Github - [Ozzy-King/comp2001_70 (github.com)]

Deployed API - [Swagger UI (plymouth.ac.uk)]

Contents

Introduction

This is my report for comp 2001 70% coursework. This document will talk about the background of what the micro service is, the design around the micro service and how it works, the issues around privacy, integrity and security and how I remedied these issues as well as how I've prevented the top 10 OWASP vulnerabilities, and the implementation of the actual micro service with some snippets of code and the rough steps each request takes.

Background

For the 70% part of the module I was tasked with re implementing course work 1 database work into course work 2 with reduced tables and some better procedures. There are less tables in course work 2 as there would be too many table to manages and keep track of. I was also tasked with creating an API with asp.net that facilitated the ability to for CURD(creation, update, read, delete) operations on the database tables, along with a swagger front end to allow to easier testing. This course work is for the profile microservice, to allow other microservices to interact with it and retrieve required details.

Design

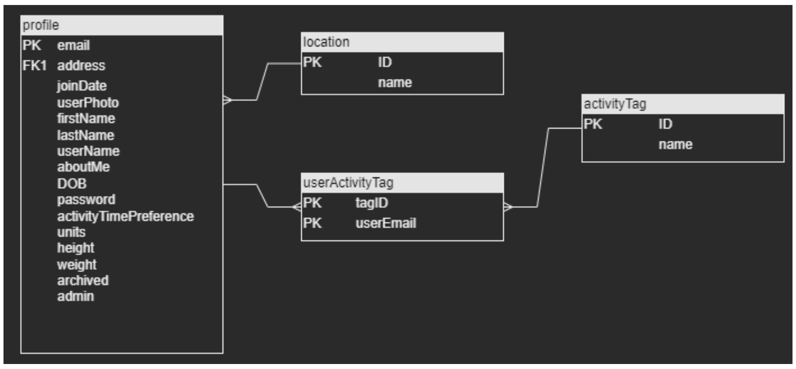
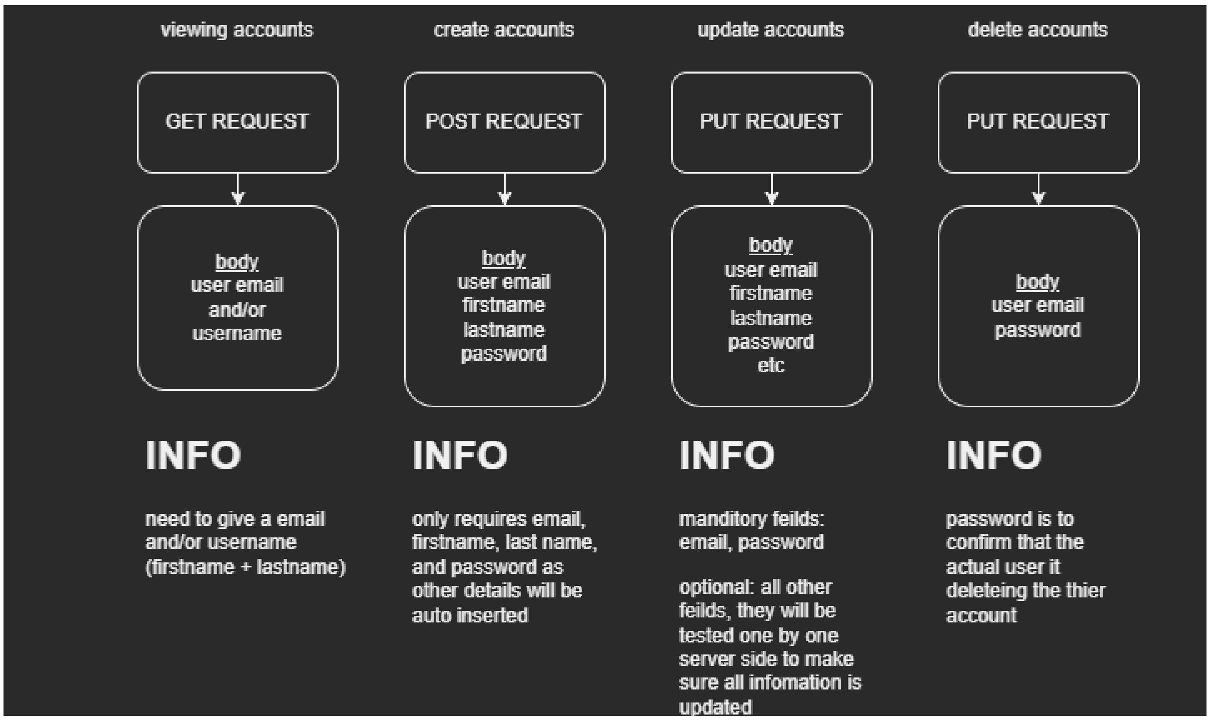


Figure 1- this is the new ERD for the coursework 2 database

Composite(complex) key for useractivty table

The ERD and the implemented schema are a reduced version of courseworks 1 schema. I've reduced the schema due to having too many table that would need to be kept track of, and greatly lengthening the time that would be needed to implement the required operations. I've included 4 table to fully demonstrate my understanding of how database relations work by showing many to one and many to many resolutions using a linker table. I also changed the linker table to include 2 primary keys that act as forgien keys as well.



need to give a email and/or username (firstname + lastname)

only requires email, firstname, last name, and password as other details will be auto inserted

manditory feilds: email, password

optional: all other feilds, they will be tested one by one server side to make sure all infomation is updated

password is to confirm that the actual user it deleteing the thier account

Figure 2 - CRUD layout for the profile tale

Figure 2 shows the basic requests required for full CRUD of the tables. Figure 2 is for profile specifically but can be easily modified to be applied to any other tables crud. For tag and location, you can only read from the database; this is due to the user not having the ability to create, delete, or update these tables, and only allows admins (people with access to the database directly) to add these tables. For profile you can see the info below explaining what is needed for each request, there will be a more in-depth insight into the actual format of the request bodies.

