AberLink - Creating a link between University and Discord accounts

CS39440 Major Project Report

Author: Joel Adams (joa38@aber.ac.uk)
Supervisor: Dr. Neal Snooke (nns@aber.ac.uk)

31st March 2021
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in Computer Science and Artificial Intelligence (GG4R)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, U.K.

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Joel Luca Adams

Date: .....

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Joel Luca Adams

Date: .....

Acknowledgements

Mr. Dan Monaghan - for giving me the original inspiration for this project and for helping me along the way with understanding key concepts.

Dr. Neal Snooke - for collaborating with me on multiple Discord bots and improve the university online Discord presence. His bots were main motivation behind this project and I'd like to thank him for allowing me to re-create them for this project.

Dr. Alun Jones - for his continued support throughout this project and his resourcefulness in providing instructions on how to setup and use university services for this project.

Mr. Leslie Johns - for having setup the API endpoint for student attendance that has been crucial in getting this project off the ground.

My family and friends - for providing support and helping me get through university. A special thank you to everyone in the Comp Sci After Dark Discord server for being a great group of people and friends.

Dr. Angharad Shaw, Mr. Chris Loftus and the Computer Science Department - for helping to populate the Comp Sci Community Hub which is what lead to the university online Discord presence, without which wouldn't have lead to the creation of Neal Snooke's bots. A special thank you to Chris Loftus for acknowledging my work on DemoHelper and turning it into a job that inspired me to work on more Discord bots and the eventual creation of AberLink.

Abstract

Discord [1] is a video and text platform that allows users to set up servers for small communities. The Department of Computer Science has used it to create virtual module servers, similar to the current blackboard alternatives. The main issue however with these servers, however, is that they are always public so when an invite link is distributed it can be sent to anyone including people who are not related in any way to the University. This opens up servers to potential attacks and misuse. AberLink aims to solve this problem by providing the department with a mechanism to control access to module servers based on their membership with the University. This service also provides attendance monitoring of practicals which are located online on Discord in a specific server and at a specific time. This data is then fed into the central attendance monitoring system 'Student Record'.

AberLink consists of 2 components; a website that allows users to link up Discord [1] accounts to University accounts and a Discord bot called AberLink that controls users' access to module servers and records students' attendance.

The service was developed on the Linux distribution Debian 10 (Buster) [2] and uses the open-source HTTP server Apache2.0 [3] for website hosting along with the Python framework Django [4] for the website. This website is locked behind the Open-ID Connect [5] login system that only authenticates users with Aber accounts. Once logged in a user can link multiple Discord accounts to their Aber account. The service also employs the use of a Discord bot using the Python framework Discord.py [6]. The back-end database uses PostgreSQL [7] and is used to communicate between the website and the Discord bot.

The AberLink Discord bot is based on two previous bots called Aber Verify Bot [8] and I am here! [9], both created my Neal Snooke. These bots verified students' Discord accounts and helped to mark attendance during practicals respectively. AberLink greatly modifies the original implementation of these bots to use a proper back-end database and an API to update attendance.

Contents

List of Figures

List of Tables

Chapter 1

Background & Objectives

1.1 Background

This project required a substantial amount of discussion with IS and CS-support due to the sensitivity of this project and the data that it interacts with/stores. As meetings can sometimes take weeks to organise the background and spike work for this project was completed relatively quickly. By the end of the first week most of the spike work required to form the Project Outline was collected so an early version of the document was sent out to IS and CS-support to help get the ball rolling. This blog post also has some details about the first week of research here https://cs39440blog.wordpress.com/2021/02/01/week-1-25-01/.

Note: The dev folder of the technical submission also contains spike work that may not be listed below.

1.1.1 Research Spikework

Web Hosting & Containers - This was definitely one of the sections that I had the least experience working with but thanks to help from CS-support they guided me through the process and setup a Debian 10 (Buster) [2] container. Debian is definitely the best OS for this project as it is flexible and has many useful libraries for my project. This came preloaded with Apache2 [3] which is used to host websites so was the logical choice for me to work with. NGINX [10] was also considered for this project but was scrapped in favour of Apache2 due to the assistance available from CS-support and online documentation.

Coding Languages - Over the summer I worked on the DemoHelper [11] project and learnt how to create my own Discord bots in Python. This spawned many Discord bot projects that can be seen in the comp sci server. From these projects I gained a solid foundation of Python and is the reason behind me choosing this as my primary coding language this project. It also had a knock on effect for choosing the database and web framework that worked well or used Python. HTML, CSS and some Javascript was also used to develop the website pages.

Databases - Early on it was decided that a relational database was best suited for the data as it is only storing simple user information. The data would be split up into two tables; one for Aberystwyth user information and one for storing Discord account information. These would then be linked using a primary key in the Aberystwyth user table and then a foreign key in the Discord user table. The system would also be designed to allow users to have multiple Discord accounts so the database used a one-to-many relationship.

There are a few databases out there that support these features but PostgreSQL (PSQL) [7] was used as I am already familiar with using it in a second year project and it has native support for the web framework.

Website Frameworks - For the website framework I wanted to use something that would scale easily, have lots of documentation and preferably be written in Python. There are lots of frameworks that fit this category such as Flask and web2py but settled on Django. This was chosen because it came with integrated features such as custom user models, pre-made admin pages and authentication support. It also came with built in features to write data to and read from PSQL [7] so that was a big bonus too. Many other web frameworks were also considered for this project such as Node.js [12], React.js [13] or Angular.js [14] but were not used as I have little to no experience working in JavaScript and might be far more work than using a Python library.

1.2 Analysis

1.2.1 Objectives

After completing the spike work and prior research the next step was breaking the project down into sections and deliverables. Below is a list of the items that were used as milestones for the project completed in descending order.

- **Research & Discussion with IS/CS-support.**
  - Discuss how to access University data and how to login users who are only on the University network using OpenID Connect [5].
  - Build the attendance API so that it is secure and only marks students for current practical and not previous practicals.
  - Setup PSQL [7] and make it secure from outside attacks.
- **Version control, documentation and setup.**
  - Create a container on the University network to remain secure and locked under VPN access.
  - Documentation and version control using git on the University's GitLab upon the departments request so that it can be easily redeployed as a complete service later on.
  - A blog (usually bi-weekly) to document the process and progress of this project.
- **Creation of Python back-end for website and the database.**

  - Build the website using the Python framework Django [4].
  - Establish a PSQL [7] database for data storage.

- **Re-writing 'AberVerify' and 'I am here' into single Discord bot.**
  - Recreate the two mentioned bots in Python instead of JavaScript.
  - Make the bot use a Discord users information to lookup their Aber ID in the database using relational keys.
  - Make the bot update attendance using the University provided API endpoint.

- **Interface for lecturers and students on website.**
  - Create webpages for users to add Discord accounts, view the module servers they are in and the one's that they are not.
  - Create Admin pages only visible to staff to view all the connected students, remove them and configure their roles in servers for which they have administrative privileges.

- **Resource links and further webpages.**
  - Create a Discord bot function to get information such as the Aber account that is linked to the Discord account.
  - Discord bot function to display other useful discord bots that can be added to the server. e.g. DemoHelper [11]
  - Create webpages to display information such as the 'privacy policy', 'ethics form', 'blog' and 'about this project'.
  - Have a sort of search function for getting useful aber resources such as student record and aber su website.

- **Further potential work**
  - Integrate DemoHelper Discord bot into AberLink
  - Add multiple language support

1.2.2 Possible Security Issues

Security is an important component of this project as this system deals with federated authentication and controls access to Discord servers. If someone was to hack into the system they could potentially change their details to pose as another user, therefore, we must consider the security risks that this service may have. This section aims to cover risk analysis and robustness of the system against potential attack vectors.

- **Direct Database Access** - This is the first point of attack that could be used to get access to Discord and Aber account details. This has been secured by using a PostgreSQL [7] database that is located in Departments servers so it is already behind a very secure and up to date firewall. To access the database you also need to have a registered Aber account and be using the campus WiFi or the University's

VPN so that adds another layer of security. If the user however has access to both of these then brute forcing the PSQL login system is difficult as it has many security levels and incoming connections are monitored by the University.

- **Unauthorized Admin Access** - This is the possibility that the user will try and brute force access to the admin page of the website. Before they attempt this they would need to get past the website's OpenID Connect [5] authentication system that requires an Aberystwyth account to authenticate. The website also has another layer of security as it requires the user to be connected to the campus' internet or be logged in on the VPN. If they get past both methods there is no easy way to spoof the system to gain access to the administrator panel as the backend uses cookies to keep the user authenticated.
- **Accessing Database Credentials through Back-end Code** - All database credentials have been hidden in files that are not stored in the git repository and are loaded from JSON and .env files. This is good practice and creates a simple way for maintainers to setup and change variables such as database passwords easily instead of going through source code and manually editing it.
- **Accessing Data through AberLink Discord Bot** - The Discord bot does not contain any sensitive data, it merely queries the database using generic queries that are changed depending on the input variable. It can however be used to gain an understanding of the database model that is used.
- **Worst Case Scenario** - If the user somehow gains unauthorized access to the data they will only be collecting a list of emails and linked Discord accounts. No password data is ever stored in the database because OpenID Connect [5] is used to authenticate Aber accounts and Discord accounts are linked using OAuth2 [15]. This means that no password data is ever exchanged from the authentication systems to the website or database.

1.3 Process

For this project I found that Extreme Programming (XP) would best fit my project as early on I discovered that I worked best by breaking the project down into deliverables and then into components of work. You can see from the above section 1.2.1 the list of objectives (Functional Requirements) that I was working towards. Below are a list of the processes that I have followed or worked around in XP:

- **Project iterations** - These usually last around one week, however sometimes last longer due to unforeseen issues.
- **Pair Programming** - Due to the nature of this project being independent pair programming was not a viable approach. To compensate for this every morning after code was written I would go back through and review the code adding comments or refactoring the code.

- **TDD vs BDD vs FDD** - My style of coding usually revolves around writing code to pass some specific goal that has been set followed by user testing and finally unit testing (when applicable). Feature Driven Development (FDD) definitely fits my style best for this project.
- **When do iterations run?** - These usually ran from Monday to Friday so as to keep my weekends free to work on other projects and think over the next project iteration.
- **Where are requirements recorded?** - The project requirements can be found in the above section 1.2.1 and in the next section of this document 1.4. There is also a board of issues on the GitLab page that can be used to review the timestamps for when work was completed.

1.4 Functional Requirements

The following is a list of functional requirements used to test and evaluate the system. **FR1** - **FR2** are focused on testing the websites' back-end authentication and data saving. **FR3** - **FR4** are visual tests to check that users can see all their information and ensure that they do not have access to pages they should not. **FR5** - **FR8** are tests to ensure that the Discord bot works correctly.

- **FR1** - Authenticate Aber User accounts and return useful Metadata
  - **FR1.1** - Only authenticate Aber accounts and reject all other accounts.
  - **FR1.2** - Return Aber user information along with the usertype.
  - **FR1.3** - Make users with ”staff” usertype have administrative privileges.
- **FR2** - Authenticate Discord users accounts and link to Aber account
  - **FR2.1** - Once authenticated with Discord metadata is returned about account.
  - **FR2.2** - Discord account gets saved and linked to the Aber account using the primary key.
  - **FR2.3** - Multiple accounts can be linked to the same Aber account.
- **FR3** - Home webpage
  - **FR3.1** - Webpage displays Aber username and full name.
  - **FR3.2** - Webpage displays linked Discord accounts.
  - **FR3.3** - Discord ID's of linked accounts are sent to Discord API to get profile picture and username.
  - **FR3.4** - Users can view Discord module servers that they are in and join one's which they are supposed to be in.
- **FR4** - Admin webpage
  - **FR4.1** - Webpage only allows accounts which have the ”is_admin” permission to view.

  - **FR4.2** - Webpage displays a list of Aber user accounts.
  - **FR4.3** - Webpage displays a list of Discord user accounts that are linked to Aber accounts.
  - **FR4.4** - Webpage does not allow admins to change any of the accounts data apart from their admin access.
  - **FR4.5** - Webpage allows admins to delete a user or Discord account entry.
  - **FR4.6** - Website does not allow admins to add new Discord or Aber accounts.
  - **FR4.7** - Webpage that allows admins to monitor their Discord module servers and configure them.

- **FR5** - Discord bot verification
  - **FR5.1** - Bot can configure a server for verification.
  - **FR5.2** - Bot can very students when command is called or when the user joins and is already authenticated.
  - **FR5.3** - Bot has a message for Alumni verification.
  - **FR5.4** - After user is verified their Aber email gets appended to their username. e.g JoelinRome#4893 becomes JoelinRome [joa38].

- **FR6** - Discord bot attendance
  - **FR6.1** - Bot can get a users information from database using a reverse lookup using Discord ID and get Aber and linked Discord accounts.
  - **FR6.2** - Bot can mark students attendance in practicals by doing a reverse lookup of their aber username and sending it to an API endpoint. Then send the user a DM with the status of their attendance e.g. Attended module CS10000 at 11:00 on 13/04/2021.

- **FR7** - Discord bot utilities
  - **FR7.1** - Bot has a config file to choose wether a users name gets modified to include their Aber email at the end.
  - **FR7.2** - Bot will display the current server configurations.
  - **FR7.3** - Bot can clear all messages from a channel.
  - **FR7.4** - Bot displays a source command showing where source code can be found.
  - **FR7.5** - Bot displays a list of other useful bots that should be added to the server e.g. DemoHelper [11].
  - **FR7.6** - Bot can be pinged to get latency and check wether database connection is working correctly.
  - **FR7.7** - Bot has a search tool to get information on Aber University resources such as student record, blackboard and abersu website.

- **FR8** - Discord bot database connections
  - **FR8.1** - Connects to database and returns information on connection.

- **FR8.2** - Can reconnect to database in case of database disconnect during uptime.
- **FR8.3** - Python function to get a Discord user's information using their Discord ID.
- **FR8.4** - Python function to get a Aber user's information using their aber email.
- **FR8.5** - Python function to get database information e.g. connection status, latency and polling.

Chapter 2

Design

2.1 Overall Architecture

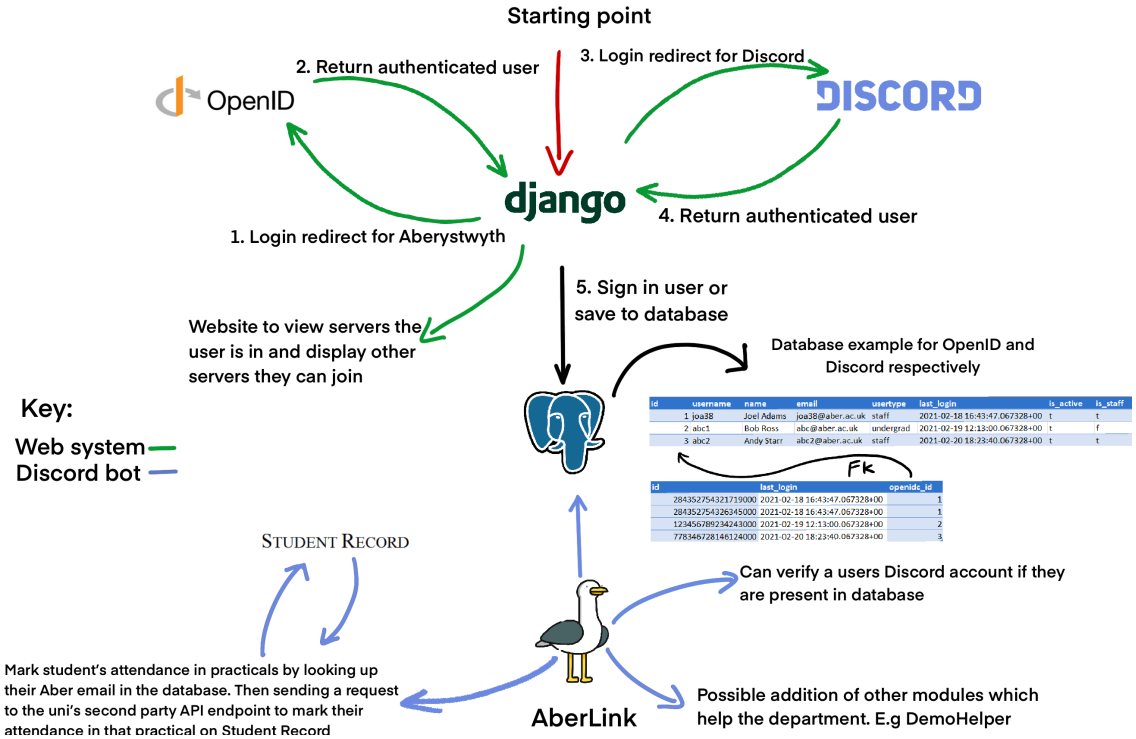

Figure 2.1: Architecture diagram of the overall system

As seen above here is a brief overview of the main components used in this project which are discussed in more detail later on in this document. Below is a list of the main folders and the files contained within them. The folder structure is based largely off on the second year group project folder structure.

