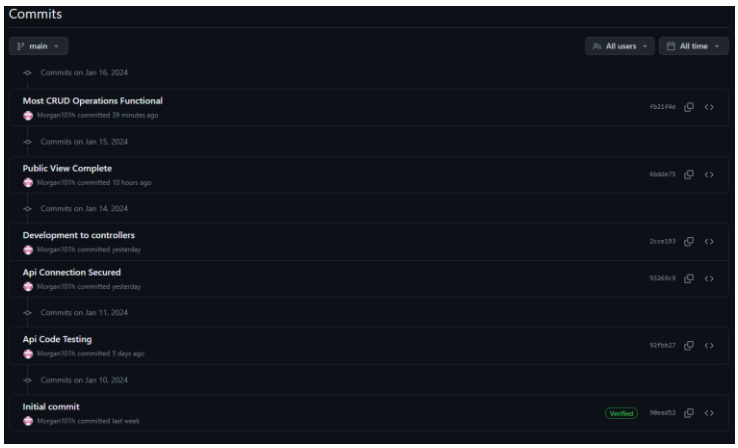
Morgan Hodge

Introduction

GitHub Repository: https://github.com/Morgan101h/Comp2001_70

Micro Service: Swagger UI (plymouth.ac.uk)

This report will document the creation of the micro service I implemented. The GitHub repository can be accessed at the top of the report, this will contain the entirety of the project and a READ.ME file. The Git Repository records the development of the Visual Studio code as it is pushed, providing readers with a timestamp and a brief summary of information on the modifications made with each commit.



This micro service had to use the existing authenticator API that was provided by the client, store data on a Microsoft SQL Server, and be written in C# and ASP.NET. In this report I will discuss how I addressed issues around Legal, Social, Ethical and Professional(LESP) factors. I will discuss the security aspects of the design and provide a variety of UML diagrams and logical ERDs to illustrate the intricate design process.

Background

The implemented microservice was for a 'ProfileService' that had to perform several operations including: Create, Read, Update, Delete (CRUD) operations on profiles, make a view that allows all users to see every profile (but just a restricted view of the stored data), Permit users to modify their own profiles (but only those users may do so), and allow only admins to archive(Delete) profiles.

The Profile service will be used as a backend API for a well-being trail application. This app allows users to create profiles, view trails, edit profiles and perform more functionalities. When users create an account various data is collected such as name, email, password, etc, and the data is stored on a Microsoft SQL server database hosted on dist-6-505.uopnet.plymouth.ac.uk.

The microservice was required to provide a clear endpoint interface represented via swagger, and an output of data in JSON format.

Design

Architecture of the microservice:

The Profile service acts as a backend interface between the SQL Database and RESTful API. Once a user uploads data to the trails app, the data gets stored in the SQL Database. We have designed the microservice to allow for interaction between services, users can perform CRUD operations on their profiles within the API.

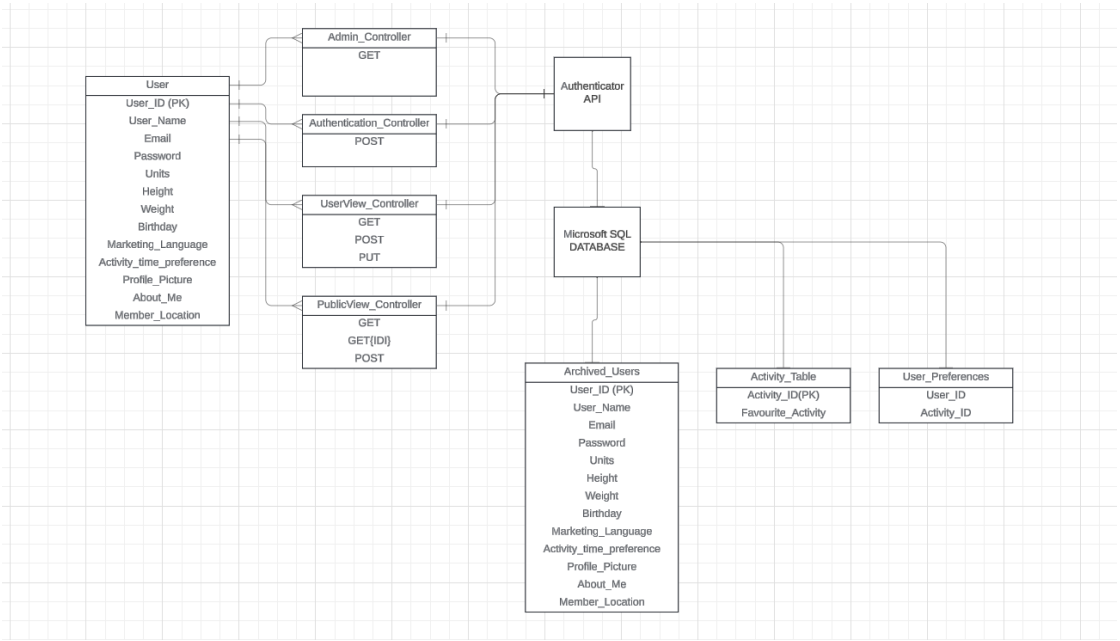
The microservice implemented:

- CRUD operation on profiles
- Anyone can view all profiles but has limited view
- Profiles are owned by a user and can only be edited by the user who created them
- Only an admin can remove a profile, but it must be archived not deleted

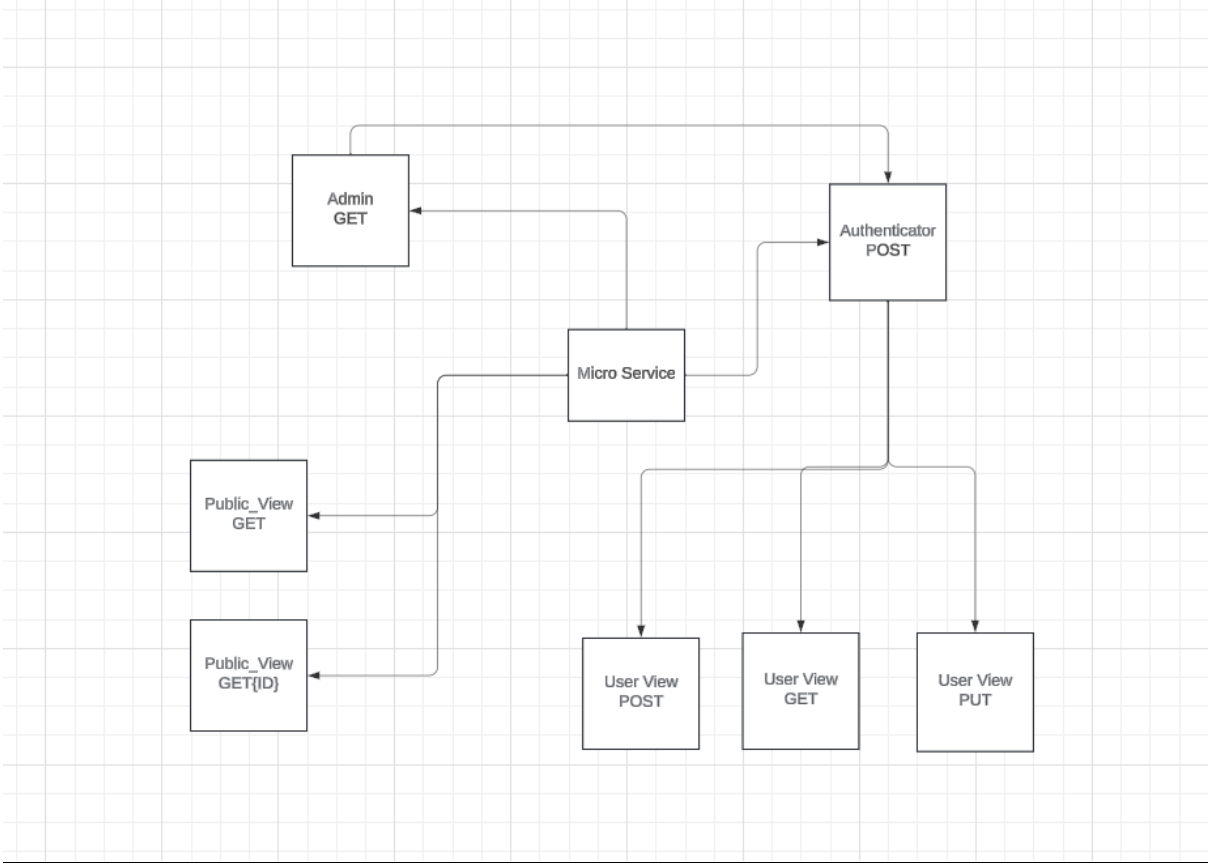
Database Design:

The Database consists of four tables that each store various information about the user such as their name, weight, height, birthday and more. All the user Data is stored under the table "CW2.USER_DATA", this is also the table that gets updated on the API once the POST command is ran. In visual studio, the GET command accesses the "CW2.USER_DATA" table to retrieve data to display to the user.

Logical ERD



UML



API

The API used facilitates communication and data exchange between the user interface and backend SQL Database. When users upload or interact with data in the app, the API handles these requests by fetching, updating or storing data in the database. This allows for real time data processing and ensures that the app remains responsive and up to date with the user's inputs and actions.

The application was required to use the existing Authenticator API:
https://web.socem.plymouth.ac.uk/COMP2001/auth/api/users

Security Considerations:

It was crucial we mitigated all possible security concerns when creating this application considering we are storing sensitive user data. If an issue was to occur and as a result user data was stolen than we could be held accountable for breaching regulations such as GDPR and the Data Protection act. Failure to comply with these regulations lead to issues such as financial penalties, reputational damage, legal actions, lawsuits and more.

To comply with these regulations, we ensured that user data was stored securely, this was done by using Microsoft SQL Server as the server of choice. During the design process multiple databases were researched however, this was chosen as it is arguably one of the most secure database software available due to its constant updates, positive reviews and being one of the most popular cloud-based databases .

In order to maintain the integrity of this application, we have made the limited view data such as name and profile picture only accessible to users with an account. This mitigates the chance of trolls and spam attackers as they will not be able to freely view data that can be used maliciously. This reinforces comfort and a sense of security within the users.