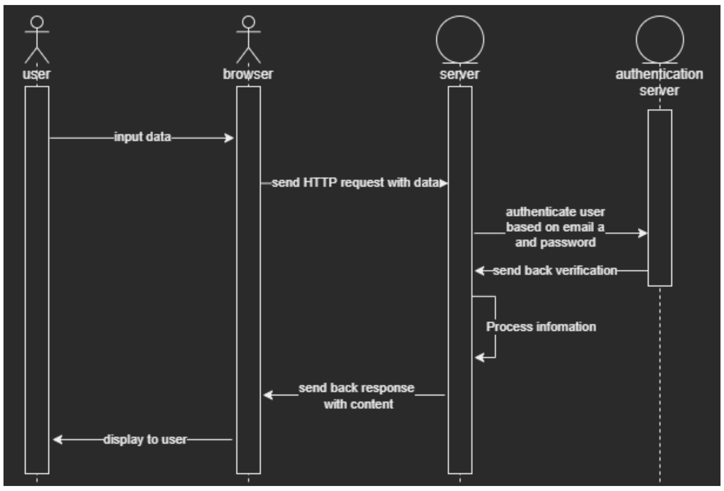

user
browser
server
authentication server
input data
send HTTP request with data
authenticate user based on email a and password
send back verification
Process infomation
send back response with content
display to user

Figure 3 - outline on the web browser and server interaction when sensitive information is includes as the server will authenticate with the authentication microservice.

Figure 3 is showing the basic request flow for the requests that would return sensitive data if successful, this is shown by the fact that there is a validation request of users from an external server, once the authenticator server returns, the API server will continue and process the request to return the appropriate value that begin data from the database or an error response.

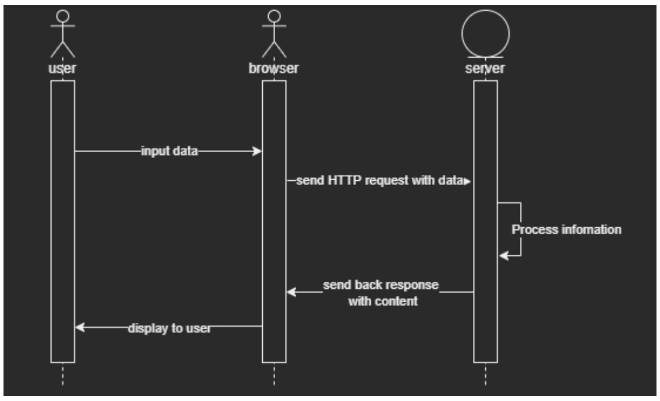


Figure 4- the interaction between web browser and server when no authentication is needed as no sensitive information is being attempted to be accessed.

Figure 4 is the same as the figure 3 but without the outside authentication, this is due to figure 4 showing the process flow for requests that don't hold sensitive data.

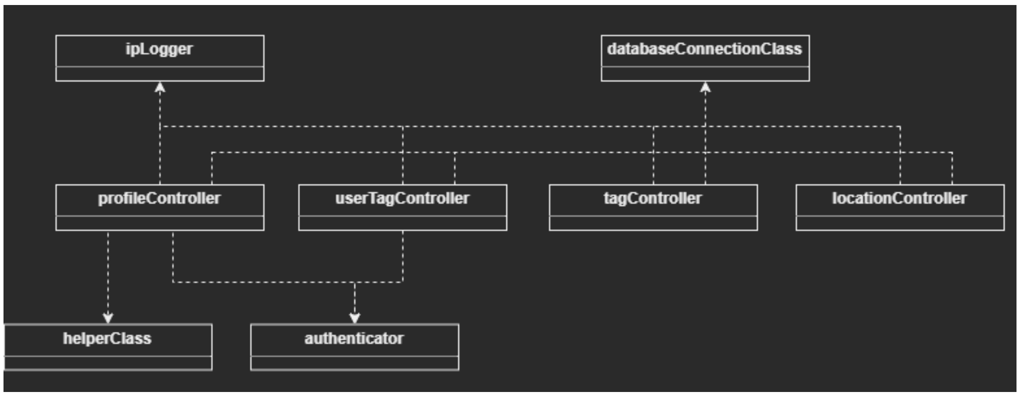


Figure 5 - class diagram showing the connectivity between classes and controllers.

Figure 5 is showing the connectivity between classes. The helper class is only ever used but the profileController to validate dates so they can be stored on the server, and the authenticator class used by the userTagController and profileController to be able to validate the actual user. All the Controllers will use the ipLogger class and databaseConnectionClass class so they can all log the Ip currently connecting and later restrict the Ip requests if needed, as well as connecting to the database to send requests for data and receive responses.

There are multiple ways to get data from browser to server and server to browser. The method I went with was a mix between key-value pairs in the URL (urlencoded) and body content, this allows for easily sending data over when reading table data, and then allow for a well-defined structured request body when creating, updating, and deleting. The format for the data to be sent in is Json instead of something like XML or YAML, as Json is considerably smaller, flexible and would be faster to send.

If I had longer to develop this API I would attempt to implement protobuf (Proto Buffer) sterilization as it is smaller and faster than Json, and is more rigid in how the data is sent over so less error could occur. If I where to start over I would change some collum names on the database to help show how each table is connected, for example change “address” to “locationID” in profile and change “ID” to “locationID” in the location table.

privacy, integrity, security and preserving the data

Information security

There is two layers of authentication on the parts that require authentication of the user, as well as sanitisation of the input data so that injection can't occur. The server also has implemented a Ip tracker to only allow a certain number of requests per second, currently being 25; this is to stop any kind of brute forcing of passwords and potentially prevents DOS (denial of service) attacks. Security I have missed out but would add with more time would be to salt and hash the passwords so if malicious actors gain access to the database the passwords can't be stolen and used.

Information privacy

The only part of the database that should be private/inaccessible from unauthorised users is the user's information itself. Unless the correct details are input then the sensitive information won't be sent to the user. An extra step I took to increase information privacy is by creating a view that holds all the public facing details, this separates out the sensitive information so only the database can access the profile table directly instead of the API.

Information integrity

Integrity is kept through the 2 layers of authentication, the first is through the authenticator server and then on the database side on the procedure. With this, data cannot be changed without first making sure the user is who they are, once they've be validated, they are allowed to input data along the request. Before data is sent to the database the server will sanitise the information to prevent any injections.

Information preservation

When inputting data on to the server through the API, preservation is kept on the server side before going on the database such as checking date format is correct, integers and string are in correct format and that all the data required is supplied.