- **config** - Describes how to setup the project; a bash script for installing the relevant

dependencies, a README.md for manual configuration of the project and example configuration files for Apache2 [3], authentication [16], Discord bot and Django [4].

- **dev** - Contains the spike work completed throughout this project and some information on errors encountered throughout this project and how they were overcome.
- **docs** - Contains the documents created for this project; including this document, the project outline, the ethics for and the usability form.
- **src** - Contains the source code required to run this project. There are two folders for easier maintainability; a folder for the website and a folder for the Discord bot.
  - **AberLinkAuthentication** - A folder containing all the website resources and the Django [4] code. It also contains a virtual environment [17] folder that is not stored in git because dependencies update all the time.
  - **AberLinkDiscord** - A folder containing the Discord bot.

2.2 Website Architecture and Design

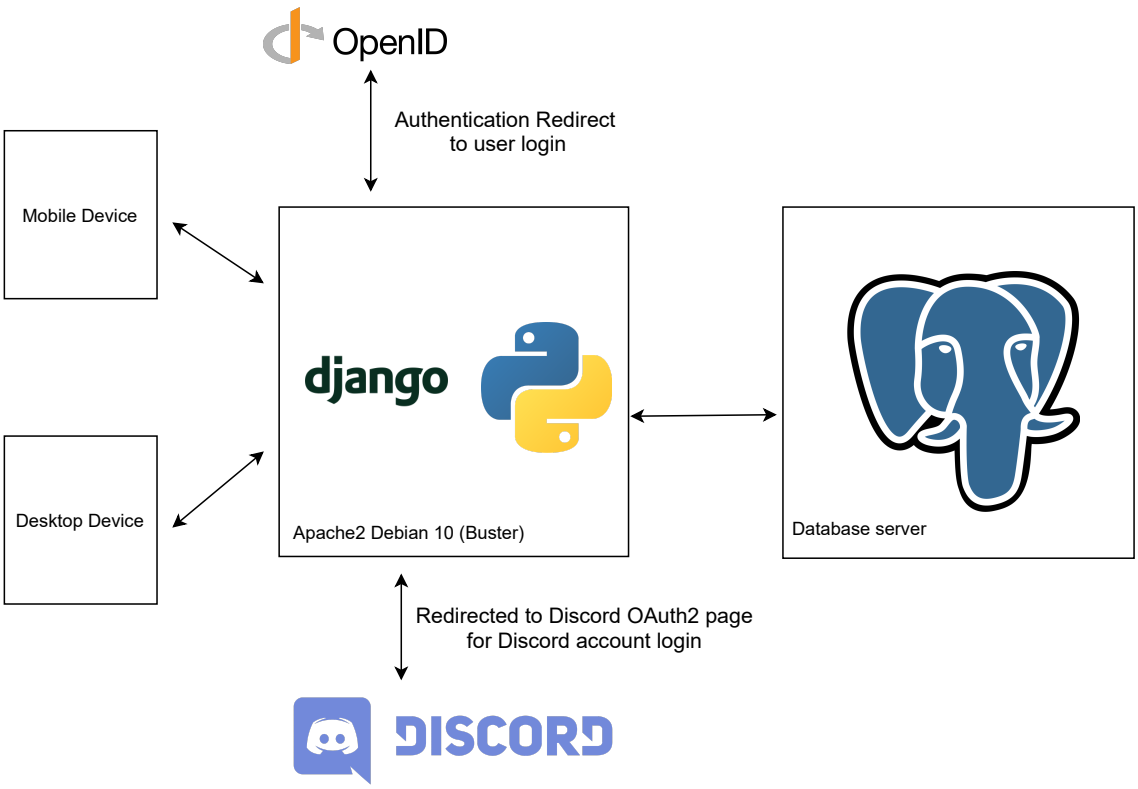

Figure 2.2: Architecture diagram of website

The website is built using the Python framework Django [4] that comes with the feature to generate a template project. This feature creates the overall file structure and generates all the code located in the folder `src\AberLinkAuthentication\AberLinkAuthentication`. This code has been slightly modified to support the use of a PostgreSQL [7] database instead of the default SQLite. The code has also been modified to work with Apache2 [3] using the ASGIref [18] Python library so that Apache2 can update the site when changes are made to the Django code. Apart from this folders code, the rest of the project is written from scratch using online resources and Django documentation.

The back-end of the website uses the HTTP server Apache2 [3] along with an SSL certificate provided using the Linux package Certbot [19]. An Apache2 mod called libapache2-mod-auth-openidc [16] is then used to make the website use OpenID Connect [5] authentication forcing the user to sign in when they visit the website. This helps to protect and prevent the site from any external attacks as all users get instantly redirected to the Universities OpenID Connect login page.

As a developer I... modified the Django template and wrote the code for the website.

As a maintainer I... am expected to change some code variables and install some packages as described in the README in Appendix D.

As a server owner (staff) I... am expected to make sure that all my students have linked their accounts to AberLink.

Once authenticated using OpenID Connect the users' data is then saved to the database and they can then add Discord accounts by using the button displayed on the screen. This then redirects them to Discord's OAuth2 authentication page and once they have logged in with a discord account then it redirects them back to the website and saves that information to the database as well. Included below is a flow chart explaining the process works.

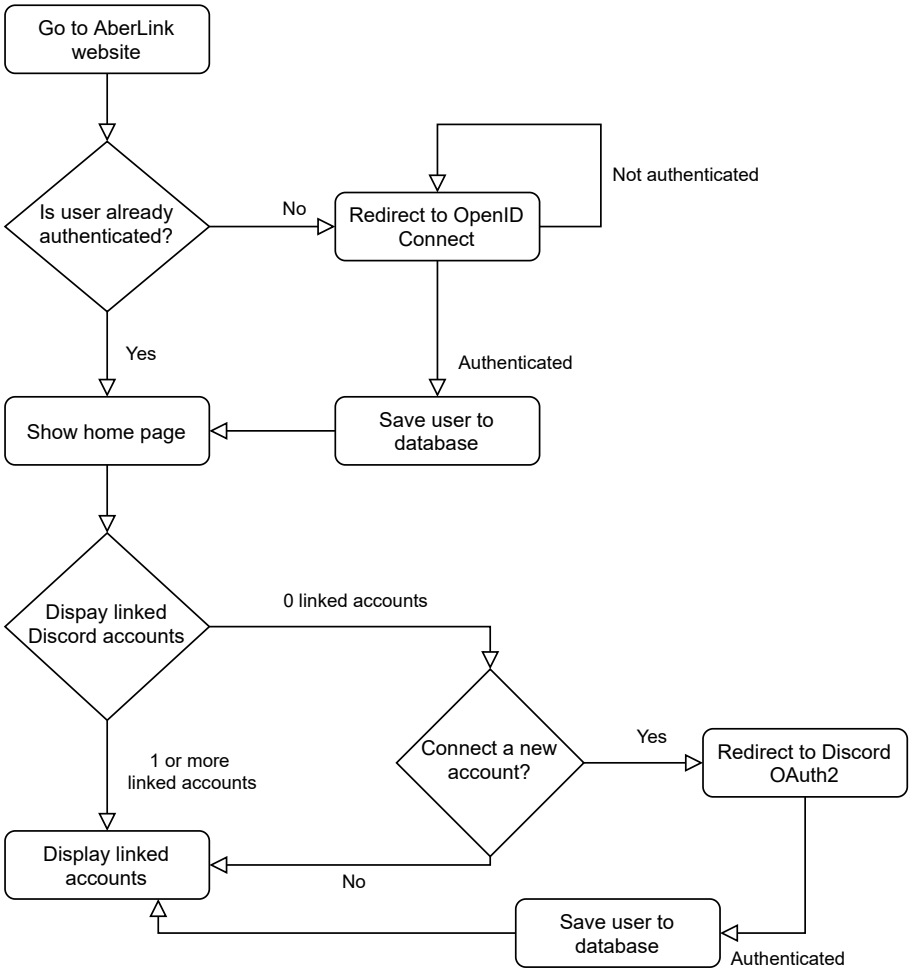

Figure 2.3: Website flowchart for authentication of accounts and display of user data

The website can be found here https://mmp-joa38.dcs.aber.ac.uk/ and is only accessible over VPN.

2.3 Discord Bot Architecture and Design

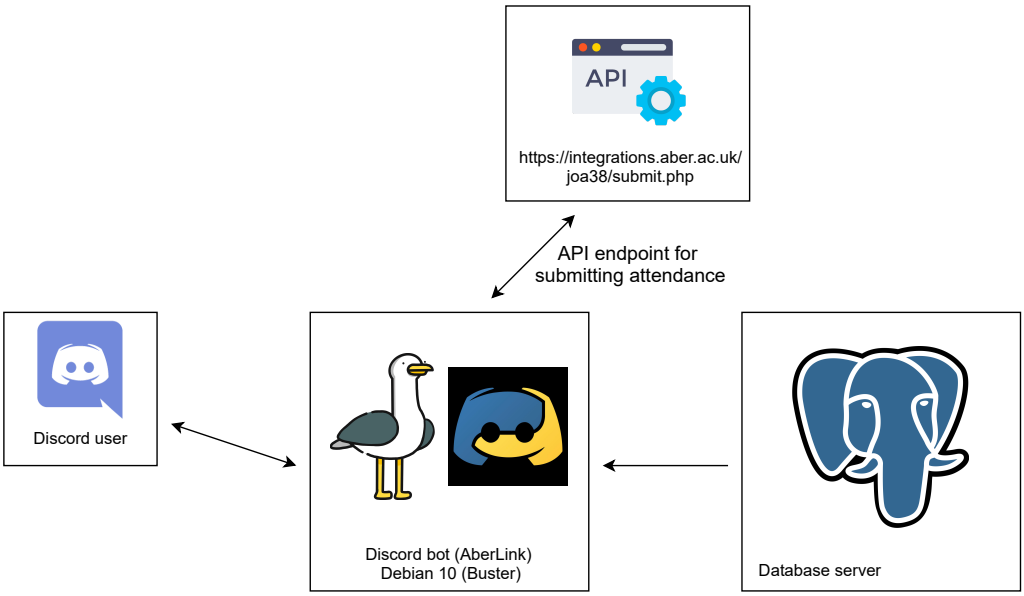

Figure 2.4: Architecture diagram of Discord bot

The Discord bot (AberLink) is usable through the Discord application available on mobile, desktop and their website. AberLink has many commands which can be easily found by typing the '!help' command in Discord on a server where the bot is present or via a direct message to the bot. The basic premise however is that it can access the database to retrieve information about a user but it never saves any information to the database however. The file structure is displayed below.

```
AberLinkDiscord
├─ AberLink.py
├─ Pipfile
├─ .env
├─ cogs
│  ├─ __init__.py
│  ├─ db.py
│  ├─ here.py
│  ├─ utilities.py
│  └─ verify.py
```

Figure 2.5: Tree diagram showing Discord file structure

As seen above in the file structure the main file that is used to run the program called AberLink.py, initialising the bot instance and loading all the files from the cogs folder. Discord.py [6] uses a smart system where code is not written into the main file (in this case AberLink.py) but instead broken down into components or 'cogs' as they are called here. The .env file is used to to store sensitive variables such as the Discord token required to run the bot and the data used to connect to the database server. This file is also not uploaded to git so the data is never exposed to the wider web and is only stored locally. The Pipfile is used to store the dependencies required to setup the bot such as discord.py [6], requests [20], psycopg2 [21] and the version of Python required to run the bot. The Pipfile can be initialised using the Python virtual environment pipenv [17].

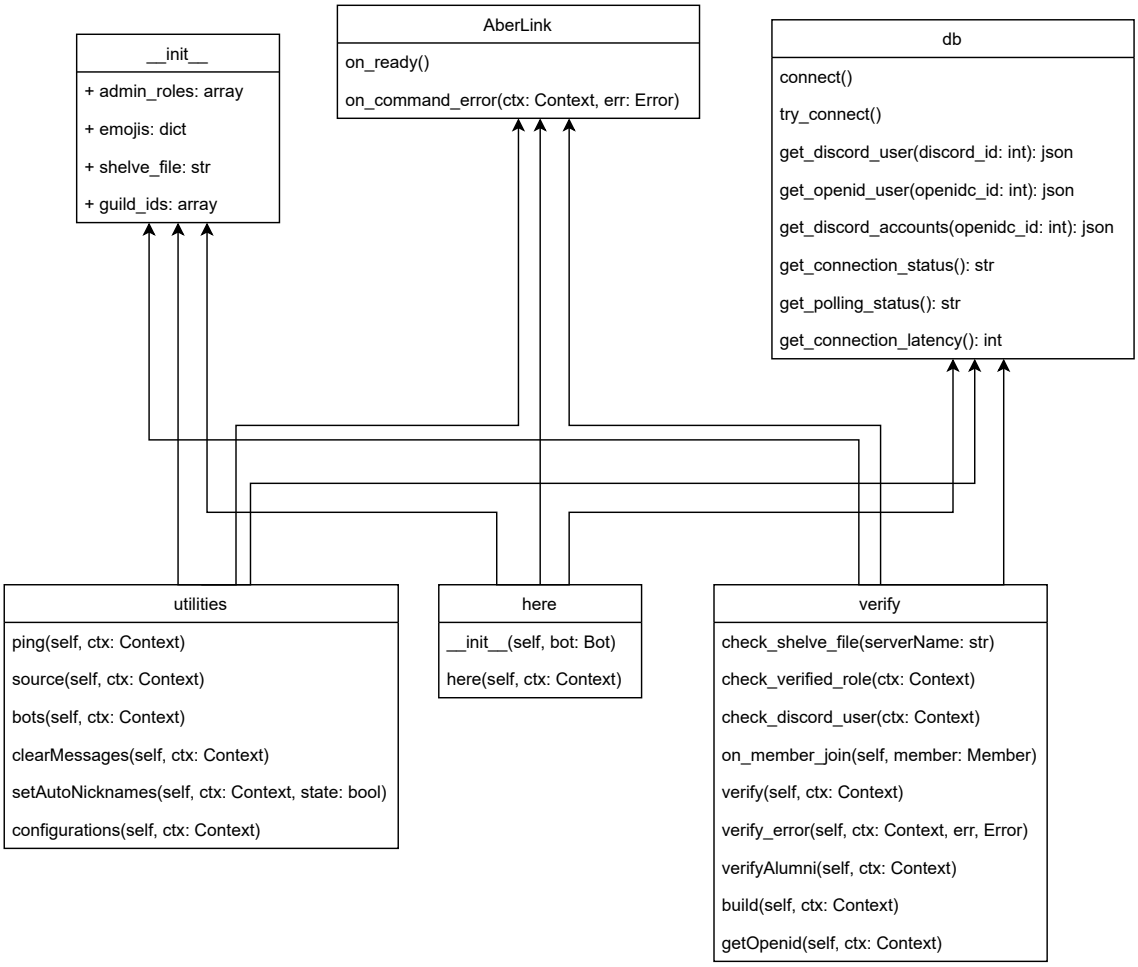

Figure 2.6: AberLinkDiscord UML diagram

2.3.1 Database Interaction

Below is a sequence diagram explaining how AberLink connects to the database and sends requests back and forth. It uses the Python library psycopg2 [21] to connect and send SQL requests back and forth; the details of which can be found in the db.py file. Once connected to the database if an error occurs while making a request then the code

executes a function (try_connection()) to reconnect to the database. Please see Appendix A for more details on psycopg2.