Legal, Social, Ethical and Professional (LSEP)

Ethical consideration was given to customers privacy by obtaining consent from the user before collecting personal data. When a user accesses the alltrails website, the site presents the user with a prompt that informs the user what data will be collected and what it will be used for.

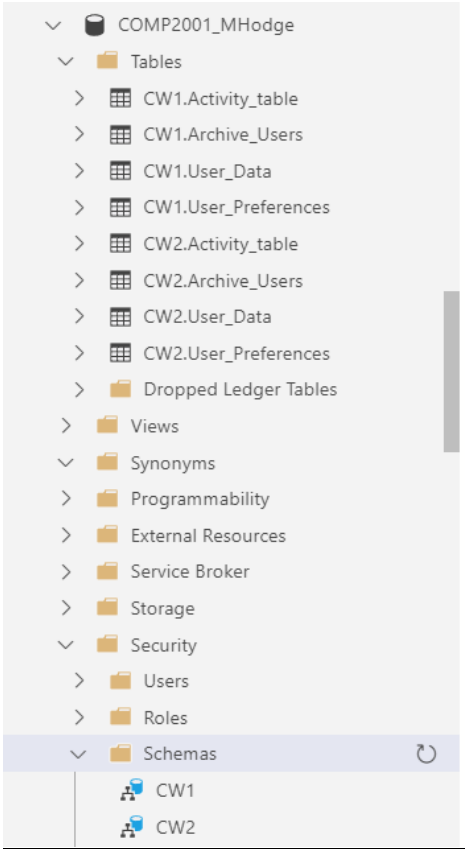
Consideration was also given to users who may not remember/be aware of their user ID, when a user views all accounts by using the public view controller, the user ID is shown. This allows the users to access their user ID without having to log in to their account. Consideration was given to this decision, we ensured that only restricted user details were displayed as this information would be accessible to anyone, therefore no personal data was displayed in this view.

Legally we had to ensure that we followed the regulation around data security as this application handles personal data such as addresses , phone numbers and email addresses. The laws we had to adhere to were the GDPR act and Data Protection act. We took precautions such as prior research to find the most appropriate database software, it was crucial to use the most secure database that was available to keep user's data secure. The regulations also included how we would user data and where we would share it, so it was necessary to inform the user of any changes or upcoming data transfers before conducting them. The application has aimed to be as transparent as possible with the users about what data is stored and how we use their data.

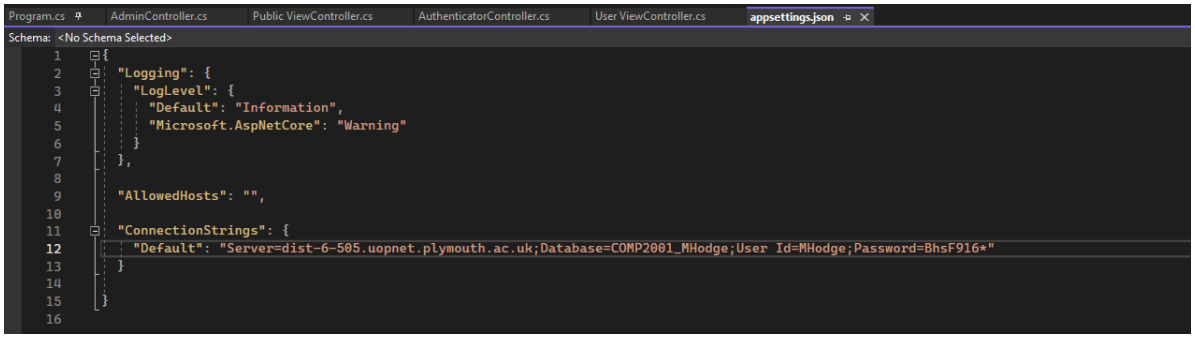
Because this is a popular trails app used by many nature enthusiasts, the error handling had to be conducted in a professional manner. It had to be conducted in a way that non IT familiar users can understand the error messages. To resolve this issue, we ensured all errors returned a useful message that is user friendly and professional.

Implementation

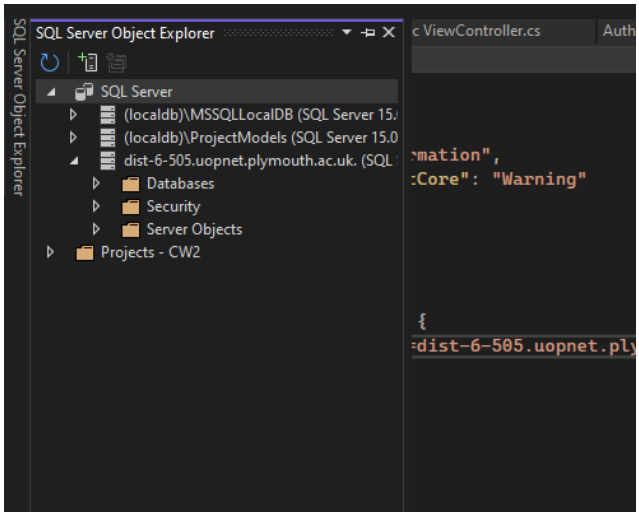
After the application design was finished, the database was ready for deployment. The initial task was to establish a new schema instance in the database called "CW2." Once this was done, all of the information and tables from the CW1 Schema were moved to the CW2 schema. This was required to protect the data in an independent duplicate of the schema so that, in the event that an error occurred, I could always restore the original database from a backup.