Implementation

I first set up a project in visual studio code that ended up being wrong as I was planning on using ".cshtml" to display was retrieved from the database, but quickly found out this couldn't work with swagger so ended up deleting an restarting and the project to actual have the ability to use swagger and create proper controllers for the API. (Clark1, 2023)

I still haven't updated the project by the second commit but did start working on the architecture for API using draw.io and seen earlier as well as creating the tables, views and procedures for the database coursework 2. I tweaked the userActivityTag table from coursework1 by creating 2 primary key collum to stop duplicate from being made without the need for triggers (SQL optimization, n.d.). This only included the basics but was sufficient to get started on the actual project. (Clark2, 2023) I also did a quick commit to update the URL list for resources that I was using. (Clark3, 2023)

I finally switched to the API swagger project and started by creating the profile controller with the action to get public facing user data by receive input from the request, as well as creating a database connection class using connection string format from this website (sql-server, n.d.), to make it easier to send and receive information to and from the database (Clark4, 2023) I ended up needing to search up the reason for why the sqlconnection wasn't working and used the is website to help fix this issue (the-type-or-namespace-name-sqlconnection, n.d.)

```
            string connectionString = "Server=dist-6-
505.uopnet.plymouth.ac.uk;Database=COMP2001_OClark;User Id=OClark;Password=GgyC627+;";
            string commandString = "SELECT * FROM cw2.publicUserData where email = '" + email +
"'";
            DatabaseConnectionClass databaseConnection = new DatabaseConnectionClass();
            databaseConnection.connect(connectionString);
            return databaseConnection.executeCommand(commandString);
```

Figure 6 - the code used to retrieve public profile data (it gets updated later on to be more secure)

The fifth commit added saw quite a bit of addition. I implemented POST, PUT and DELETE for the profiles as well as some more GET requests for the public profile data. I added a sanitization(figure 7) procedure, using this website to escape the correct characters (Blog, n.d.)

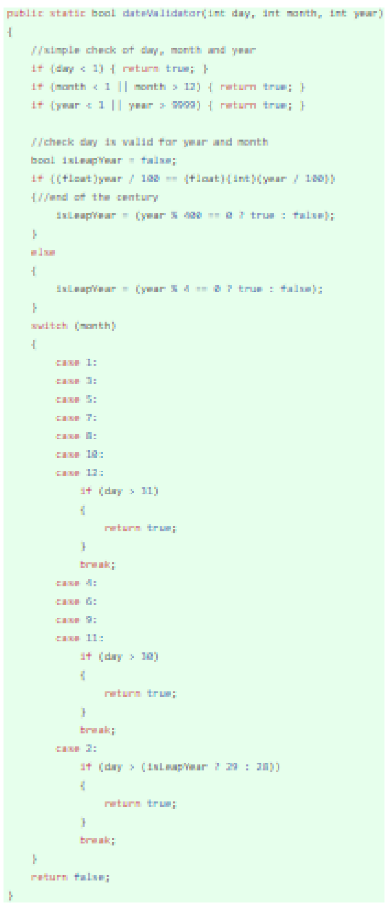
, to the database connection class to be used on the values passed to be inserted into the sql commands as well as error codes and error string builder to give between feedback to the user. A helper class was also created with a date validator function to be able to test in the date added is a valid date or not (figure 8). I also added data to the location table to be tested in the future. (Clark5, 2023)

The sixth commit was to separate out the location controller into its own controller as well as create a tag controller that handle GET requests only. (Clark6, 2023) between this commit and the last commit I also started to implement the sanitization of function into the actions to prevent sql injection from occurring.

```
public string sanatizer(string command) {
    string output = command;
    //need to sanatize \ first or it will try to sanatize the newly added backslashes
    output = output.Replace("\\", "\\\\");
    output = output.Replace("'", "''");
    output = output.Replace("\"", "\\\"");
    output = output.Replace("--", "\\-\\-");
    output = output.Replace("%","");
    output = output.Replace("_","");
    return output;
}
```

Figure 7 - sanitsation function

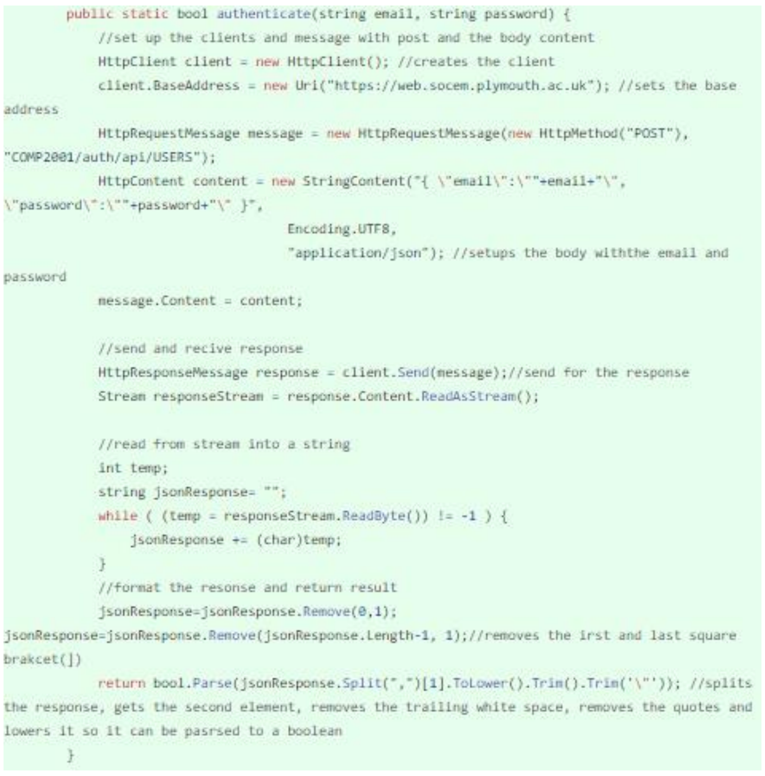
The seventh commit included usertag controller creation for crud, as well as creating an authentication class and function to be able to authenticate the users are who they say they are. To create the authentication(figure 9) class and function I used the httpclient class using Microsofts documentation to get started (Make HTTP requests with the HttpClient - .NET, n.d.) . it got a bit fiddly towards the end of the procedure to isolate the required data from the string array. (seventh commit, 2023)



```
public static bool dateValidator(int day, int month, int year)
{
    //simple check of day, month and year
    if (day < 1) { return true; }
    if (month < 1 || month > 12) { return true; }
    if (year < 1 || year > 9999) { return true; }

    //check day is valid for year and month
    bool isLeapYear = false;
    if ((float)year / 100 == (float)(int)(year / 100))
    {//end of the century
        isLeapYear = (year % 400 == 0 ? true : false);
    }
    else
    {
        isLeapYear = (year % 4 == 0 ? true : false);
    }
    switch (month)
    {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            if (day > 31)
            {
                return true;
            }
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            if (day > 30)
            {
                return true;
            }
            break;
        case 2:
            if (day > (isLeapYear ? 29 : 28))
            {
                return true;
            }
            break;
    }
    return false;
}
```