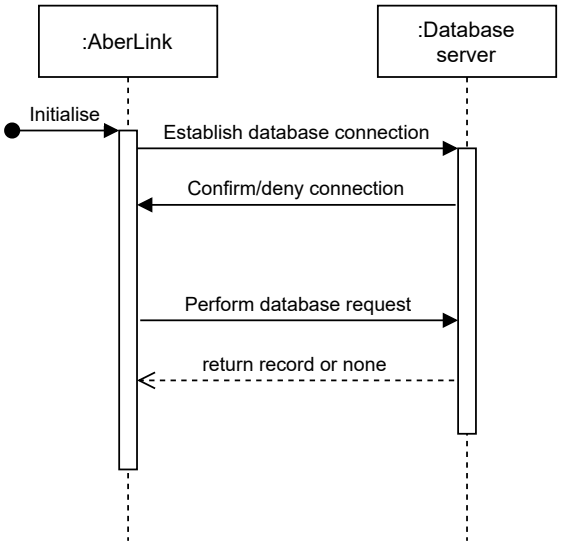

Figure 2.7: Sequence diagram for Discord bot to database

2.3.2 Discord Commands Interactions

Discord has two main methods for interacting with Discord bots. The first is by simply typing the command along with the prefix e.g. if the prefix is `!` and the command that's being invoked is called `help` then the user would type `!help`. This option has been around since the beginning of Discord bots and is a reliable and simple to use method. The only caveat is that the end user needs to remember what the commands are to use them as there is no autocompletion of commands.

The new option for for this system is to use 'slash commands' which are a new feature in Discord that allow the user to type a '/' in Discord which then displays a list of available commands that can be used. This feature is incredible and would be very useful to implement if it was not for the current lack of support. Discord.py [6] does not support the feature at all as it would require an entire library re-write so external libraries have emerged such as discord-py-slash-command [22]. These libraries however are currently very early in development and therefore subject to change and restructuring. As this project will be deployed after its creation then we want it to be robust and not break after a new update to the slash commands library. The Discord API for slash commands is also still in beta and new features are coming out all the time that drastically change it's implementation.

Despite its infancy, I have created a proof of concept. Below is a screenshot to show a working example of the AberLink source command appearing as a Discord '/' command.

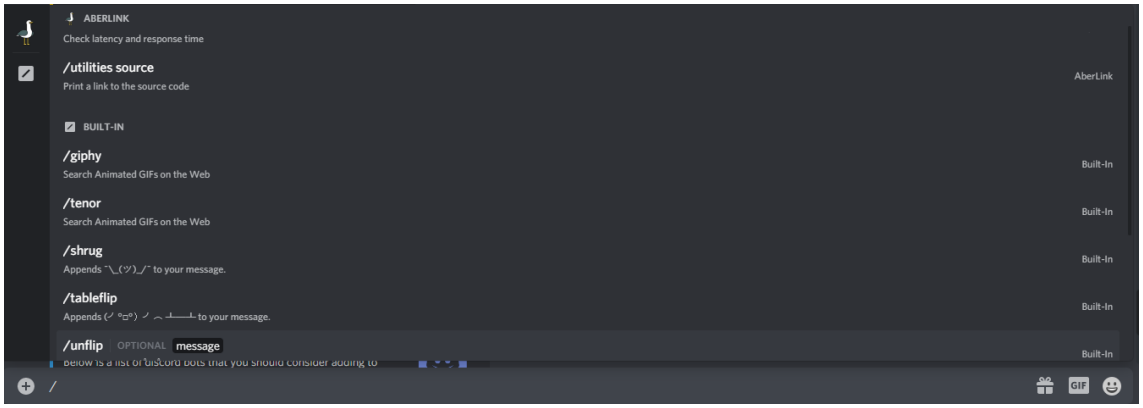

Figure 2.8: Sequence diagram for Discord bot to database

2.3.3 Complicated Behaviours

Most of the Discord bot code is relatively straight forward however there is one function in particular that needs some explanation called `build()` located in `AberLinkDiscord/cogs/verify.py`. This command is used to set up the server for authenticated users. This is done by giving authenticated users the verified role and all channels are made accessible only to members with that role. Users without that role (@everyone) are denied read permission. One special channel is visible to everyone to provide a place where they can interact with the bot to become verified. Below is a list of steps that the command executes.

1. Begins by simulating that the bot is typing giving the user visual feedback that the command is running as this function usually takes 500ms to 1s.
2. It then attempts to search for the @everyone role, verify role and #verify_channel so that it can update their permissions.
3. The @everyone role is then stripped of its permissions entirely as it is the only role that is given by default to new joining users. By removing all permissions the new user cannot view any channels.
4. It then gets the verify role and if it does not exist then it is created and given all the default permissions that @everyone used to have. This basically makes it so that users that have the verified role can do everything that the @everyone role can no longer do.
5. The bot is then given the verify role so that they can view the server channels and interact with members that have the new verified role.
6. If the #verify_channel does not exist then it is created and a message is pinned to the channel asking users to type !verify to verify their accounts.
7. The #verify_channel is then configured so that users with the @everyone role can view and type in the channel to get verified.

8. The #verify_channel then removes the ability for the verified role to see the channel and therefore sets up a barrier where @everyone and verified can no longer see eachother.

9. Finally once completed the bot stops typing and sends a message containing all the changes which have been made and the time spent making said changes.

2.4 Database design

Below is the Entity Relationship diagram that describes how the database works. The tables were created in Django which then remodels the psql [7] database to fit those requirements.

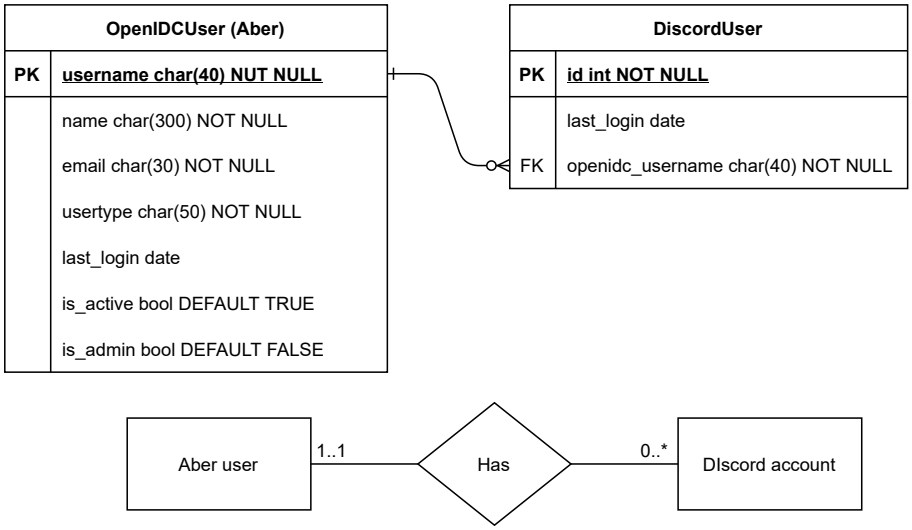

Figure 2.9: Entity relationship diagram for database

The **OpenIDCUser** is the main University account that the user authenticates with and is called so because it uses the OpenID Connect [5] system to authenticate users. The username, name and email are determined from the OpenID Connect response and are all unique. The usertype is also determined by the response and is used to determine if the user will be given administration privileges e.g. if usertype="staff" then admin=True or if usertype="student" then admin=False. Due to Django's custom user models it requires that the authenticated account has a is_active column that is kept true unless the account is marked as deactivated and cannot be used on the website.

The **DiscordUser** is simple and contains a unique Discord id that is a snowflake; a technology invented by twitter to keep id's unique. It also contains a last_login date to identify when the account was last authenticated with and lastly a openidc_id that is a foreign key from the OpenIDCUser's id.

2.5 User Interface

2.5.1 Website

This section includes some image mockups of the website. The website itself only has to perform the following tasks:

- Sign a user in and visually display that they are signed in (see top right of mockups and top centre screen text)
- Allow the user to link one or more discord accounts using the login with Discord button (see bottom middle of the mockup)
- Allow the user to delete their data from the database or unlink a Discord account (see bottom middle of the screen)
- Display admin pages for staff (see top right of mockups)
- About this project page and contact us (see top left)

Below are some mockups created before starting work on the website so as to have a reference guide for the navigation bar and how it looks when a user has 0, 1 or 2 Discord accounts connected. These mockups were created and using the design application Figma [23].

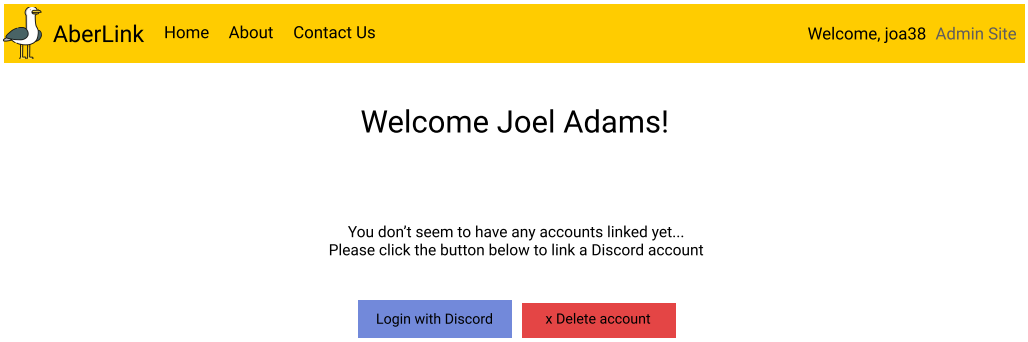

Figure 2.10: Website mock-up for 0 users

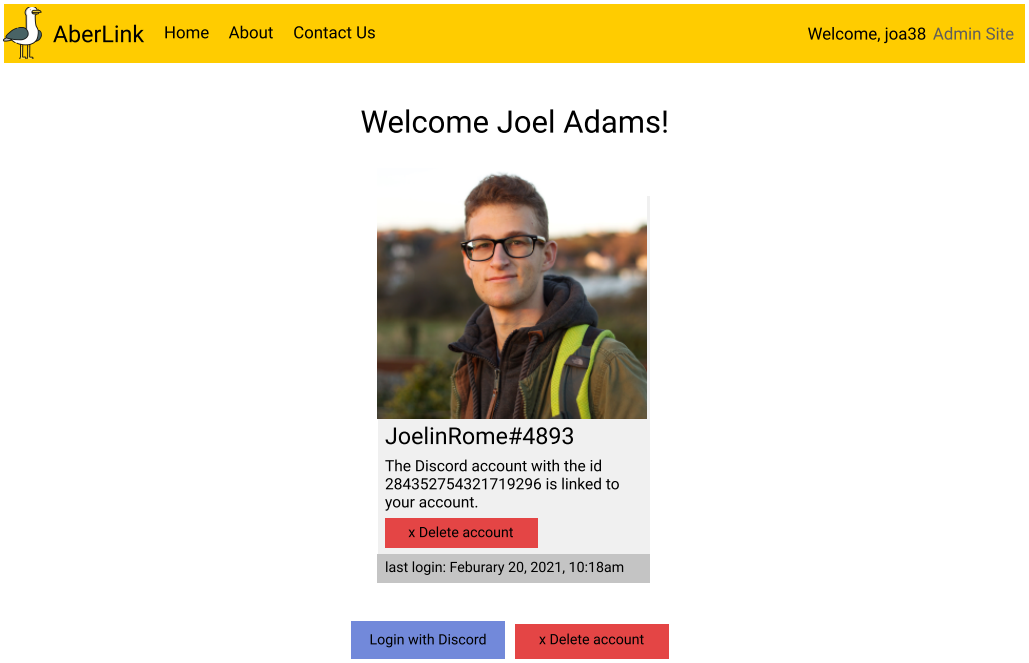

Figure 2.11: Website mock-up for 1 user

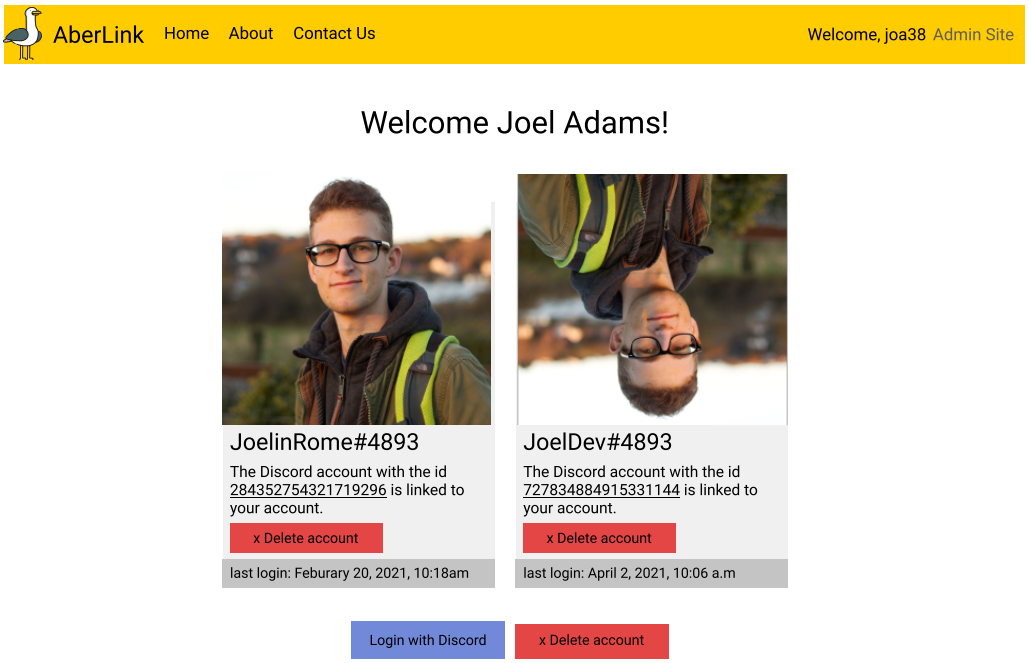

Figure 2.12: Website mock-up for 2 users

2.5.2 Discord bot

This section discusses the ways in which messages can be formatted in AberLink. Both Discord bots and users can send messages to Discord using the regular plaintext method (see 2.15), however, the bots have access to a few extra options for customizing messages before sending them. Bots have a feature to send messages using Embeds which contain far better formatting than regular messages and are used extensively throughout AberLink. These Embeds allow you to customize the side bar colour (see 2.13 and 2.14), attach thumbnails and images (see in top right of 2.13) and send nicely formatted content. Below are some examples of the output of bot commands that use Discord Embeds and for the second example 2.14 I have included a plaintext alternative 2.15 to help visually see the difference between these messages.