Once the database was prepared for implementation, the next step was to setup a workspace to write the code, the code had to be written using C# and ASP.net. This was completed by using visual studio with empty API framework with read/write actions. The SQL Database connection to Visual Studio was the first task in Visual Studio to be completed. This required establishing a connection string within the appsettings.json file that allowed the connection between the two software's.

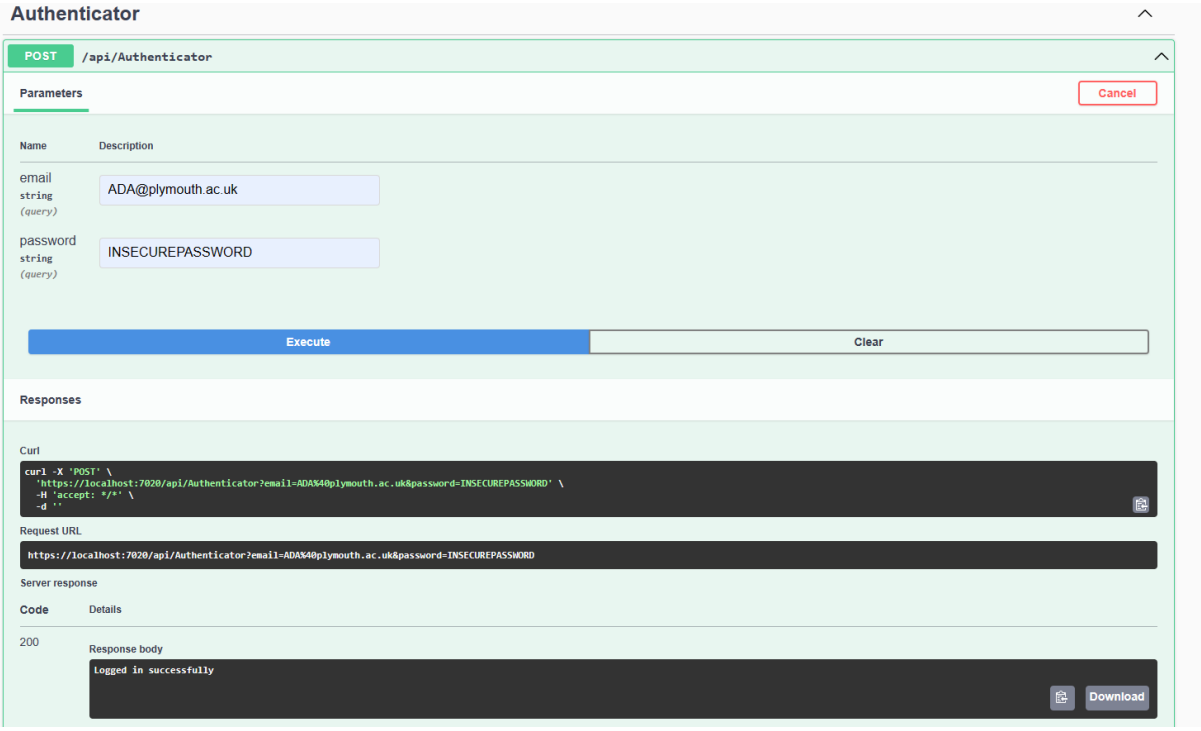


Once this was established, the SQL Server Object explorer was configured to connect to the database, this allowed a live view of the database within visual studio.