Figure 8 - date validation function



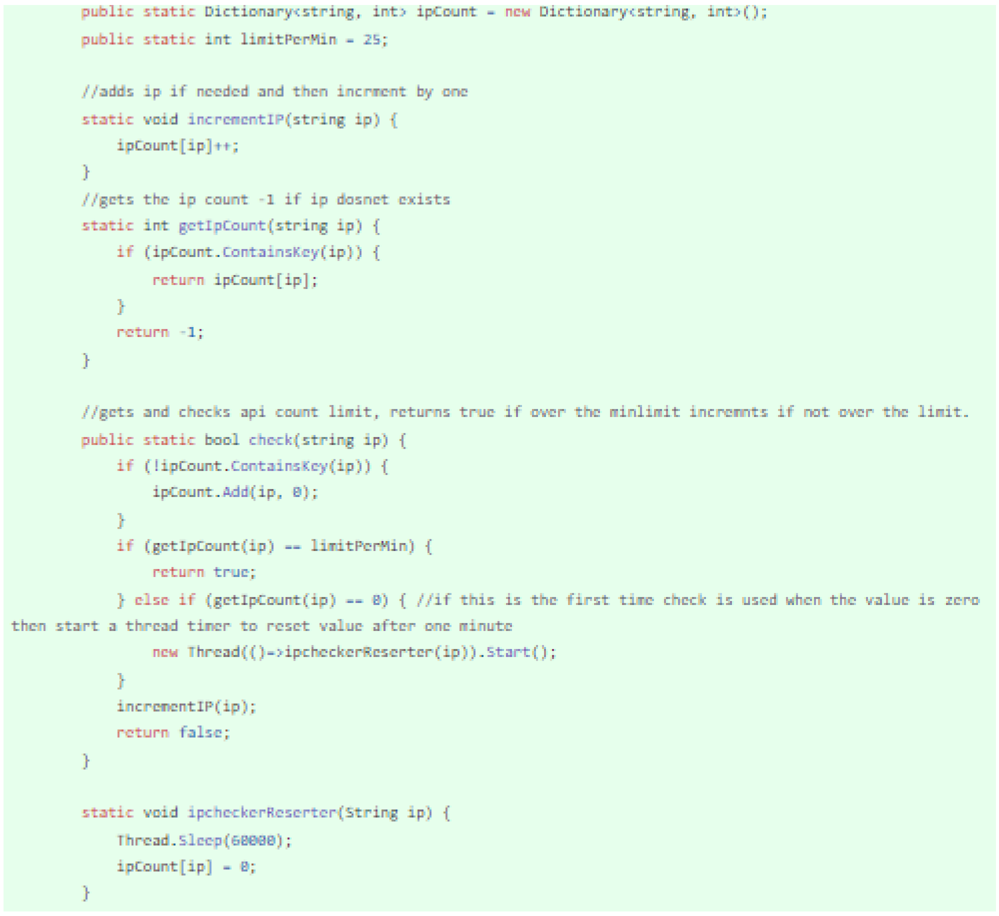
```
        public static bool authenticate(string email, string password) {
            //set up the clients and message with post and the body content
            HttpClient client = new HttpClient(); //creates the client
            client.BaseAddress = new Uri("https://web.socem.plymouth.ac.uk"); //sets the base
address
            HttpRequestMessage message = new HttpRequestMessage(new HttpMethod("POST"),
"COMP2001/auth/api/USERS");
            HttpContent content = new StringContent("{ \"email\":\""+email+"\",
\"password\":\""+password+"\" }",
                                        Encoding.UTF8,
                                        "application/json"); //setups the body withthe email and
password
            message.Content = content;

            //send and recive response
            HttpResponseMessage response = client.Send(message);//send for the response
            Stream responseStream = response.Content.ReadAsStream();

            //read from stream into a string
            int temp;
            string jsonResponse= "";
            while ( (temp = responseStream.ReadByte()) != -1 ) {
                jsonResponse += (char)temp;
            }
            //format the resonse and return result
            jsonResponse=jsonResponse.Remove(0,1);
jsonResponse=jsonResponse.Remove(jsonResponse.Length-1, 1);//removes the irst and last square
brakcet([)
            return bool.Parse(jsonResponse.Split(",")[1].ToLower().Trim().Trim('\"')); //splits
the response, gets the second element, removes the trailing white space, removes the quotes and
lowers it so it can be pasrsed to a boolean
        }
```

Figure 9- authenticator function in authenticator class

The eighth commit is when I added a Ip tracker to allow the ability to limit the API calls to 25 per minute to prevent dangerous individuals from trying the brute force passwords. this utilises thread creation on first call and will reset the API call count to 0 after 60 seconds and will start again on the subsequent API call.



```
        public static Dictionary<string, int> ipCount = new Dictionary<string, int>();
        public static int limitPerMin = 25;

        //adds ip if needed and then increment by one
        static void incrementIP(string ip) {
            ipCount[ip]++;
        }
        //gets the ip count -1 if ip dosnet exists
        static int getIpCount(string ip) {
            if (ipCount.ContainsKey(ip)) {
                return ipCount[ip];
            }
            return -1;
        }

        //gets and checks api count limit, returns true if over the minlimit incremnts if not over the limit.
        public static bool check(string ip) {
            if (!ipCount.ContainsKey(ip)) {
                ipCount.Add(ip, 0);
            }
            if (getIpCount(ip) == limitPerMin) {
                return true;
            } else if (getIpCount(ip) == 0) { //if this is the first time check is used when the value is zero
 then start a thread timer to reset value after one minute
                new Thread(()=>ipcheckerReserter(ip)).Start();
            }
            incrementIP(ip);
            return false;
        }

        static void ipcheckerReserter(String ip) {
            Thread.Sleep(60000);
            ipCount[ip] = 0;
        }
```

Figure 10- the ipLogger class showing the connection between functions and how to checks and resets the counters`1

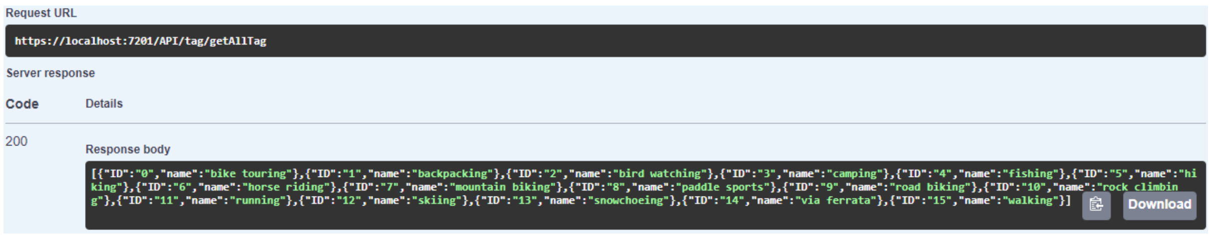
Evaluation.