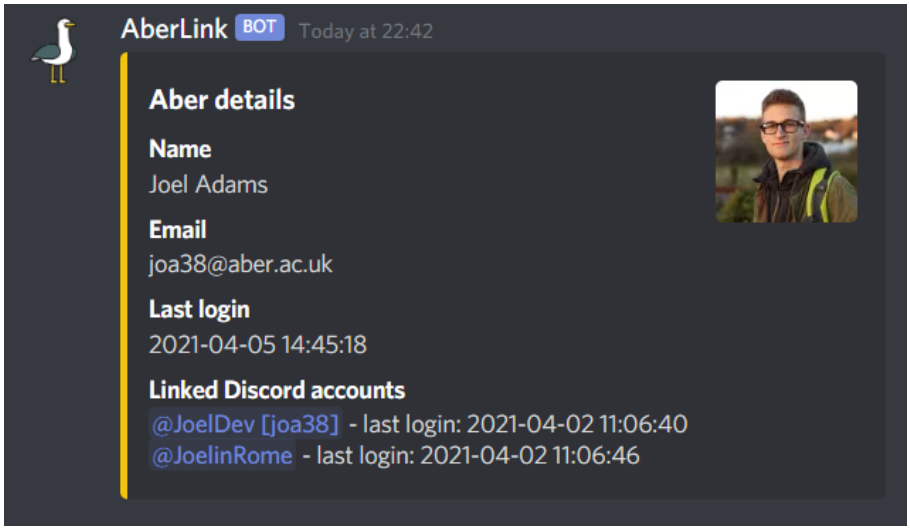

Figure 2.13: Discord Embed example for getting users information

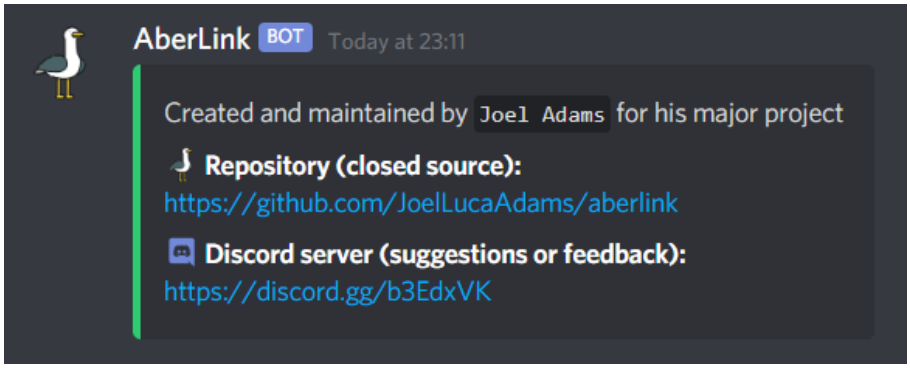

Figure 2.14: Discord Embed example for source command

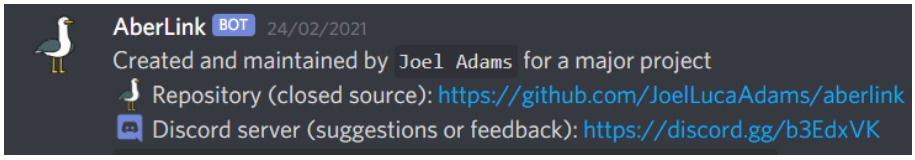

Figure 2.15: Discord plaintext example for source command

2.6 Version Control System and IDE

Over the last year of working in Python I have used JetBrains Python IDE PyCharm [24] to code my Python projects but recently switched to Visual Studio Code [25] due to the amount of extensions it supports and the versatility it has. It was chosen because it has extensions for querying a psql [7] database, compile Python programs, use virtual environments and write LaTeX documents like this one. It also allows you to remotely connect to a server or container over SSH which is very useful as the majority of the project was developed on a Linux (Debian 10 Buster [2]) container located in the University https://mmp-joa38.dcs.aber.ac.uk/. PyCharm does not support this feature and was one of the key features why it was ruled out for this project.

GitLab was chosen for the version control system using the instance provided by the University to store my code https://gitlab.dcs.aber.ac.uk/. Mirroring has also been setup so that the project is copied to another repository called aberlink on my primary GitHub account in case of emergencies or loss of data. Included below is an example of the commits that are made to the repository.

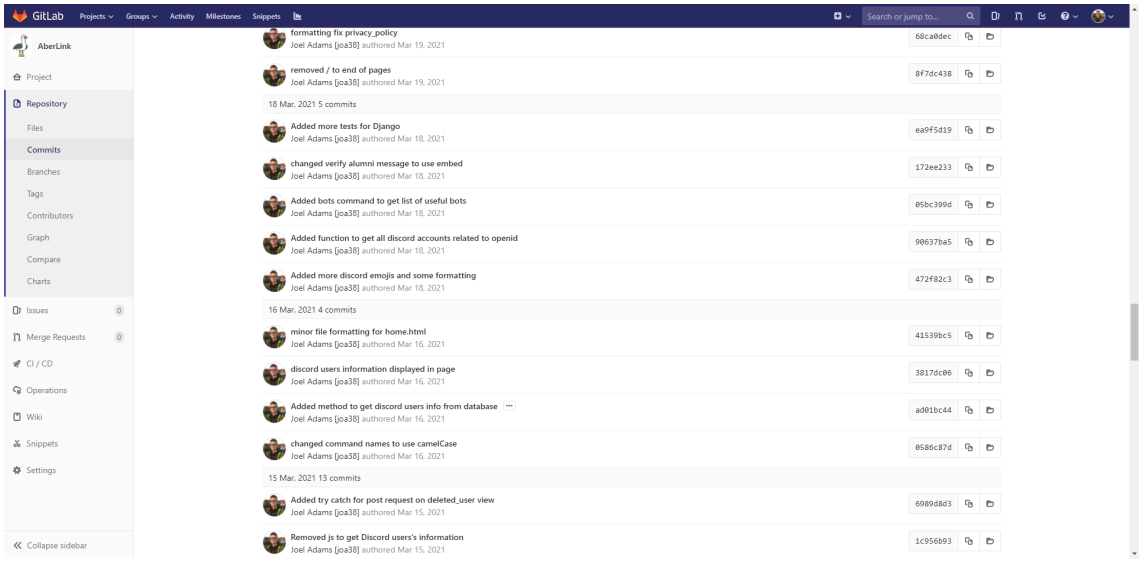

Figure 2.16: Example of GitLab repository commits

Chapter 3

Implementation

3.1 Code Implementation & Third-Party libraries

As seen below the Website and Discord bot have been split up into two separate sections because during implementation to help keep the systems separate. This helps with code maintainability, readability and allows the administrator who sets up this project to deploy the website and bot services on different containers or networks. For more information of third party libraries please see appendix A.

3.2 Website Building and Final Design

To create and run the website a set of libraries and Linux packages are used to perform certain tasks. Firstly, the Linux package Apache2 [3] is used as the web hosting framework got the website along with the library libapache2-mod-auth-openidc to reroute all incoming traffic to OpenID Connect [5] for user authentication. On top of these the linux tool certbot [19] is used to create a Let's Encrypt certificate for the website to enable HTTPS.

As discussed previously in this document in section 1.1.1 there were many libraries that were considered when deciding on the website framework. Django [4] was the framework of choice and it is open-source and free to use on personal projects. It is also very useful as it generates the majority of the code required to create and run a website so most of the code used will be picked up by the system for UAP. Apart from the generated template, a "Django Application" called login that contains the files and code which is used to run the website. I can confirm that the folder, login, is all my own code and should not fall under UAP.

The website pages also use a third party library called bootstrap [26] that is used to generate responsive mobile-first CSS for the website. The final design of the website ended up being very similar to that of the mockups made in section 2.5 and included below are some images from the final website.

Note: The navigation bars in the images below are smaller than the mockups due to

the change in screen size. These images were taken on a 27inch monitor whereas the mockups were made for a 15inch monitor.

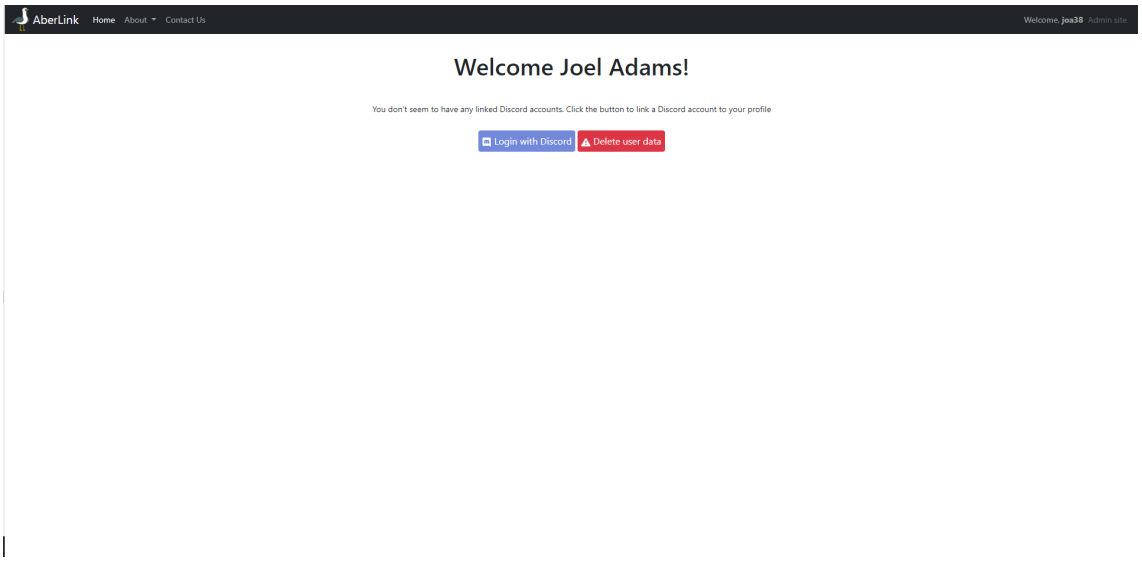

Figure 3.1: Final website with 0 linked Discord accounts
https://mmp-joa38.dcs.aber.ac.uk

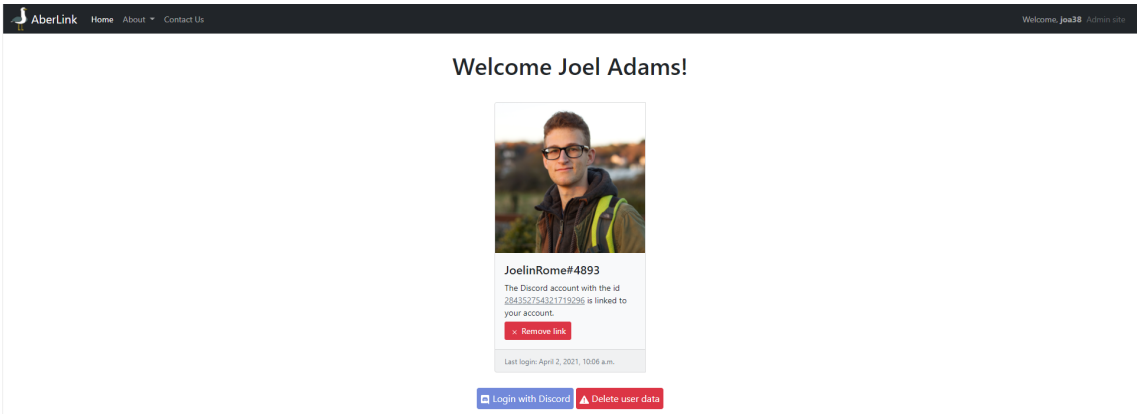

Figure 3.2: Final website with 1 linked Discord accounts
https://mmp-joa38.dcs.aber.ac.uk

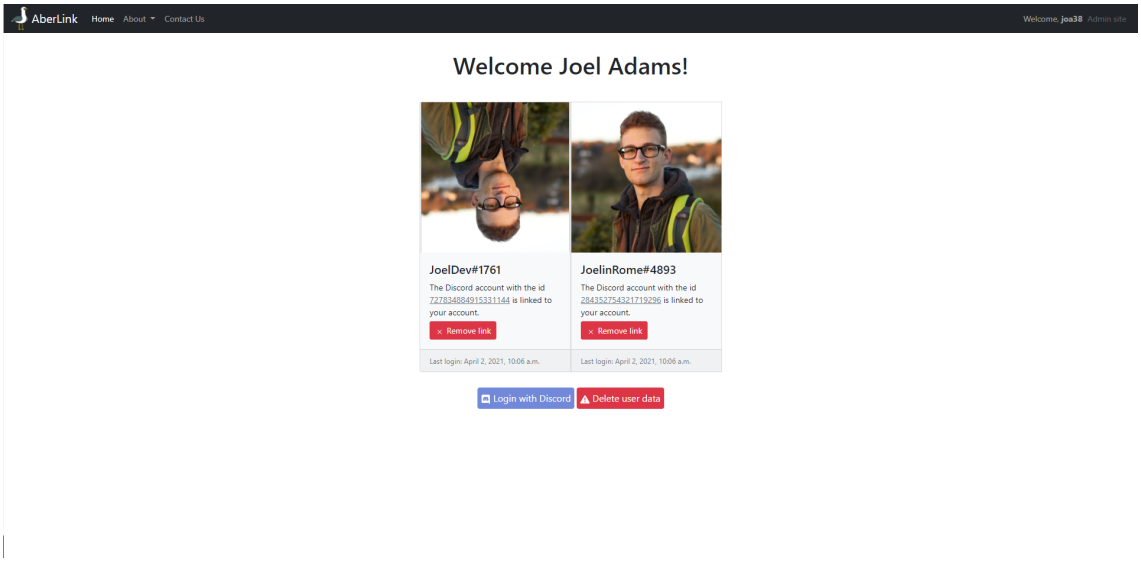

Figure 3.3: Final website with 2 linked Discord accounts
https://mmp-joa38.dcs.aber.ac.uk

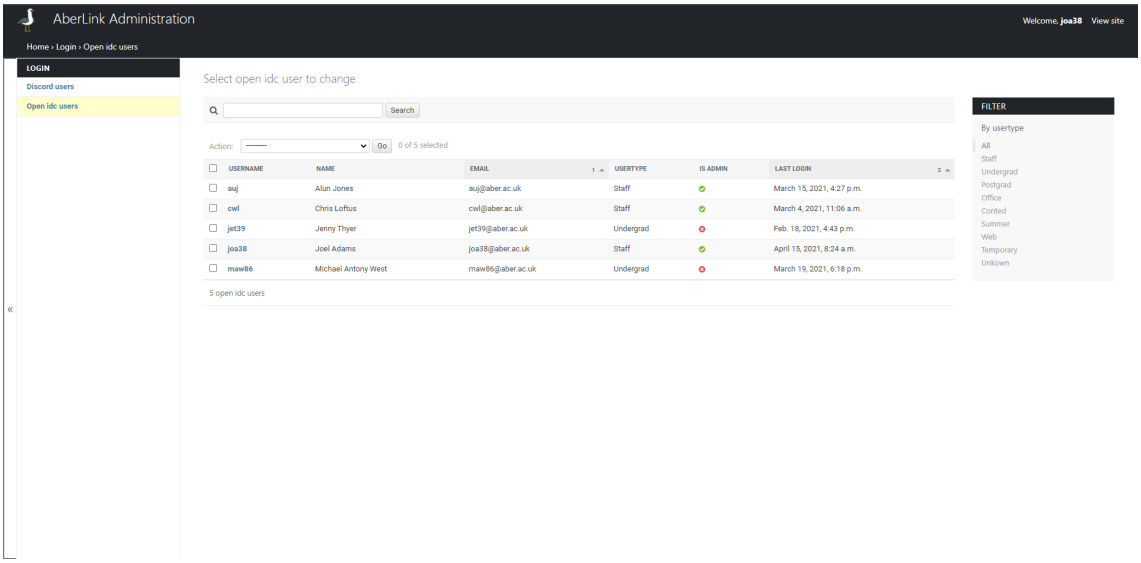

Figure 3.4: Final website Admin page for Aber accounts
https://mmp-joa38.dcs.aber.ac.uk/admin/login/openidcuser/

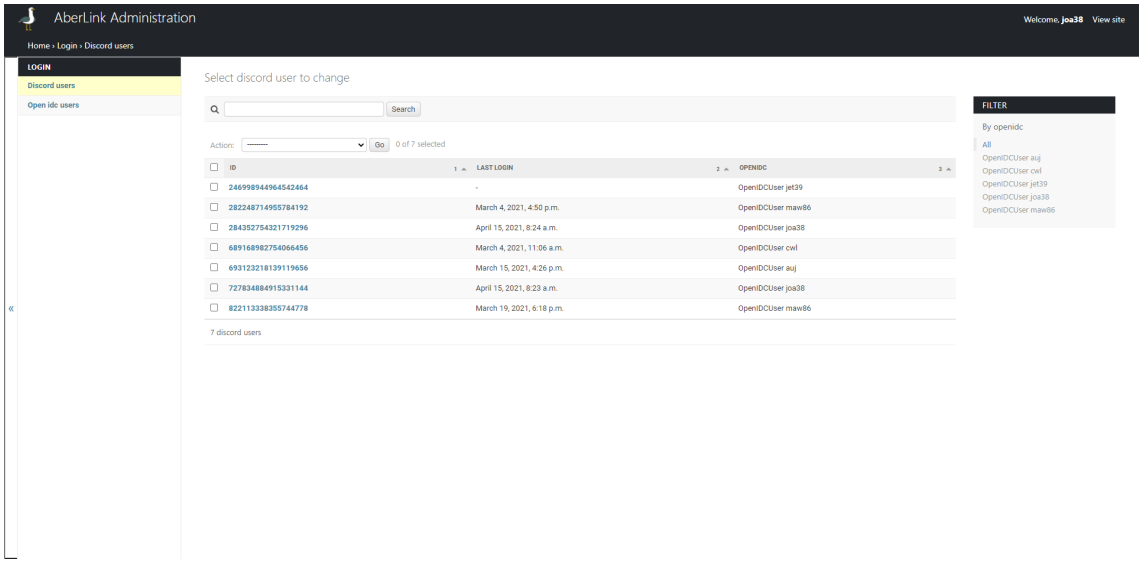

Figure 3.5: Final website Admin page for Discord accounts
https://mmp-joa38.dcs.aber.ac.uk/admin/login/discorduser/

Most of the users visiting this website will have not have a clue about this project or any of its policies so included below are the other two webpages created for this project. These pages display information about the project and privacy policy respectively.