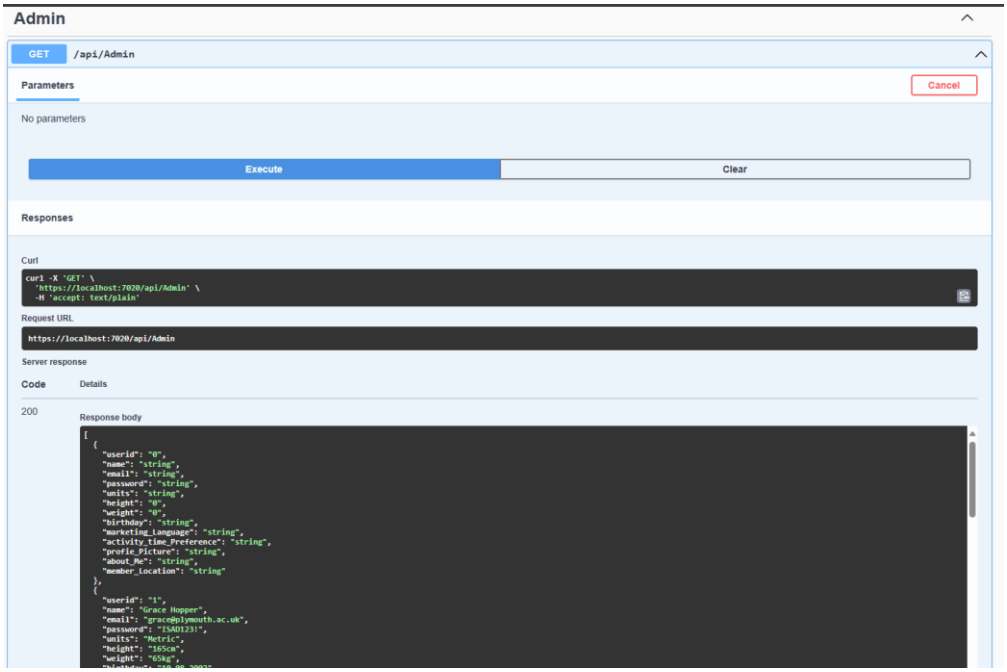


Once the code could access and interact with the database, the next step was to create the controllers. Controllers act intermediary between the model and the view, receiving user input and deciding how to interact with the model and update the view, they are used to handle HTTP requests such as Get, Post, Put and Delete. This would be how the CRUD operations are created and used through a variety of HTTP requests.

The first controller created was AuthentiactorController, this controller used a SQL select command to access the database and view to see if a user's entered credentials match with that on the database. If the match is a success the user will be logged into their account and message will be returned saying "Logged in Successfully".



The next stage of implementation was to create the admin view, the admin access was given to the trail example data, so all three users can access the database as admins due to it being sample data. The admin can view all users in the database using GET and use the Delete request to archive profiles.



Public View controller was then created, this controller allows anyone to view all profiles but have limited view. For this GET and GET{ID} were used, the GET method displays to the user a limited view of the data for all users whereas GET{ID} allows users(if they know) to enter an userid and be displayed with the corresponding user. As this data can be viewed by all users, we ensured to maintain the security of the data by only displaying selected information.