I tested the API through swagger using the json layout as per the structure below

TagController

getAllTag

Using a standard GET request



Request URL

```
https://localhost:7201/API/tag/getAllTag
```

Server response

Code Details

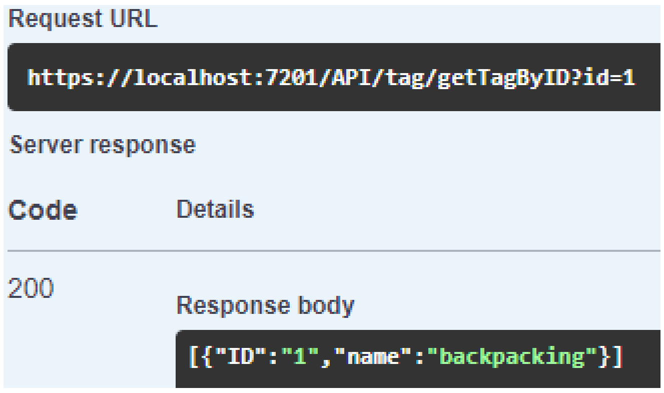
200

Response body

```
[{"ID":"0","name":"bike touring"},{"ID":"1","name":"backpacking"},{"ID":"2","name":"bird watching"},{"ID":"3","name":"camping"},{"ID":"4","name":"fishing"},{"ID":"5","name":"hi
king"},{"ID":"6","name":"horse riding"},{"ID":"7","name":"mountain biking"},{"ID":"8","name":"paddle sports"},{"ID":"9","name":"road biking"},{"ID":"10","name":"rock climbin
g"},{"ID":"11","name":"running"},{"ID":"12","name":"skiing"},{"ID":"13","name":"snowchoeing"},{"ID":"14","name":"via ferrata"},{"ID":"15","name":"walking"}]
```

getTagByID

a GET request for a key-value pair.


Request URL

```
https://localhost:7201/API/tag/getTagByID?id=1
```

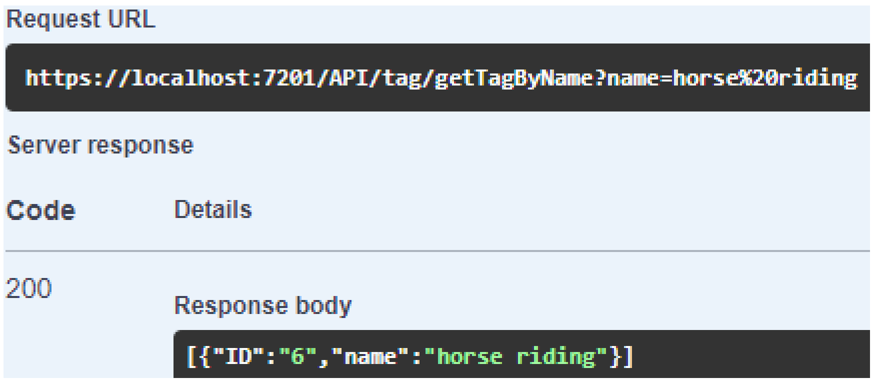
Server response

Code	Details
200	Response body

```
[{"ID":"1","name":"backpacking"}]
```

getTagByName

another GET request with key-value pair


Request URL

```
https://localhost:7201/API/tag/getTagByName?name=horse%20riding
```

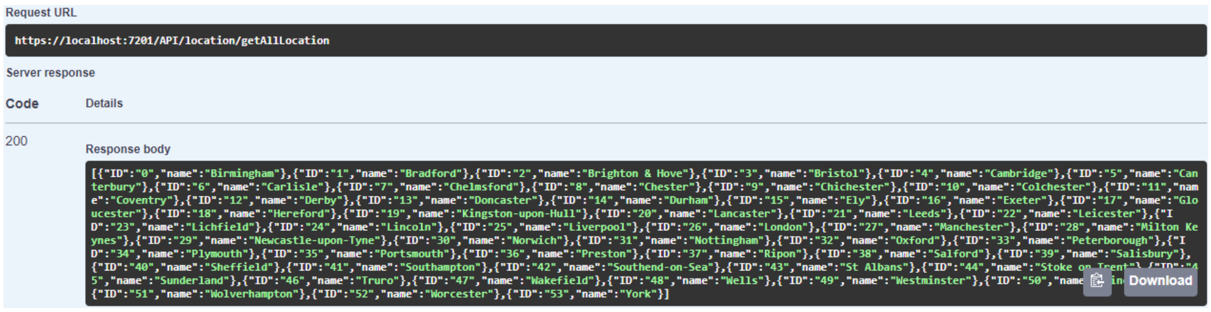
Server response

Code	Details
200	Response body

```
[{"ID":"6","name":"horse riding"}]
```

locationController

getAllLocation


Request URL

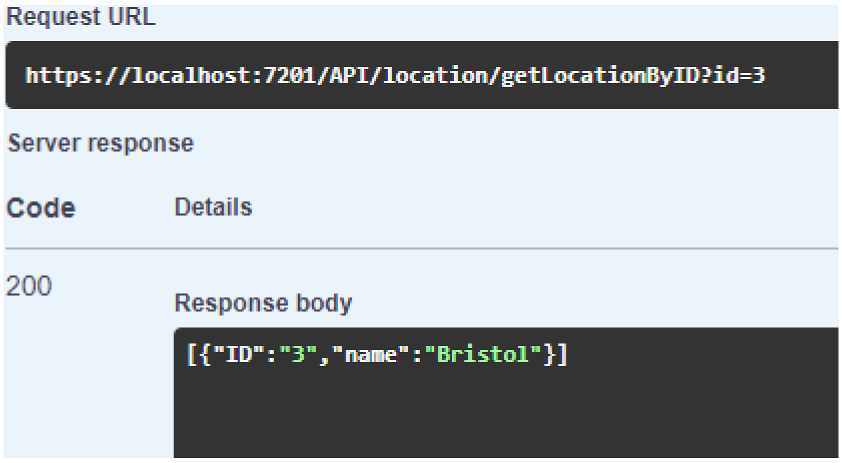
```
https://localhost:7201/API/location/getAllLocation
```