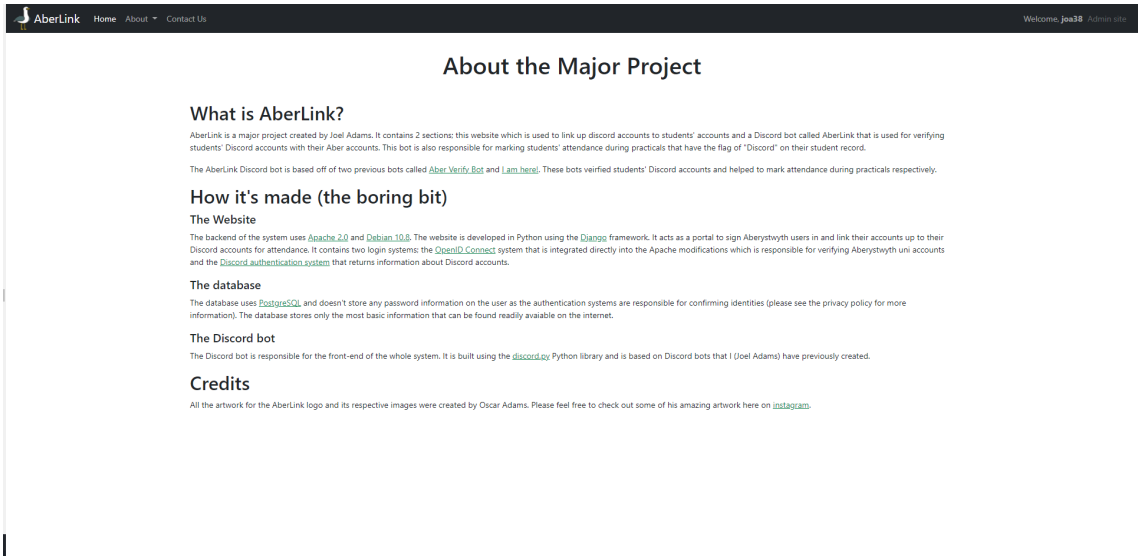

Figure 3.6: Webpage for information on what the project is about
https://mmp-joa38.dcs.aber.ac.uk/major-project

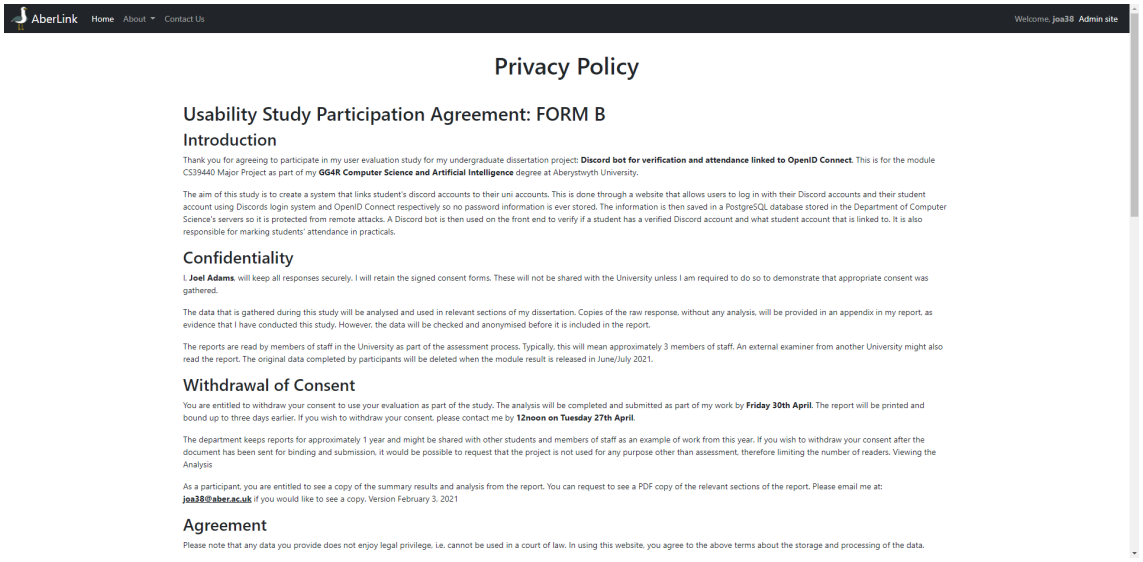

Figure 3.7: Webpage displaying agreement form and Ethics form https://mmp-joa38.dcs.aber.ac.uk/privacy-policy

3.2.1 Complicated Behaviours

3.2.1.1 Retrieving and Displaying Discord Information

The main complicated behaviour occurs when the user logs into the website and gets redirected to the homepage after they have authenticated one Discord account. When this happens the now logged in Discord account is linked to the OpenID Connect account and their information is saved to the database. As seen in table 3.2 the only information that gets saved to the database is their Discord ID and time of login, notice how their profile image or name are not saved to the database. This is because this information is subject to change and is not required to be stored in the database. This was done to ensure that the table is normalised to the 3rd normal form. It does however create an issue where the website does not know any information on the connected Discord account account apart from their ID.

To get around this a function is used that creates an API request to Discord querying information on the connected account. As seen below the function takes in a `QuerySet` that contains a list of connected Discord accounts and then makes an API request to the url `https://discord.com/api/v8/users/{discord_user.id}` with each Discord ID. After it has looped through all the connected accounts it returns an array of JSON objects that contain information on the connected accounts.

```python
def get_discord_users(discord_users: QuerySet):
"""
uses bot authorisation to get Discord users' info
Returns list of Discord users
"""
json_response = {}
for discord_user in discord_users:
  response = requests.get(f'https://discord.com/api/v8/users/{
   discord_user.id}',
  headers={'Authorization': f'Bot {config["DISCORD_TOKEN"]}'
  })
  user = response.json()
  user = {user["id"] : response.json()}
  json_response.update(user)
return json_response
```

Figure 3.8: Python function to make an API request about connected Discord account

The HTML webpage is then passed this content where it turns the JSON information into a JavaScript function and therefore making it a selectable document element. The document then loops over each Discord account and executes the following JavaScript function to extract information from the JSON object. Once it has extracted the information it replaces the profile picture and name of connected Discord account in the document.

```javascript
let json_data = JSON.parse(document.getElementById("
   discord_user_info").textContent)
let discordAvatar = json_data["{{user.id}}"].avatar
let discordUsername = json_data["{{user.id}}"].username + "#" +
   json_data["{{user.id}}"].discriminator
document.getElementById("discordAvatar{{forloop.counter}}").src
   = `https://cdn.discordapp.com/avatars/{{user.id}}/${
   discordAvatar}.png?size=256`
document.getElementById("discordUsername{{forloop.counter}}").
   textContent = `${discordUsername}`
```

Figure 3.9: JavaScript script to extract information from the Discord users JSON object

3.2.1.2 Deleting Account or Unlinking Discord accounts

This project respects the GDPR regulations and gives users the right to delete their data from the database or unlink a Discord account. This feature was simple to implement but became difficult when I wanted to include popups that would appear when the user tried to delete their account as pictured below. The difficult part about this comes when trying to get information on which account is going to be deleted and ensuring that the correct one is selected.

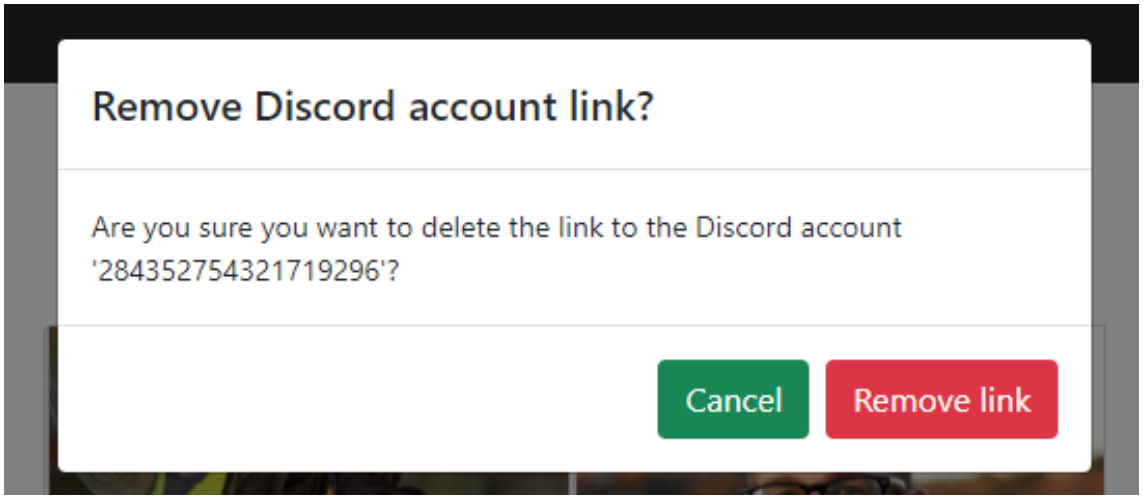

Figure 3.10: Example of a popup that appears when a user tried to unlink a Discord account

Implementing this feature was relatively straight forward however getting it to display information on the specific account that would be unlinked was harder than expected. Getting the HTML document to parse information on which button had been pressed in the document was as simple as adding an extra attribute to the button. As seen in the figure below the button contains the attribute `value` that is assigned to the Discord account.

```
1 <button type="button" id="discordDeleteButton" class="btn btn-
    danger btn-sm" data-bs-toggle="modal"
2   data-bs-target="#discordDeleteButtonModal" value="{{ user.id
    }}" style="margin-top: 5px;">
```

Figure 3.11: HTML button to delete a Discord account containing value attribute with the Discord ID

When the button is pressed it calls a JavaScript script and displays the popup/modal. It then uses the function `event.relatedTarget.getAttribute` to get the information from the button on the account that is linked to it.

```
let deleteDiscordAccountModal = document.getElementById('
    discordDeleteButtonModal')
deleteDiscordAccountModal.addEventListener('show.bs.modal',
    function (event) {
  // Gets the button and it's data attribute and prints it out
    in the .modal-body
  let recipient = event.relatedTarget.getAttribute('value')
  deleteDiscordAccountModal.querySelector('.modal-body').
    textContent = 'Are you sure you want to delete the link to
    the Discord account \'' + recipient + '\'?'
  document.getElementById('discordIdText').value = recipient
})
```

Figure 3.12: JavaScript script to get context on the button that has been pressed and edit the popup to include this information

Once the confirmation button in 3.10 is pressed the webpage will send a form containing the Discord id via a post request to the same webpage. The Python function for this webpage then detects that a post request has been made and it attempts to find the linked account and remove it from the database.

```
def openidc_response(request):
  ...
  if request.method == 'POST':
  try:
    discord_id = request.POST.get("discord_id")
    DiscordUser.objects.filter(id=discord_id).delete()
  except KeyError:
    pass
  ...
```

Figure 3.13: Extract from Django Python function to display the home page and handle post requests to remove Discord accounts

3.3 Discord Bot

The Discord bots (AberLink) implementation uses many of the design principles discussed in section 2.3 of this document.

3.3.1 Setup and Configuration

The bot contains a main file called AberLink.py that configures the bot giving it all the required permissions/settings to run. It loads the sensitive credentials from a .env file

using the Python library dotenv [27]. This file contains key-value pairs that hold the Discord bot token that is accessible from the Discord developer portal `https://discord.com/developers/applications` and information required for the bot to connect to the database.

```
# load the private discord token from .env file.
load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')
WEBSITE = os.getenv('WEBSITE_URL')

# Initialise the database connection
PostgreSQL.connect()

# Initialise the Bot object with an accessible help Command
   object
helpCommand = DefaultHelpCommand()

bot = commands.Bot(
    command_prefix="!",
    help_command=helpCommand,
    intents=discord.Intents.all(),
)

bot.run(TOKEN)
```

Figure 3.14: Snippet of code from AberLink.py main Discord bot file

As seen in the figure above the key value pair for the `DISCORD_TOKEN` is loaded in from the file and then used at the end of the code extract to run the bot. In the middle we see an important section called `commands.Bot` that configures the bot with key information such as which prefix to use to call commands. E.g. if the prefix is set to `!` then a command is invoked using `!<command_name>`, by changing the prefix to `!test` then the command is invoked using `!test<command_name>`. As seen above the prefix is not limited to a single character so it can be as long as needed.

3.3.2 Discord Command Example

The bot creates commands that can be used in a Discord server using the Discord.py [6] library. Below is an example of the ping command that is used as a way to check that the bot is alive. Discord.py turns this Python function into a Discord command when the decorator `@commands.command(aliases=['p'])` is used where the `aliases=['p']` parameter indicates an alias that can be used to call the command. The function then starts a timer using time() and stops after the first message is sent, then the message is edited to include the latency taken to edit and modify the message. Included in this response is also information on the database such as the latency, polling status and response time.

```python
@commands.command(aliases=['p'])
async def ping(self, ctx: Context):
  """
  Returns latency and response time of Discord and the database
  """
  start_time = time()
  message = await ctx.send(f' pong `DWSP latency: {str(round(ctx.bot.latency * 1000))}ms`')
  end_time = time()
  db_latency = PostgreSQL.get_connection_latency()
  db_poll = PostgreSQL.get_polling_status()
  await message.edit(content=f' pong \n{emojis["discord"]} `DWSP latency: {str(round(ctx.bot.latency * 1000))}ms` ' +
    f'`Response time: {str(int((end_time - start_time) * 1000))}ms` \n' +
    f'{emojis["aberlink_database"]} `Database Polling status: {db_poll}` `Database latency: {db_latency}ms`')
```

Figure 3.15: Example of AberLink's ping command code

The commands response is then sent to the Discord channel with a response that looks similar to the one pictured below.