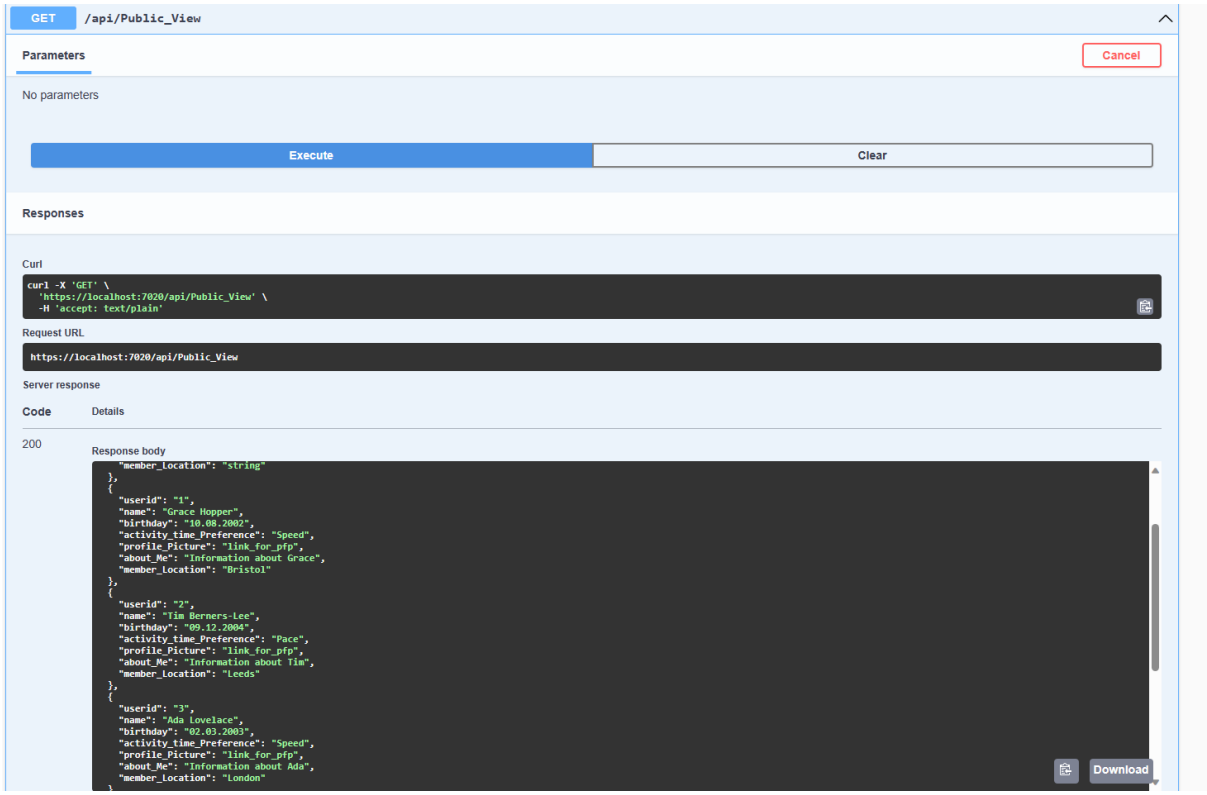


Figure 1 GET

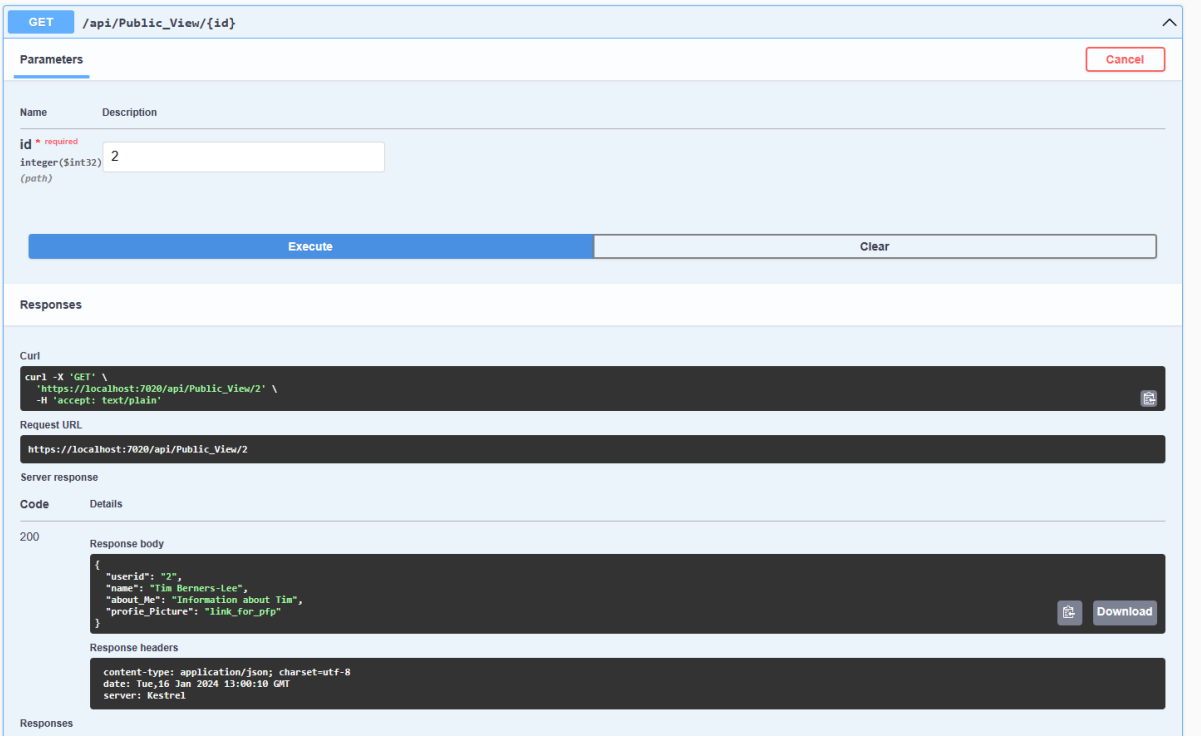
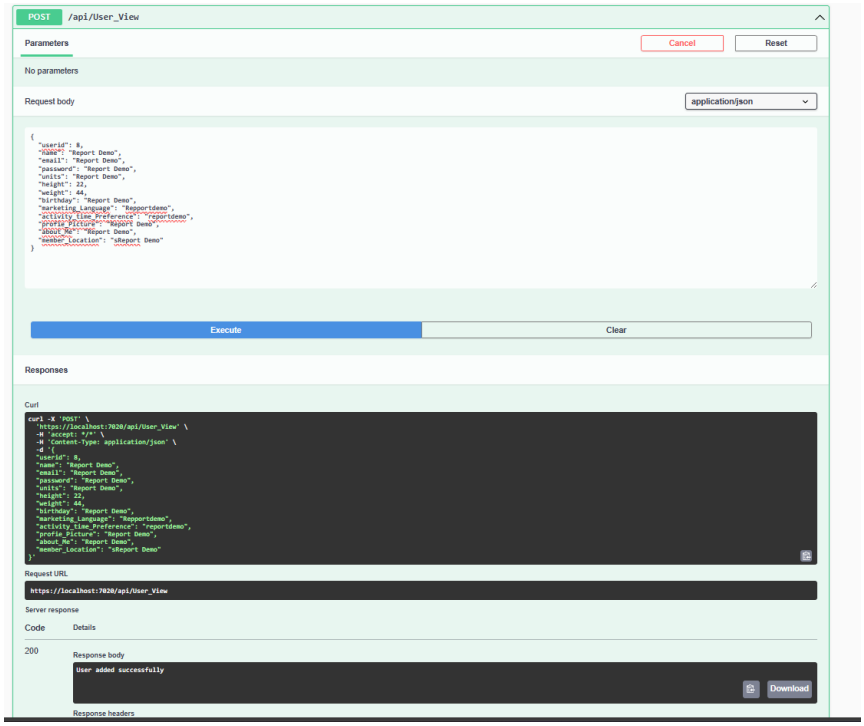
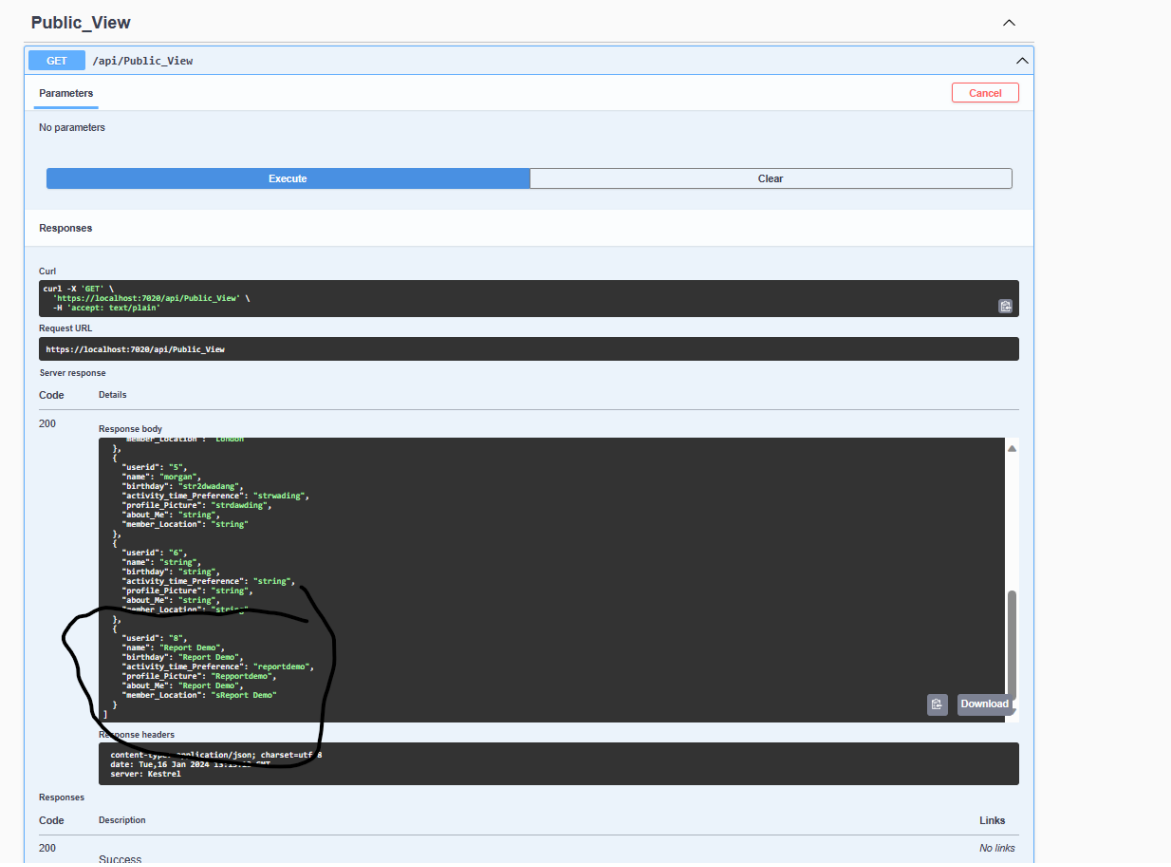