Server response

Code	Details
200	Response body

```
[{"ID":"0","name":"Birmingham"},{"ID":"1","name":"Bradford"},{"ID":"2","name":"Brighton & Hove"},{"ID":"3","name":"Bristol"},{"ID":"4","name":"Cambridge"},{"ID":"5","name":"Can
terbury"},{"ID":"6","name":"Carlisle"},{"ID":"7","name":"Chelmsford"},{"ID":"8","name":"Chester"},{"ID":"9","name":"Chichester"},{"ID":"10","name":"Colchester"},{"ID":"11","nam
e":"Coventry"},{"ID":"12","name":"Derby"},{"ID":"13","name":"Doncaster"},{"ID":"14","name":"Durham"},{"ID":"15","name":"Ely"},{"ID":"16","name":"Exeter"},{"ID":"17","name":"Glo
ucester"},{"ID":"18","name":"Hereford"},{"ID":"19","name":"Kingston-upon-Hull"},{"ID":"20","name":"Lancaster"},{"ID":"21","name":"Leeds"},{"ID":"22","name":"Leicester"},{"I
D":"23","name":"Lichfield"},{"ID":"24","name":"Lincoln"},{"ID":"25","name":"Liverpool"},{"ID":"26","name":"London"},{"ID":"27","name":"Manchester"},{"ID":"28","name":"Milton Ke
ynes"},{"ID":"29","name":"Newcastle-upon-Tyne"},{"ID":"30","name":"Norwich"},{"ID":"31","name":"Nottingham"},{"ID":"32","name":"Oxford"},{"ID":"33","name":"Peterborough"},{"I
D":"34","name":"Plymouth"},{"ID":"35","name":"Portsmouth"},{"ID":"36","name":"Preston"},{"ID":"37","name":"Ripon"},{"ID":"38","name":"Salford"},{"ID":"39","name":"Salisbury"},
{"ID":"40","name":"Sheffield"},{"ID":"41","name":"Southampton"},{"ID":"42","name":"Southend-on-Sea"},{"ID":"43","name":"St Albans"},{"ID":"44","name":"Stoke on Trent"},{"ID":"4
5","name":"Sunderland"},{"ID":"46","name":"Truro"},{"ID":"47","name":"Wakefield"},{"ID":"48","name":"Wells"},{"ID":"49","name":"Westminster"},{"ID":"50","name[illegible]in
{"ID":"51","name":"Wolverhampton"},{"ID":"52","name":"Worcester"},{"ID":"53","name":"York"}]
```

getLocationByID


Request URL

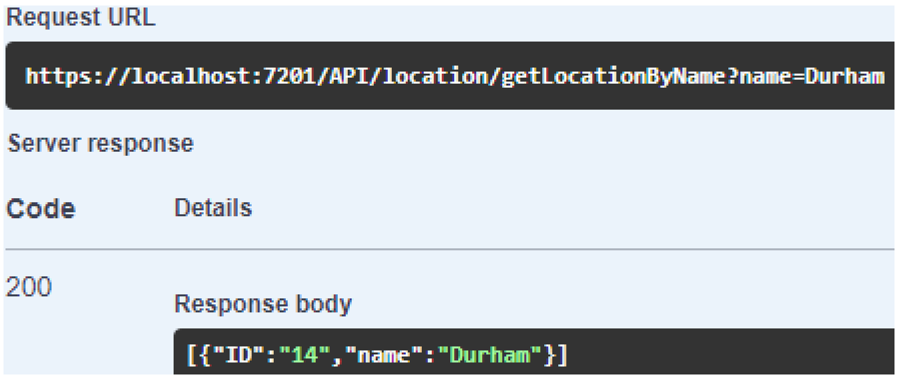
```
https://localhost:7201/API/location/getLocationByID?id=3
```

Server response

Code	Details
200	Response body

```
[{"ID":"3","name":"Bristol"}]
```

getLocationByName



Request URL

```
https://localhost:7201/API/location/getLocationByName?name=Durham
```

Server response

Code Details

200

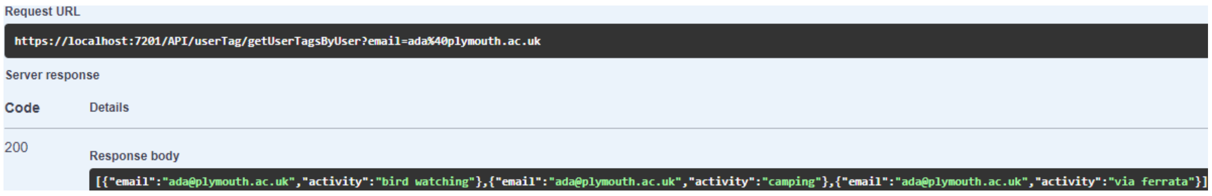
Response body

```
[{"ID":"14","name":"Durham"}]
```

userTagController

getUserTagsByUser

you pass an email in the key-value pair and will return all the users tags



Request URL

```
https://localhost:7201/API/userTag/getUserTagsByUser?email=ada%40plymouth.ac.uk
```

Server response

Code Details

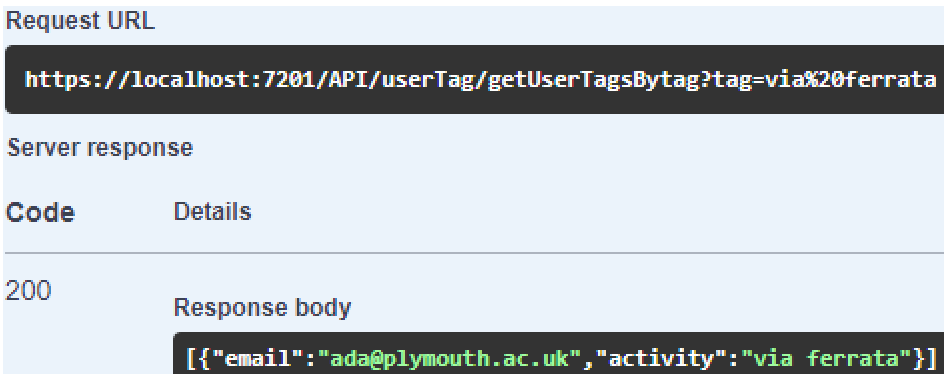
200

Response body

```
[{"email":"ada@plymouth.ac.uk","activity":"bird watching"},{"email":"ada@plymouth.ac.uk","activity":"camping"},{"email":"ada@plymouth.ac.uk","activity":"via ferrata"}]
```

getUserTagsBytag

returns all user emails with specified tag



Request URL

```
https://localhost:7201/API/userTag/getUserTagsBytag?tag=via%20ferrata
```