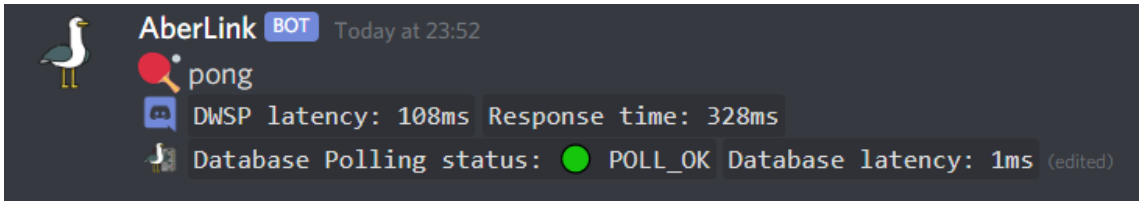

Figure 3.16: Example of ping command in Discord

3.3.3 How do I access a list of the commands in AberLink?

The Discord.py library contains a useful function called `DefaultHelpCommand()` that will generate an automatic list of commands available in your program by analysing each function that has the command decorator `@commands.command()`. The list of commands is then accessible through Discord using the command prefix plus the keyword help which in this case would be !help. Below is an example of the current output from the help command at time of this screenshot.

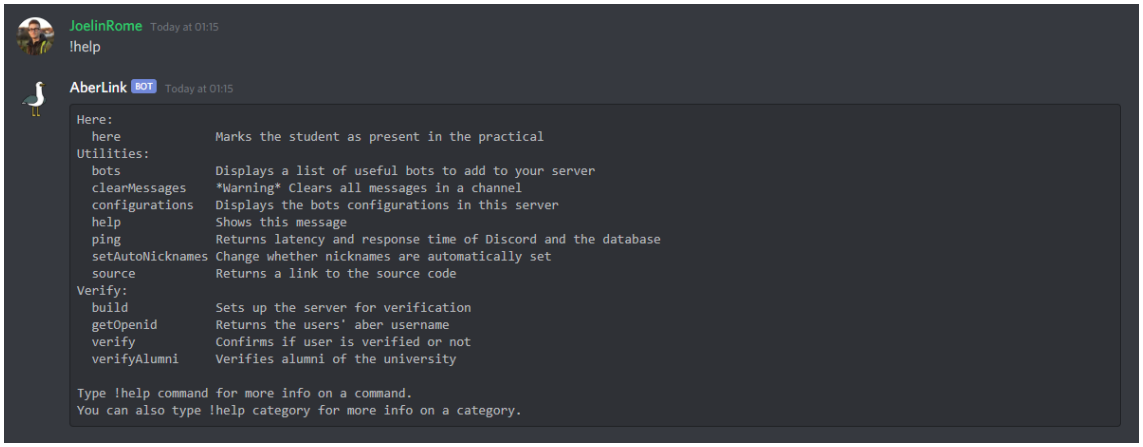

Figure 3.17: Example of Discord help command to display list of Discord commands for AberLink

3.3.4 Database

The Discord bot uses the Python library psycopg2 [21] to connect remotely to the PostgreSQL [7] database using a file located inside of the `AberLinkDiscord/cogs` folder called `db.py`.

```
CONN = 0

def connect():
"""
Connects or reconnects to database
"""
load_dotenv(find_dotenv())
DB_NAME = os.getenv('DATABASE_NAME')
DB_USER = os.getenv('USER')
DB_PASSWORD = os.getenv('PASSWORD')
DB_HOST = os.getenv('HOST')
DB_PORT = os.getenv('PORT')

try:
  global CONN
  CONN = psycopg2.connect(database=DB_NAME, user=DB_USER,
   password=DB_PASSWORD, host=DB_HOST, port=DB_PORT)
  print(f'Reconnected to PSQL database: {CONN}')
except psycopg2.OperationalError as err:
  print(f'Error connecting to database. Error: {err}')
  raise
```

Figure 3.18: Extract of the connection function to the PostgreSQL database from the AberLink Discord bot

As seen in the code extract above it uses the same functions as described in section 3.3.1 for loading data from the `.env`. Once it has loaded in the correct key value pairs it then attempts to connect to the database using the function `connect()` and print the output to the command line. If this is not the case e.g. database is down then then the try except statement catches the error and prints it out to the command line. If the database connection is successful however the information about that connection is then stored in a global variable called `CONN` which stands for connection.

To ensure that the connection will always be restored if there is a catastrophic failure a function was added that attempts to get a cursor on the database and if it fails then it calls the function above in figure 3.18. This function `try_connection()` is called by any piece of code that relies on extracting information from the database to ensure that there will never be a database error.

```
1 def try_connection():
2   """
3   Attempts to get a cursor from the database otherwise it
     restarts the database connection
4   """
5   try:
6     cur = CONN.cursor()
7   except psycopg2.InterfaceError:
8     PostgreSQL.connect()
```

Figure 3.19: Extract of the try connection function for connecting to the database from the Discord bot

Once a connection is established with the database and connection issues are handled then SQL queries can be made on the database to get information. In Discord there are only really two functions that are required to read information on a student; a function to lookup a students Discord account using their Discord ID and a function to lookup a students Aber OpenID Connect information using their username. Both of these functions are relatively similar so only one has been included to illustrate how this works.

```
def get_discord_user(discord_id: int):
"""
Returns a discord user if they exist or None
"""
PostgreSQL.try_connection()
cur = CONN.cursor()

cur.execute(f"SELECT * FROM login_discorduser WHERE id={
    discord_id}")
row = cur.fetchone()
if row is not None:
  return {"id": row[0], "last_login": row[1], "openidc_id": row
    [2]}
```

Figure 3.20: SQL function used to get information on a Discord user from the database in the Discord bot

As seen above a parameter of the Discord ID is passed to the function which then gets a database connection using the `try_connect()` as previously described in the last section and executes an SQL statement on the database. The first record is then fetched from the database and is then returned as a JSON object to allow for it to be easily accessed. If there are no records that match then None is returned.

3.3.5 Verification

This command is used to verify users accounts. It begins by trying to find the verified role on the server, then using the function above in figure 3.20 to get the users information. The function then checks that the user exists and then uses another SQL lookup to get information on the OpenID Connect [5] Aber account. Once these requests have been completed the user is given the role of `verified` followed by the message ”You are now verified with AberLink”. If the feature discussed in section 3.3.8 to allow automatic nicknames is enabled then the bot will also change the users nickname.

```
@commands.bot_has_permissions(manage_nicknames=True,
    manage_roles=True)
@commands.command(aliases=['v'])
async def verify(self, ctx: Context):
  """
  Confirms if user is verified or not
  """
  verified = await check_verify_role(ctx)
  db_discord_user = await check_discord_user(ctx)
  if verified is None or db_discord_user is None:
    return
  db_openid_user = PostgreSQL.get_openid_user(db_discord_user["
   openidc_id"])
  email = db_openid_user["username"]
  user = ctx.message.author
  await user.add_roles(verified, reason='Assigning user the
   verified role')
  await ctx.send("You are now verified with AberLink")

  if check_shelve_file(ctx.guild):
        await user.edit(nick=f'{user.name} [{email}]', reason="
   Changing users\'s nickname")
```

Figure 3.21: Discord command to verify students

3.3.6 Attendance

Attendance is recorded using the Discord function `here`. Once the command has been typed then the Discord bot will delete the message and open a DM or Direct Message with the user. It then sends a query to the database and if the user exists then a query is made to retrieve their aber account. After this is completed a request is made to the API endpoint along with the data object `{'username': <aber_email>}` which updates student record. The response from this request is then evaluated and an appropriate response is sent via the DM channel that was opened earlier.

```
@commands.command(aliases=['h'])
async def here(self, ctx: Context):
  """
  Marks the student as present in the practical
  """
  await ctx.message.delete()
  dm_channel = await ctx.author.create_dm()
  discord_user = PostgreSQL.get_discord_user(ctx.author.id)
  url = 'https://integration.aber.ac.uk/joa38/submit.php'

  # Gets openid user from Discord id
  if discord_user is None:
    await dm_channel.send(f'You have not been verified yet.
   Please visit {WEBSITE} to get verified')
    return
  openid_user = PostgreSQL.get_openid_user(discord_user["
   openidc_id"])

  # getting request from API endpoint
  data = {'username': openid_user["username"]}
  api_response = requests.post(url, json=data).json()

  # If evaluates to true send user message with timestamp
  if eval(api_response["status_updated"].title()):
    current_time = datetime.now()
    current_time = current_time.strftime("%d/%m/%Y %H:%M:%S")
    await dm_channel.send(f'Your attendance in the server `{ctx.
   guild.name}` has been recorded for the module: `{api_response
   ["module_code"]}` with timestamp: `{current_time}`')
  else:
    await dm_channel.send('Your attendance has not been recorded
    for this practical. If you believe this is incorrect please
   contact your module coordinator.')
```

Figure 3.22: Discord command to mark students attendance

3.3.7 Error handling

Discord.py includes a decorator called `@Bot.event()` that goes inside of the AberLink.py file for dealing with bot events. This paired with a function called `on_command_error()` deals with error messages. Below is an example of some of the if statements that deal with bot errors. The final else if command to check if the command is not found is extremely important as it tells the bot to ignore any commands that does not exist in the case where two Discord bots use the same command prefix but have different commands.

```
if isinstance(error, commands.errors.CheckFailure):
  await ctx.send(error)
elif isinstance(error, commands.errors.MissingRequiredArgument):
  await ctx.send('You are missing a required argument.')
elif isinstance(error, commands.errors.CommandNotFound):
  pass
```

Figure 3.23: Example of AberLink's on_command_error() checking statements

3.3.8 Server Configurations

AberLink contains a feature called autoSetNickname that will automatically assign nicknames to users when they join or verify on the server by appending their aber username to the end of their name. e.g. JoelinRome#4893 becomes JoelinRome [joa38]. This was added to help lecturers easily identify which student is which and if the student is uncomfortable with this they can always manually change it back.

Some lecturers might not enjoy this feature and prefer to turn it off so included is a micro database using the python module [28] that creates a file stored in the same directory as the Python file. This basically creates a file that can be accessed at any time and acts as persistent storage. It contains key value pairs e.g. `{server_id : true}` that store a boolean on whether the automatic naming feature is enabled or disabled.

3.4 Database

The Database implementation was relatively straight forward as Django [4] generates all of the required tables for the project to function once you define the models. I've included the model used to generate OpenID Connect [5] aber user model below. In this figure you can see that there are two classes; OpenIDCUserManager to create a new database object and OpenIDCUser to make the database model.

The OpenIDCUserManager class has three main parameters; one to call itself, a user which is a JSON object and password which is defaulted to None as no password data is stored. The information from the user object is passed throughout the code and is used to decide if the user should have admin permissions. The user is then saved to the database.

The OpenIDCUser class contains a nested class called usertypes that is a list of all the possible choices for incoming usertypes from user authentication. Further down we can see that there is a set of definitions for the database model such as the id, username, name, etc. Note however that the usertype definition uses the choices class to define what role a user may have. As this model is the main model used to authenticate users on the website Django requires that the model contains a few special items including a password which I have set to None and the arrays at the bottom called USERNAME_FIELD and REQUIRED_FIELDS.

```
class OpenIDCUserManager(BaseUserManager):
    def create_user(self, user, password=None):
        new_user = self.model(
            username = user['OIDC_CLAIM_preferred_username'],
            name = user['OIDC_CLAIM_name'],
            email = user['OIDC_CLAIM_email'],
            usertype = user['OIDC_CLAIM_usertype']
        )
        if user['OIDC_CLAIM_usertype'] == "staff":
            new_user.is_admin = True
        new_user.save(using=self._db)
        return new_user

class OpenIDCUser(AbstractBaseUser):
    objects = OpenIDCUserManager()

    class usertypes(models.TextChoices):
        STAFF = 'staff'
        UNDERGRAD = 'undergrad'
        POSTGRAD = 'postgrad'
        OFFICE = 'office'
        CONTED = 'conted'
        SUMMER = 'summer'
        WEB = 'web'
        TEMPORARY = 'temporary'
        UNKNOWN = 'unknown'

    id = models.AutoField(auto_created=True, primary_key=True,
   serialize=False)
    username = models.CharField(max_length=40)
    name = models.CharField(max_length=300)
    email = models.CharField(max_length=30)
    usertype = models.CharField(max_length=50, choices=usertypes
   .choices)
    last_login = models.DateTimeField(null=True)
    password = None
    is_active = models.BooleanField(default=True)
    is_admin = models.BooleanField(default=False)

    USERNAME_FIELD = 'id'
    REQUIRED_FIELDS = ['username', 'name', 'email', 'usertype']
```

Figure 3.24: Django database model of OpenID Connect Aber users

The tables generated for the database are pictured below and they do differ quite drastically from the tables discussed during the design section 2.4 of this document. This is

because I was not originally accounting for the tables that get automatically generated by Django [4] that are required for the application to run. Below the figure is information to explain what the Django generated tables mean.

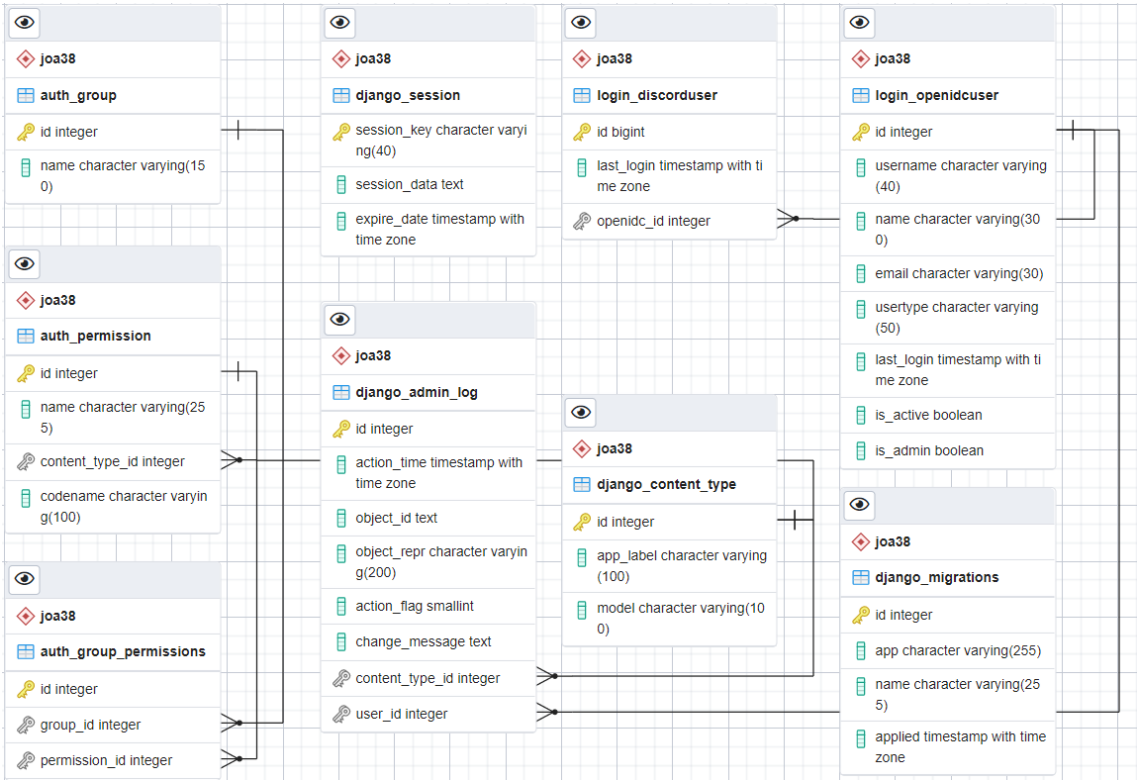

Figure 3.25: Final Entity Relationship diagram for database

- **User tables** - These two tables are the ones that have been discussed previously in this document for storing user data.
  - **login_openidcuser** - This table stores the OpenID Connect [5] information and is used as the authenticated user account.
  - **login_discorduser** - This table stores information on the discord user and contains a foreign key relationship with the table above.
- **Django tables** - Tables generated by Django [4]
  - **django_migrations** - This table contains the history of the changes made to the database using Django. It acts as a way to revert to previous versions of the database in the case of errors.
  - **django_session** - Stores sessions on the currently logged in users.
  - **django_content_type** - Stores information on all available models in the database.
  - **django_admin_log** - Stores history of logins for administrative users.
  - **auth_group & auth_group_permissions & auth_permissions** - These tables are part of the backend for the authentication of users.

