and best practices in database management.

## Implementation

### General Information

All of the actions in the API check if the user is logged in first, otherwise they are not allowed to perform the action.

All of these also have links to the code that it runs.

### Users

GET – Get all users, returns the main view on the database with followers and favourite activities shown

GET{id} – Gets a specific user from main view

POST – Creates a user with the inputs provided, can be used without logging in

PUT – Edits the current logged in user, only admin can set usertype to admin

DELETE – Deletes (Archives) a user, admin only

### Login/Logout

GET – Logs in the user using the auth api and the database with the passwords

GET – Logout, clears all login data

### Archive Users

GET – Get all archive users, admin only

GET{id} – Get a specific archived user, admin only

### Follower/Counts

POST – Follow user

DELETE – Unfollow user

GET – Get Following count of a user

GET – Get Follower count of a user

### Activities

GET – Get a list of all activities

GET{id} – Get a specific activity

POST – Create an activity, admin only

PUT – Edit activity, admin only

DELETE – Delete activity, admin only

### FavouriteActivities

GET – Get a list of all users and their favourite activities

GET{id} – Gets a list of all favourite activities from a specific user.

POST{id} – Favourites an activity

DELETE{id} – Unfavourite an activity

## Evaluation

An improvement that could be made is that instead of using double hashing to help remove clusters and reduce collisions I could have instead used uniform probing which is asymptotically equivalent to double hashing. [[1]](#_References) This would have an improvement that means the computational requirements would be a lot less.

Another thing that would be a massive improvement to the project would be to implement good documentation in the API, this would help people that want to use the endpoints be able to understand the data types and how to interact with the API a lot better.

I also could have created a global function to contact the database instead of using repeating code. However, this is not that important as different functions require different handling of the data and therefore the code is not as repetitive as it seems.

Finally, I could heavily improve on fault tolerance, the reason for this is that my program doesn’t handle errors in a perfect manor and could be improved to be clearer in what went wrong.

### Testing

#### Login

A screenshot of a computer

Description automatically generated

#### Logout

#### Get All Users

A screenshot of a computer

Description automatically generated

#### Get Specific User

A screenshot of a computer

Description automatically generated

#### Post User (Create)

A screenshot of a computer

Description automatically generated

New user is now present in database: A screenshot of a computer

Description automatically generated

#### Edit User

Before editing:

A screenshot of a computer

Description automatically generated

Editing Data:

A screenshot of a computer

Description automatically generated

After editing:

A screenshot of a computer

Description automatically generated

This is the data stored in the database and as you can see the data has been changed to be the new data as pat is now an admin with the new email and encrypted password.

#### Delete User (Archive User)

Logged in as grace we are going to delete pat now.

A screenshot of a computer

Description automatically generated

As you can see pat is now deleted from the main users tableA screenshot of a computer

Description automatically generated

And is present in the archive tableA screenshot of a computer

Description automatically generated

#### Follow

A screenshot of a computer

Description automatically generated

Before following a user with login of grace

A screenshot of a computer

Description automatically generated

After executing with id 2 and 3:

A screenshot of a computer

Description automatically generated

#### Unfollow

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## References

### 1.

*Leo J. Guibas and Endre Szemeredi. 1976. The analysis of double hashing(Extended Abstract). In Proceedings of the eighth annual ACM symposium on Theory of computing (STOC '76). Association for Computing Machinery, New York, NY, USA, 187–191. https://doi.org/10.1145/800113.803647*