Server response

Code Details

200

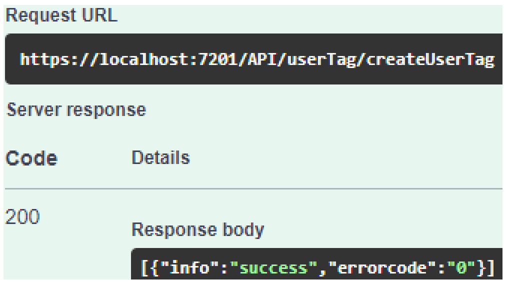
Response body

```
[{"email":"ada@plymouth.ac.uk","activity":"via ferrata"}]
```

createUserTag

instead of passing values through the key-value pair in the URL it is passed using Json on the body.

```
{
"email":"ada@plymouth.ac.uk",
"password":"insecurePassword",
"tagid":"10"
}
```



Request URL

```
https://localhost:7201/API/userTag/createUserTag
```

Server response

Code Details

200

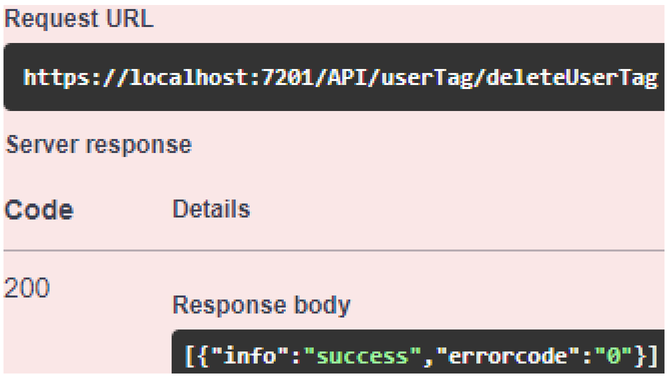
Response body

```
[{"info":"success","errorcode":"0"}]
```

deleteUserTag

deletes a tag accosiated with a specific user email

```
{
"email":"ada@plymouth.ac.uk",
"password":"insecurePassword",
"tagid":"10"
}
```



Request URL

```
https://localhost:7201/API/userTag/deleteUserTag
```

Server response

Code Details

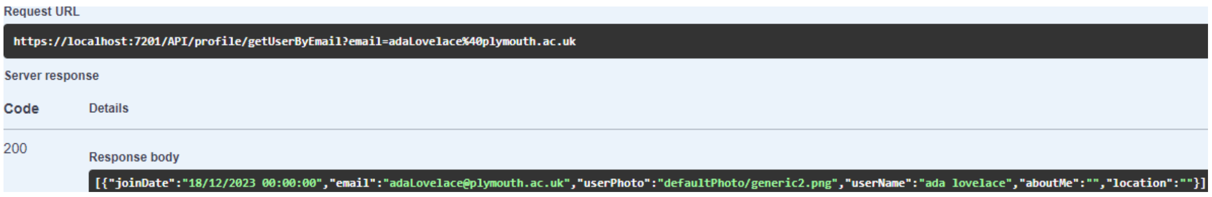
200 Response body

```
[{"info":"success","errorcode":"0"}]
```

profileController

getUserByEmail

this action is to retrieve public data of the user



Request URL

```
https://localhost:7201/API/profile/getUserByEmail?email=adaLovelace%40plymouth.ac.uk
```

Server response

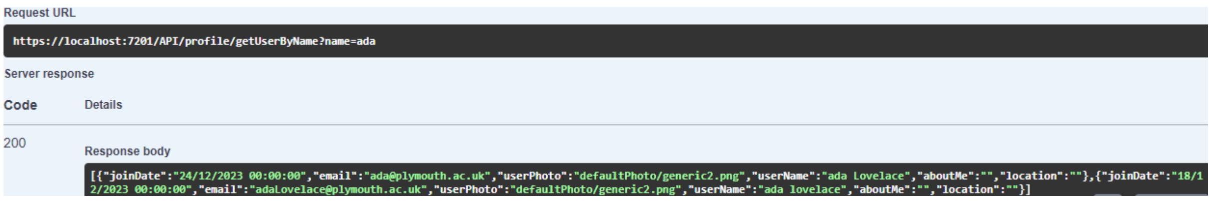
Code Details

200 Response body

```
[{"joinDate":"18/12/2023 00:00:00","email":"adaLovelace@plymouth.ac.uk","userPhoto":"defaultPhoto/generic2.png","userName":"ada lovelace","aboutMe":"","location":""}]
```

getUserByName

retrieve public data of users that have the entered text in their name (this action uses some wild cards)



Request URL

```
https://localhost:7201/API/profile/getUserByName?name=ada
```

Server response

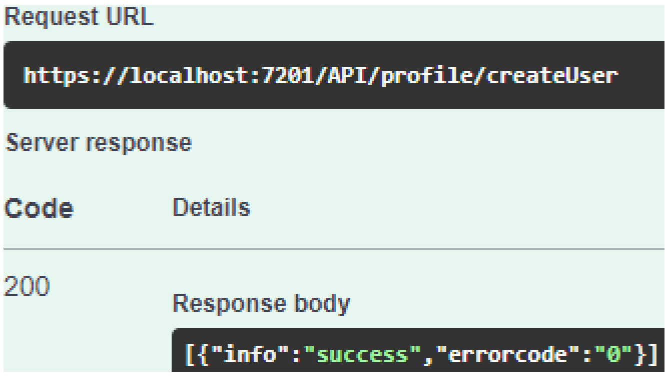
Code Details

200 Response body

```
[{"joinDate":"24/12/2023 00:00:00","email":"ada@plymouth.ac.uk","userPhoto":"defaultPhoto/generic2.png","userName":"ada Lovelace","aboutMe":"","location":""},{"joinDate":"18/1
2/2023 00:00:00","email":"adaLovelace@plymouth.ac.uk","userPhoto":"defaultPhoto/generic2.png","userName":"ada lovelace","aboutMe":"","location":""}]
```

createUser

```
{
"email":"grace@plymouth.ac.uk",
"password":"ISAD123!",
"firstname":"Grace",
"lastname":"Hopper"
}
```