Below are examples of the records that are stored in the database tables.

Note: The column last_login in the table 3.1 contains the word datetime but should have a datetime object as seen in table 3.2. It has been removed so that the table fits nicely in this document.

id	username	name	email	usertype	last_login	is_active	is_admin
1	joa38	Joel Adams	joa38@aber.ac.uk	staff	datetime	t	t
2	jet39	Jenny Thyer	jet39@aber.ac.uk	student	datetime	t	f
3	maw86	Michael Antony West	maw86@aber.ac.uk	student	datetime	t	f

Table 3.1: Aberystwyth user table example

id	last_login	openidc_id*
727834884915331144	2021-02-18 16:43:47.067328+00	1
284352754321719296	2021-02-18 16:43:47.067328+00	1
246998944964542464	2021-02-04 11:14:40.057891+00	2
282248714955784192	2021-02-12 17:35:23.044226+00	3

Table 3.2: Discord user table example

3.5 Configuration and Setup

After completion of this project there is a plan to setup this service permanently for the department so a folder was created called `config` containing information on how to set up this project. This folder contains a `README.md` file located in Appendix D that explains how and what is needed to be configured to get the system up and running properly. it also contains a bash script called `setup.sh` that installs and the relevant dependencies and sets up the virtual environments for the projects. Also included are a few example configuration files for Apache2 [3], Django [4], OpenID Connect authentication [5] and Discord.py [6].

3.6 Unforeseen Issues

An issue that was encountered was with creating the custom user model in Django that would have been used to model the database. The documentation and videos found online about implementing user models were rather cryptic and difficult to understand, however after much research a video explaining how to implement a good custom user model was found that resolved the issues. Once this was completed I realised that Django is not happy with modelling the primary key of a table using a char so switched over to using an int. This turned out to be a good idea as Aberystwyth University tends to recycle old emails so over the next 5 years there could be an issue where the database would try and create a new entry in the database with the same email.

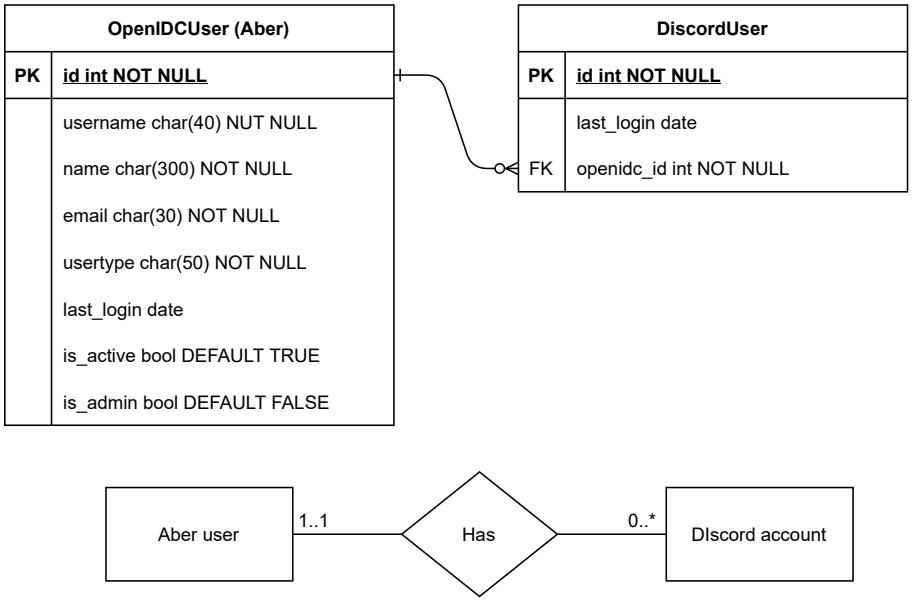

Figure 3.26: Updated Entity relationship diagram for database

3.7 Review

3.7.1 Review Against Planned Requirements

Most of the planned requirements have not changed during implementation however some of the promised tasks have only been partially fulfilled. Please see Appendix C.1 for a table of the functional requirements specified in 1.4. For more information on requirements that did not pass please see 4.5.

In the final section of section 1.2.1 **Further potential work** I have decided against integrating DemoHelper into AberLink as this would greatly increase its complexity and make it much more difficult to maintain. Welsh language support has also not been implemented as I do not speak the language nor understand it well enough.

3.7.2 Review Against Project Process

Below is included a table describing the iterations and what was completed each one. These iterations were based on the list of objectives created in section 1.2.1 and span over the course of two months. For more detail please see the wordpress blog here https://cs39440blog.wordpress.com/.

Iteration	Time taken (days)	What happened?
1	5	Research into services required to run the project and basic setup of University container. Lots of meetings to discuss what data can be used/stored and began setup of API endpoint for attendance.
2	5	Setup HTTPS on container, installed Django [4] and created basic website using it. Linked up Django to database and created basic Discord login system.
3	5	Added OpenID Connect [5] to website for aber user login. Created basic models to link OpenID accounts to Discord accounts in PostgreSQL [7].
4	7	Created Admin pages and implemented proper database model for Discord and Aber users
5	5	Created Discord bot skeleton and added support for it to connect to the database.
6	5	Added attendance and verification to Discord bot.
7	5	Added content to main webpage and option to delete a Discord account or all data. Added custom error message webpages for 400, 403, 404 and 500.
8	5	Updated main user page so that it queries Discord API to get profile picture and username and update webpage with details. Added more features to Discord bot to interact with database.
9	5	Created Configuration folder for installing the software along with bash script to install dependencies.
10	5	Added Discord bot functionality to add configurations to a server and feature to automatically change Discord users nickname to append aber email.
11	16	Working on LaTeX document for project report.

Table 3.3: Project iterations and what occurred

Included is a graph of GitLab commits over time below to reflect the tables results as described above.

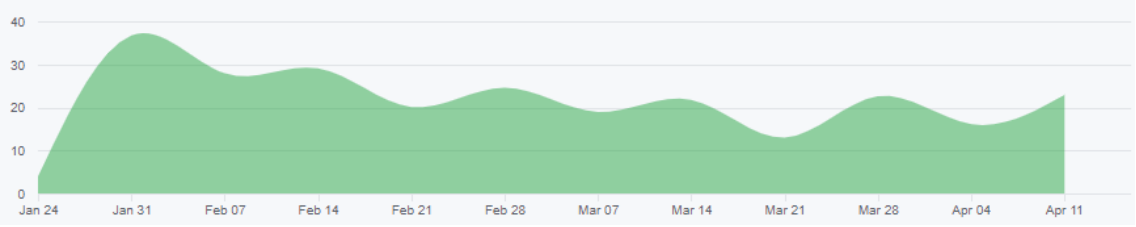

Figure 3.27: GitLab graph of commits over time

Chapter 4

Testing

As previously mentioned in section 1.3 a Feature Driven Development (FDD) approach was used for this project, working in one week iteration windows to build and test the software. This included all of the testing sections mentioned below apart from automated testing which was completed at the end of the project instead.

The Discord.py Python framework does not contain any unit testing and there are no third party libraries which sufficiently perform this job. For the Django framework however there are some unit tests built in so they have been used in this project and are detailed further on in section 4.2.1.1. These Django tests are used to cover both the website and the database as the website cannot be tested by itself.

4.1 Sample Data

Sample data was considered for this project however it would be very difficult to implement because the OpenID Connect [5] system used to authenticate and log users in relies on a proper database to pull user data from. The University was not comfortable with creating and supplying fake user accounts that would authenticate with this system as the database where these would be used is currently live. This would mean that they could be stolen and used to access live systems such as Student Record or blackboard. The alternative to this would mean that a mock database would be needed which would take up a considerable work to create. The workaround for this system is far simpler and involved asking both staff and students to try out the system and test it until it breaks. The outcomes of this are detailed in section 4.4.

4.2 Automated Testing

Automated testing has been difficult for this project as mentioned above. Originally there was a plan to include automated testing by using the DevOps frameworks in Git, however as the universities GitLab instance was used then there was no access to dedicated

pipelines/kubernite clusters to build and run the code.

4.2.1 Unit Tests

4.2.1.1 Website & Database

Django provides some useful documentation (https://docs.djangoproject.com/en/3.1/topics/testing/) for creating unit tests for both Django and the database/user models. These tests are useful for checking that Discord accounts and OpenID Connect [5] Aber accounts link up properly and that webpage urls are correct and render the appropriate content. All of the following tests can be found in the file located in this path `src\AberLinkAuthentication\login\tests.py`.

Below are two testing classes; one called TestUrls and one called TestModels. These are responsible for testing that the urls are correctly found and that adding new users to the database works as intended. Included below is an example of each test:

```
def test_url_discord_redirect(self):
    url = reverse('Discord-response')
    self.assertEquals(resolve(url).func, views.
   discord_oauth2_redirect)
```

Figure 4.1: Django URL render test

As seen above the test function simply gets the name of the url 'Discord-response' and then checks that it returns the correct Python function (view) to render that page

```
def test_model_openidc_user_staff(self):
    self.openidc_user2 = OpenIDCUser.objects.create(
        username="abc123",
        name="Bob Ross",
        email="abc123@aber.ac.uk",
        usertype="staff"
    )
    self.assertEquals(self.openidc_user2.username, "abc123")
    self.assertEquals(self.openidc_user2.name, "Bob Ross")
    self.assertEquals(self.openidc_user2.email,
    "abc123@aber.ac.uk")
    self.assertEquals(self.openidc_user2.usertype, "staff")
    self.assertEquals(self.openidc_user2.is_admin, True)
```

Figure 4.2: Django database render test

This function creates a new user object and then checks to make sure that the user has been given the admin and has the usertype of staff.

4.2.1.2 Discord

Unit testing in Discord.py is impossible as there is no included framework for it and there are no external libraries capable of testing to check that it works.

4.2.2 Stress Testing

To test the websites capacity to deal with 100s of requests ApacheBenchmark [29] is used which is a built into the Apache2 [3] and is a module used to request a webpage and measure response time. To test this the OpenID Connect [5] authentication system was temporarily disabled so that HTML gets served to the incoming requests instead of being redirected to the login page. Due to the nature of the websites implementation you cannot query the main page because the user is not authenticated and the page can't display information on the user, however it can query one of the subpages that does not require authentication. The following line of code was used to complete the request with 1000 requests and 100 concurrent threads to attempt to tank the website's performance.

```
ab -n 1000 -c 100 https://mmp-joa38.dcs.aber.ac.uk/privacy-policy
```

This command generates the following report which shows that the website is capable at easily handling this many requests simultaneously. The longest response time for serving the requests was 642ms which is still adequately fast for the average user and shows that Django handles requests reasonable well. It can also serve on average 174 requests per second and I expect that the website will never be receiving more that 300 requests at the exact same time which can definitely be handled as seen from these results. Another factor to consider is that this development build is currently running on a small container that only has access to a small amount of resources whereas in the final build it will have access to far more resources so could serve requests up even quicker.

```
Server Software:        Apache/2.4.38
Server Hostname:        mmp-joa38.dcs.aber.ac.uk
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,2048,256
Server Temp Key:        X25519 253 bits
TLS Server Name:        mmp-joa38.dcs.aber.ac.uk

Document Path:          /privacy-policy
Document Length:        12448 bytes

Concurrency Level:      100
Time taken for tests:   5.719 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      12734000 bytes
HTML transferred:       12448000 bytes
Requests per second:    174.86 [#/sec] (mean)
Time per request:       571.882 [ms] (mean)
```

```
Time per request:       5.719 [ms] (mean, across all concurrent requests)
Transfer rate:          2174.50 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        9  422 105.1    458     547
Processing:     7   86  61.1     67     404
Waiting:        6   83  60.1     64     381
Total:         42  508  96.3    531     642

Percentage of the requests served within a certain time (ms)
  50%    531
  66%    550
  75%    560
  80%    565
  90%    574
  95%    583
  98%    593
  99%    602
 100%    642 (longest request)
```

4.3 User Interface Testing

Testing the user interface for the AberLink is mostly straight forward as Discord has full control over the UI so I've had no testing to do on that front.

The UI of the website uses the library Bootstrap [26] as it is built from the ground up with responsive design in mind and saves a lot of time that would be required to re-learn CSS fully. This meant that website would scale very nicely and uniformly depending on screen size. To ensure that the website scales and performs well the website and all of its content have been tested on multiple devices. It has been tested on mobile devices, iPads and laptops with screens ranging from 11in to 40in using Safari, Chrome, Edge and Firefox. The website has also been tested on HTML the validation website https://validator.w3.org/ to ensure that there are no large errors that may cause the website to not load on certain machines.

4.4 User Testing

Throughout this project I have worked with the mindset that you develop small sections of code and then review what effect they have on the system. This helps to catch errors as it is much easier to backtrack through small sections of code.

For user testing I asked in the comp sci server if students could try and login to the website and break it. This turned out to work very well and helped to find a few edge cases that I had previously missed in the code.

4.4.1 Attendance API Endpoint Testing

The supplied API for attendance has been tested using user testing and the University provided me with 3 different endpoints to test. They are listed below along their responses.

```
https://integration.aber.ac.uk/joa38/good.php
{"status_updated":"true","request":"200","module_code":"CS32120"}
```

```
https://integration.aber.ac.uk/joa38/bad.php
{"status_updated":"false","request":"400",
"error_message":"Unknown user or no event"}
```

```
https://integration.aber.ac.uk/joa38/submit.php
Actual live endpoint. Returns either of the two JSON messages
above along with the corrected module_code
```

4.5 Functional Requirements

The final testing was conducted using a list of functional requirements set out in section 1.4 and the testing table can be found in Appendix C.1. Most of the tests passed correctly however some of them have not.

The Functional requirements FR3.4 & FR4.7 have only been partially fulfilled. This was a purposeful design choice as it would mean that the website would require that Discord servers are added to database and that they would have to be updated when any issues occurred. This was also bad because this list would have to be maintained and could not be automatically updated which would lead to it breaking down the line.

Chapter 5

Evaluation

5.1 Were the requirements correctly identified?

I believe that the requirements for this project have been correctly identified and have fit the scope well, however, as previously explained in section 4.5 the scope of the project was reduced slightly. The rest of the requirements were achieved to their fullest.

5.2 Were the design decisions correct?

Most of the design decisions were correct, however, as detailed in section 3.4 I did not account for Django creating its own tables used for running the website.

I also did not know that Django is unhappy with using non-numerical primary keys hence my original table design in section 2.4 failed and I had to use automatically generated numbers for the primary key of the Aber users account table as seen in section 3.6.

5.3 Could a more suitable set of tools have been chosen?

I believe that my choice of software tools was very good for this project and very manageable. I believe that if I attempted this project again I would attempt to try building up the website using a tool like React or Node.js instead of Django. These tools would give me more versatility and allow me to experiment further with the website but would be considerably harder to work with. I can however see one pitfall which is that the administration pages created by Django by default would have to be built from the ground up in JavaScript.

5.4 How well did the software meet the needs of those who were expecting to use it?

5.5 Time Management of the project

The project was managed very well and I completed to coding of the project well in advance so had lots of time to tidy up the project and add new quality of life features. Please see section 3.7.2 for more information on project iterations.

5.6 Project Meetings and Blog

Project meetings usually occurred once a week and alternated between group meetings and individual meetings. The individual meetings were really helpful as they acted as a reason for me to have a working example available for each meeting. I also had a few other meetings with other members of staff from the University to discuss what could be used in the project and policies that I had to follow.

The project blog was updated roughly once a week however during the earlier sections of the project it was updated once every few days due to a higher amount of work being performed. The blog can be found here https://cs39440blog.wordpress.com/.

Annotated Bibliography

[1] Discord Inc., “Discord,” accessed January 2021. [Online]. Available: https://discord.com/

Discord is a voice, video and text communication platform.