Figure 2 GET{ID}

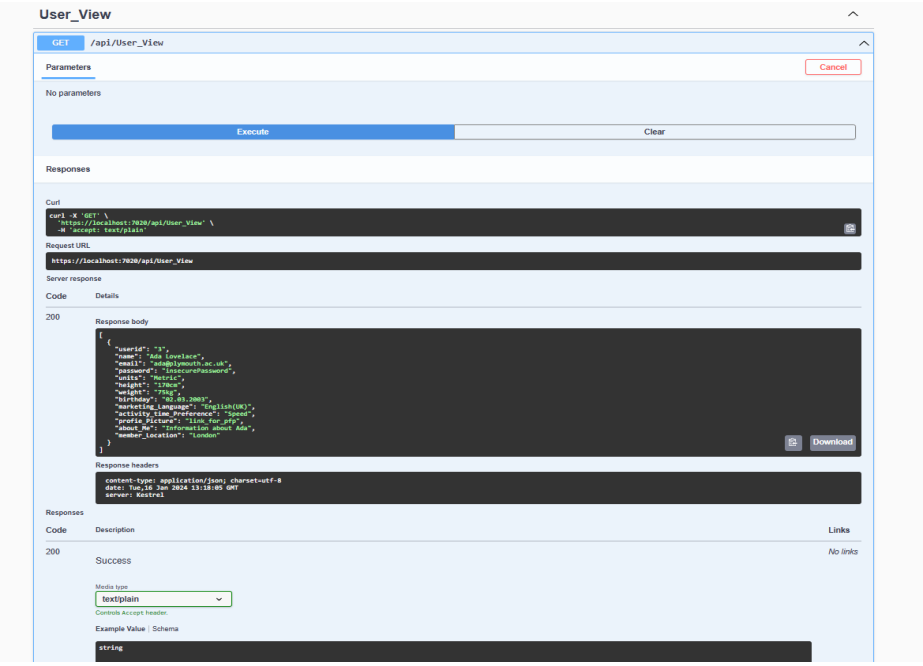
The final controller I created was User_View, this controller allowed users that were logged in and confirmed by the “Authenticator” , to access and view their accounts. It also allowed users to create a new account, and that account be uploaded to the database. The method for creating a new user was a Post method, this used the SQL command INSERT to INSERT INTO CW2.USER_DATA to insert the new account details into the database.