Request URL

```
https://localhost:7201/API/profile/createUser
```

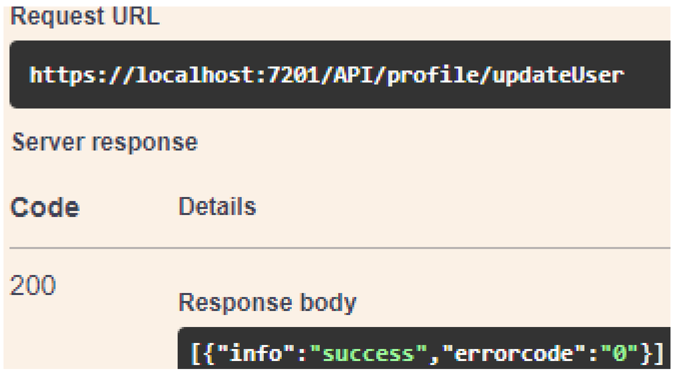
Server response

Code	Details
200	Response body `[{"info":"success","errorcode":"0"}]`

updateUser

the only mandatory fields in the json are the email and password, the rest are optional and there is also a “newpassword” field that can be added but is difficult to test now due to the account authentication being added, but before implementation did work successfully.(there is a full json schema in the github readme)

```
{
"email":"ada@plymouth.ac.uk",
"password":"insecurePassword",
"firstname":"adaChange",
"lastname":"loveLaceChange",
"userphoto":"testingPhotot",
"aboutme":"aboutme",
"DOB":"15-07-2024",
"activityPreference":"0",
"unit":"1",
"height":"1",
"weight":"1",
"address":"1"
}
```


Request URL

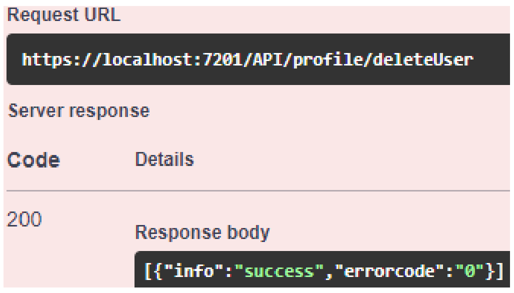
```
https://localhost:7201/API/profile/updateUser
```

Server response

Code	Details
200	Response body `[{"info":"success","errorcode":"0"}]`

deleteUser

```
{
"email":"ada@plymouth.ac.uk",
"password":"insecurePassword"
}
```


Request URL

```
https://localhost:7201/API/profile/deleteUser
```

Server response

Code Details

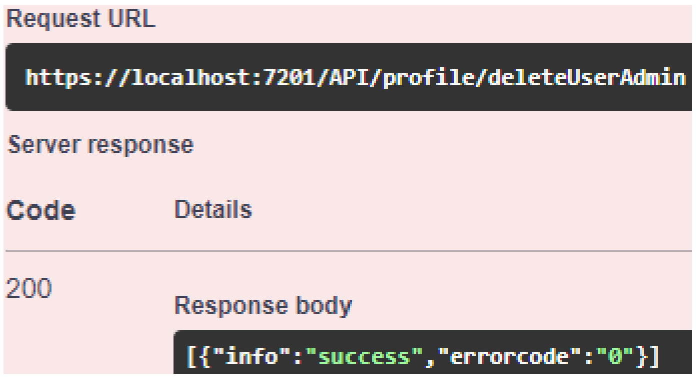
200 Response body

```
[{"info":"success","errorcode":"0"}]
```

deleteUserAdmin

this deletes/archives an account using an admins account.

```
{
"email":"ozzy@gmail.com",
"adminemail":"ada@plymouth.ac.uk",
"adminpassword":"insecurePassword"
}
```


Request URL

```
https://localhost:7201/API/profile/deleteUserAdmin
```

Server response

Code Details

200 Response body

```
[{"info":"success","errorcode":"0"}]
```

As well the action on the server, the server can also return error messages, this is beneficial as if documentation for how to use the API is not available hopefully the error messages can help with how it works. When an archived admin account is used it will say that the accounts email or password is incorrect, if it isn't archived and email and password are correct the selected user account will be archived.

References

(n.d.). Retrieved from Blog: https://www.playerzero.ai/advanced/sql-facts/how-to-escape-characters-with-the-in-operator-in-sql

Clark1, O. (2023, december). *firstCommit*. Retrieved from github: https://github.com/Ozzy-King/comp2001_70/commit/d6550b0d78ff7aca3ccbba456d1a8544049b9503

Clark2, O. (2023, december). *second commit*. Retrieved from github: https://github.com/Ozzy-King/comp2001_70/commit/c1573f415be72446141213b5cce6fe3ad4cffc40

Clark3, O. (2023, december). *third commit*. Retrieved from github: https://github.com/Ozzy-King/comp2001_70/commit/a6bb1ea00a71d461ff50feab055025138e8b62b7

Clark4, O. (2023, december). *fourth commit*. Retrieved from github: https://github.com/Ozzy-King/comp2001_70/commit/7de87f25b36e7130ace59f8f3aca3636451917e5

Clark5, O. (2023, december). *fifth commit*. Retrieved from github: https://github.com/Ozzy-King/comp2001_70/commit/73e605fef52b4cd4dc97e8112206b7470166a556

Clark6, O. (2023, december). *sixth commit*. Retrieved from github: https://github.com/Ozzy-King/comp2001_70/commit/6d81759c195f0280ed698ef2069dc2a8b0d9d9ab

Make HTTP requests with the HttpClient - .NET. (n.d.). Retrieved from Microsoft Learn : https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/http/httpclient

seventh commit. (2023, december). Retrieved from github: https://github.com/Ozzy-King/comp2001_70/commit/80ce6c8c0a12da25b212b34c304ddb0e11925a0b

SQL optimization. (n.d.). Retrieved from Primary key on two columns sql server: https://www.pragimtech.com/blog/sql-optimization/primary-key-on-two-columns-sql-server/

sql-server. (n.d.). Retrieved from connectionstrings: https://www.connectionstrings.com/sql-server/

the-type-or-namespace-name-sqlconnection. (n.d.). Retrieved from stackoverflow: https://stackoverflow.com/questions/29733221/the-type-or-namespace-name-sqlconnection