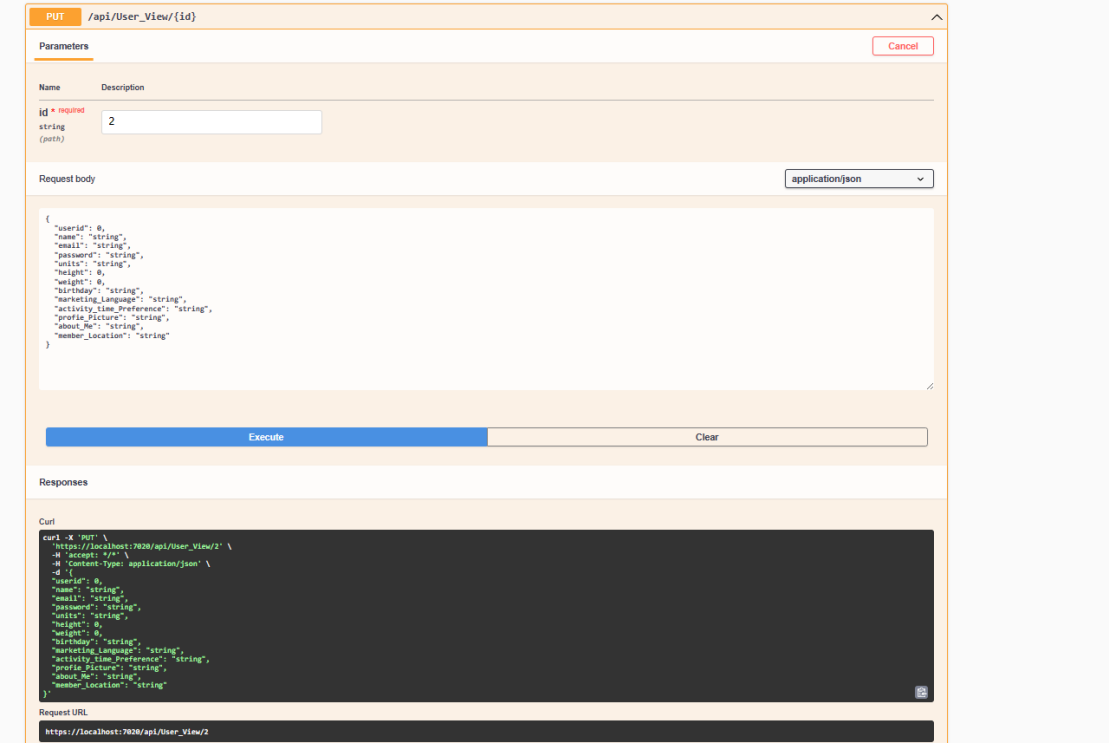
After a user has been added, the data in the other controllers such as public view and admin will be updated to display the new added data.



Within the User View controller there is a GET method that reads from the authenticator to ensure that the user is logged into an account, the GET will then display all the logged in users details using a SELECT query. For this demonstration I am logged in as Ada Lovlace:



The PUT command within the user controller allows users to update their user details, this is done by using an INSERT query and will update the existing column and value.



Evaluation

Testing

I created a testing table to check the functionality of the controllers and operations that take place within this code:

Controller Name	Request	Response	Pass or Fail
ADMIN	GET	Gets data and displays all data on users within the database	PASS
AUTHENTICATOR	POST	Validates data inserted with data in database, if match user is logged in, if not error message is displayed	PASS
USER VIEW	GET	After a user is logged in public view will display the logged in users' data	PASS
	POST	Creates a new account within the database	PASS
	PUT	Edits User Data within the database	FAIL
PUBLIC VIEW	GET	Displays all user data but with limited view on what data is displayed	PASS
	GET {ID}	Gets data based on the ID provided, data is displayed with the corresponding account to inputted ID	PASS

In conclusion the implementation of this project was a success as the requirements set by the clients were met. Users can perform CRUD operations within their accounts using the GET, POST, PUT, and Delete operations. This was demonstrated in the implementation section of this report as users can update records, create accounts , read data, and delete data, which all correspond with CRUD operations.

Throughout the documentation and creation of this project I considered the LSEP related problems that could occur within the user requirements . I handled these problems well by following legislations and maintain data integrity. An area to further improve on would be the privacy matter, I believe I could have done more to help strengthen the user's privacy when using the app, I believe this could have been done by only allowing users that have an account to access the limited view.

What went well in this project is that I gained skills and knowledge on how to write in ASAP.NET and the use of connection strings and controller classes. These skills will help me in the future within my computing career, therefore I gained something valuable from doing this.

To further improve in the future, I would like to have a stored procedure in place within the database that can INSERT user data without having to manually type out each table collum as I believe this would be more efficient and effective. To improve within myself I need to learn how to manage time better as I feel with more time and better time management skills, I would be able to add more detail and functionality to my program.