[2] Debian, “Debian 10 (Buster),” accessed April 2021. [Online]. Available: https://www.debian.org/releases/buster/

Debian 10 (Buster) is a Linux Distro and is used throughout this project

[3] Apache Software Foundation, “Apache 2.0,” accessed April 2021. [Online]. Available: https://httpd.apache.org/

Apache2.0 is a open source linux package for website hosting

[4] Django Software Foundation, “Django,” accessed April 2021. [Online]. Available: https://www.djangoproject.com/

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

[5] OpenID Foundation, “OpenID Connect,” accessed April 2021. [Online]. Available: https://openid.net/connect/

OpenID connect is a identity layer on top of OAuth 2.0 and is responsible for simple user verification. It provides a basic profile of the connected user to the client in a REST-like manner.

[6] Danny (Rapptz) and Various, “discord.py,” accessed January 2021. [Online]. Available: https://github.com/Rapptz/discord.py

Python API library for Discord currently available through Github.

[7] PostgreSQL Global Development Group, “PostgreSQL,” accessed April 2021. [Online]. Available: https://www.postgresql.org/

PostgreSQL is a powerful, open source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.

[8] N. Snooke, “AberVerify,” Sept. 2020, accessed April 2021. [Online]. Available: https://github.com/NealSnooke/Aber-Verify-Discord-Bot

Discord bot for Aberystwyth user verification in discord servers using email accounts (e.g. joa38@aber.ac.uk) and discord IDs (e.g. JoelinRome#4893). These accounts are then linked together using JSON objects. This project is currently available through Github.

[9] ——, "I am here," Sept. 2020, accessed April 2021. [Online]. Available: https://github.com/NealSnooke/IAmHere-Discord-Bot

Discord bot for marking Aberystwyth students' attendance during a practical. A student writes a message and the bot adds them to a list of students that have attended the practical and emails the lecturer at the end. This project is currently available through Github.

[10] F5 Company Leadership, "NGINX," accessed April 2021. [Online]. Available: https://www.nginx.com/

NGINX is a open source linux package for website hosting

[11] Joel Adams, Nathan Williams, "Demohelper," accessed April 2021. [Online]. Available: https://github.com/AberDiscordBotsTeam/demoHelperBot

DemoHelper is a Discord bot created to manage demonstrating online for students on Discord.

[12] OpenJS Foundation, "Node.js," accessed April 2021. [Online]. Available: https://nodejs.org/en/

Javascript framework for bulding websites that are compiled at runtime.

[13] Facebook and various contributors, "React.js," accessed April 2021. [Online]. Available: https://reactjs.org/

Javascript library used to create and deploy reactive web applications.

[14] Google and various contributors, "Angular.js," accessed April 2021. [Online]. Available: https://angular.io/

JavaScript framework used to build applications for mobile, web and desktop.

[15] Blaine Cook, "OAuth 2.0," accessed April 2021. [Online]. Available: https://oauth.net/2/

OAuth 2.0 is the industry-standard protocol for authorization.

[16] Various authors, "OpenID Connect authentication module for Apache2," accessed April 2021. [Online]. Available: https://packages.debian.org/sid/libapache2-mod-auth-openidc

mod-auth-openidc is a authentication library for Apache2 that uses OpenID Connect to authenticate users.

[17] ——, "Pipenv," accessed April 2021. [Online]. Available: https://pypi.org/project/pipenv/

Pipenv is a package manager for Python that uses a Pipfile to store a list of dependencies that need to be installed. These dependencies are then installed inside of a virtual environment.

[18] ——, "ASGI," accessed April 2021. [Online]. Available: https://asgi.readthedocs.io/en/latest/

Asynchronous Server Gateway Interface is the successor to WSGI used as a interface for Python webservers and frameworks.

[19] Electronic Frontier Foundation (EFF), "Certbot," accessed April 2021. [Online]. Available: https://certbot.eff.org/

Certbot is an open-souce tool used to create a Let's Encrypt certificate to allow a website to use HTTPS.

[20] graffatcolmingov, Lukasa and nateprewitt, "Requests: HTTP for Humans™," accessed April 2021. [Online]. Available: https://docs.python-requests.org/en/master/l

Requests is a Python library used to make queries to API endpoints or websites. It is open source and is available via PyPI.

[21] Various authors, "Psycopg2," accessed April 2021. [Online]. Available: https://www.psycopg.org/docs/

Psycopg2 is a Python library responsible for connecting to PostgreSQL databases. It is built in C as a libpq wrapper.

[22] eunwoo1104, "discord-py-slash-command," accessed April 2021. [Online]. Available: https://pypi.org/project/discord-py-slash-command/

Python library for Discord.py that adds support to use the slash commands available in the Discord API.

[23] Evan Wallace & Dylan Field, "Figma," accessed April 2021. [Online]. Available: https://www.figma.com/

Figma is a design website used to build application mockups.

[24] JetBrains, "PyCharm," accessed April 2021. [Online]. Available: https://www.jetbrains.com/pycharm/

A professional IDE for Python built bu JetBrains.

[25] Microsoft, "Visual Studio Code," accessed April 2021. [Online]. Available: https://code.visualstudio.com/

Free, open source code editor with a wide range of extensions to support compiling code in many languages.

[26] Various authors on Github, "Bootstrap 5.0," accessed April 2021. [Online]. Available: https://getbootstrap.com/

Bootstrap is a responsive plugin to help with stlylising webpages using HTML, CSS and JavaScript plugins.

[27] bbc and theskumar, "python-dotenv," accessed April 2021. [Online]. Available: https://pypi.org/project/python-dotenv/

Python library that loads key-value pairs from .env files.

[28] Python Software Foundation, "Shelve - Python object persistence," accessed April 2021. [Online]. Available: https://docs.python.org/3/library/shelve.html

Default module built into Python that enables users to create and save data to a micro database called a shelve file.

[29] Apache Software Foundation, "Apache HTTP server benchmarking tool (ApacheBenchmark)," accessed April 2021. [Online]. Available: https://httpd.apache.org/docs/current/programs/ab.html

AB is a tool used to stress test a websites capacity to deal with 1000s of requests at once.

[30] stub, "pytz," accessed April 2021. [Online]. Available: https://pypi.org/project/pytz/

Used for accurate, cross-platform timezone calculation. It is open source and is available via PyPI.

[31] aalbrecht, "sqlparse," accessed April 2021. [Online]. Available: https://pypi.org/project/pytz/

Python library for parsing, splitting and formatting Non-validationg SQL libraries. It is open source and is available via PyPI.

[32] Internet Security Research Group (ISRG), "Let's Encrypt," accessed April 2021. [Online]. Available: https://letsencrypt.org/

A nonprofit Certificate Authority providing TLS certificates to HTTPS.

[33] Ashald and ncbi, "libpq-dev," accessed April 2021. [Online]. Available: https://packages.debian.org/sid/libpq-dev

Debian package required for building C header files for the psycopg2 library.

[34] Guido van Rossum, "python-dev," accessed April 2021. [Online]. Available: https://packages.debian.org/buster/python-dev

Debian package required for building Python 2 modules such as psycopg2.

Appendices

Appendix A

Third-Party Code and Libraries

I declare that all the following libraries are unmondified.

- **Pipenv** [17] - This is a Python package manager that is used throughout this project to create and maintain virtual environments. It uses Pipfiles (text files) to store a list of dependencies that are then downloaded and stored in a virtual environment. In this project I have two Pipfiles; one for the website (`src\AberLinkAuthentication`) and one for the Discord bot (`src\AberLinkDiscord`). This library uses the MIT license so is free to use.
- **Django** [4] - This is a Python web framework used for building websites. It is used to display the webpages, sign users in and help users and staff manage their data. This library is open source and provided by the Django Software Foundation.
- **Apache2** [3] - This Debian [2] package is responsible for hosting a HTTP web server. In this case it hosts the Django [4] framework so it can be accessed on the web. This library is released under the Apache License, Version 2.0.
- **libapache2-mod-auth-openidc** [16] - Debian [2] package used to authenticate users when they first connect to the site. It authenticates users against their Aberystwyth University account and then creates a session where they can access content on the website. This library is released under the GNU Free Documentation License
- **certbot** [19] - This package was only briefly required to setup HTTPS on the website using Let's Encrypt [32]. Once SSL certificates have been made cerbot then runs a cronjob in the background to update it. This library is licensed under the Apache 2.0 license
- **psycopg2** [21] - This library is used for talking to PostgreSQL databases in Python. The version used here is called psycopg2-binary and it requires two additional dependencies to be built libpq-dev [33] and python-dev [34]. This library is licensed under the GNU Lesser General Public License.
- **Discord.py** [6] - This Python library is used to create and write the Discord bot. It is basically an interface that allows me to write bots without making API requests straight to Discord. This library is licensed under the MIT License.

- **Requests** [20] - It is used to make HTTP requests to the web. In this project it is used in the website to get information from the Discord login and in the Discord bot it is used to send and receive data from the attendance API endpoint https://integration.aber.ac.uk/joa38/submit.php. This library is licensed under the Apache Software License (Apache 2.0)
- **Pytz, SQLParse & ASGIref** [30] [31] [18] - These are dependencies required to work with Django. pytz helps with managing timezones and timezone data. SQLParse handles SQL queries, formatting and splitting. Asgiref is an interface used to talk between Apache2 [3] and Django [4] for web server hosting and frameworks. These libraries are all licensed under the MIT License.

Appendix B

Ethics Submission

Application Number: **18847**

Assessment Details

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
joa38@aber.ac.uk

Full Name
Joel Adams

Please enter the name of the person responsible for reviewing your assessment.
Neil Taylor

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
nst@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
Discord bot for verification and attendance linked to OpenID Connect

Proposed Start Date
25 January 2021

Proposed Completion Date
1st June 2021

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Does your research involve human participants?
Yes

Are you completing this form for your own research?
Yes

Does your research involve human participants?
Yes

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
I'm creating a system that links student's discord accounts to their uni accounts. This is done through a website that allows users to log in with their Discord accounts and their student account using Discords login system and OpenID Connect respectively so no password information is ever stored. This information is then saved in a backend postgresql database stored in the Department of Computer Science's servers so it is protected from remote attacks. A discord bot is then used on the front end to verify if a student has a verified Discord account and what student account that is linked to. It is also responsible for marking students' attendance in practicals.

I can confirm that the study does not involve vulnerable participants including participants under the age of 18, those with learning/communication or associated difficulties or those that are otherwise unable to provide informed consent?
Yes

I can confirm that the participants will not be asked to take part in the study without their consent or knowledge at the time and participants will be fully informed of the purpose of the research (including what data will be gathered and how it shall be used during and after the study). Participants will also be given time to consider whether they wish to take part in the study and be given the right to withdraw at any given time.
Yes

I can confirm that there is no risk that the nature of the research topic might lead to disclosures from the participant concerning their own involvement in illegal activities or other activities that represent a risk to themselves or others (e.g. sexual activity, drug use or professional misconduct).
Yes

I can confirm that the study will not induce stress, anxiety, lead to humiliation or cause harm or any other negative consequences beyond the risks encountered in the participant's day-to-day lives.
Yes

Please include any further relevant information for this section here:

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
Not applicable

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

Is your research study related to COVID-19?
No

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here:
N/A

Appendix C

Functional Requirements Table

Functional Requirement	Passed?	How?
FR1	Yes	Passed all requirements. Using functions from file *AberLinkAuthentication/login/views.py*
FR1.1	Yes	OpenID Connect only authenticates aber users.
FR1.2	Yes	function *openidc_response()* gets metadata from OpenID Connect login and usertype.
FR1.3	Yes	*openidc_response()* gets user information and logs user in or creates a new account with the correct privileges.
FR2	Yes	Passed all requirements. Using functions from file *AberLinkAuthentication/login/views.py*
FR2.1	Yes	*discord_oauth2_redirect()* gets metadata by getting authorization from url code and calls *exchange_code()* to query Discord API for metadata on user.
FR2.2	Yes	*discord_oauth2_redirect()* contains a line to get the openidc_user account and link the Discord account to it using the foreign key relationship.
FR2.3	Yes	Tested connecting 3 Discord accounts to the same openidc_user with no errors.
FR3	Partially	Most requirements passed. Using functions from file *AberLinkAuthentication/login/views.py*
FR3.1	Yes	*openidc_response()* displays main webpage along with information on the linked user from the metadata.
FR3.2	Yes	*openidc_response()* displays linked Discord accounts linked to Openidc_user account.
FR3.3	Yes	*openidc_response()* contains function *get_discord_users()* that calls Discord API using the Discord IDs to get profile picture and username.
FR3.4	No	Realised that feature would require information to be stored about module servers and also know which modules a user takes. This information is neither accessible nor easy to maintain so feature scrapped.

FR4	Partially	Most requirements passed. Using functions from file *AberLinkAuthentication/login/admin.py*
FR4.1	Yes	When user is logged in Django checks that they have *is_admin* permissions set to True before they can access admin webpages. If they don't option is greyed out and not clickable. Manually typing in Admin link results in message telling user that they do not have permissions.
FR4.2	Yes	*OpenIDCAdmin()* displays table containing aber (openidc) accounts.
FR4.3	Yes	*DiscordAdmin()* displays table containing Discord accounts and their linked openidc accounts.
FR4.4	Yes	Both *DiscordAdmin()* and *OpenIDCAdmin()* contain the line *readonly_fields* that disallow admins to change this information. Visually on website Admins can only give users admin permissions.
FR4.5	Yes	Admins can delete users accounts; both openidc and Discord.
FR4.6	Yes	Admins cannot add new Aber or Discord accounts manually.
FR4.7	No	Feature scrapped due to complexity of the design and would also not work well in Python but would be better attempted in Node.js or React. Also would require more Discord API calls to get server information and configure servers.
FR5	Yes	Passed all requirements. Using functions from file *AberLinkDiscord/cogs/verify.py*
FR5.1	Yes	*build()* sets server up for verification properly creating new roles, a verify channel and pins a message. See 2.3.3 for more details.
FR5.2	Yes	*verify.py/verify()* allows uses to call a command and manually verify, alternatively users are automatically verified when they join using *on_member_join()*.
FR5.3	Yes	*verifyAlumni()* is a callable command that tells alumni how to verify by send an email to afc@aber.ac.uk.
FR5.4	Yes	Both *verify()* and *on_member_join()* contain a function called *check_shelve_file()* that checks to see if the server has auto nickname enabled.
FR6	Yes	Passed all requirements. Using functions from file *AberLinkDiscord/cogs/here.py*
FR6.1	Yes	*here()* calls functions from AberLinkDiscord/cogs/db.py to do reverse lookup using discord ID to get Aber username.
FR6.2	Yes	*here()* gets information as described in FR6.1 and sends it to the university API endpoint `https://integration.aber.ac.uk/joa38/submit.php`. Information is then returned to the function and the user is sent a direct message about their status.
FR7	Partially	Most requirements passed. Using files from path *AberLinkDiscord/cogs/*

FR7.1	Yes	File called *shelve_file.shelve.db* exists or is created and is used in function *verify.py/verify()* to check whether to give nickname automatically.
FR7.2	Yes	Function called *utilities.py/configurations()* displays configurations available for the current server.
FR7.3	Yes	Function called *utilities.py/ClearMessages()* clears the last 14 days of messages.
FR7.4	Yes	Function called *utilities.py/source()* displays message containing link to source code and website.
FR7.5	Yes	Function called *utilities.py/bots()* displays message with useful bots that can be added to server.
FR7.6	Yes	Function called *utilities.py/ping()* displays bot and message edit latency. It also gets information on the database connection from functions in *db.py*.
FR7.7	No	Feature was not added after chat with personal tutor explaining that links can change and might break over time so terrible for maintainability.
FR8	Yes	Passed all requirements. Using functions from class *AberLinkDiscord/cogs/db.py*
FR8.1	Yes	*connect()* connects to the database, returns information on connection and defines global variable for connection.
FR8.2	Yes	*try_connection()* reconnects to database if connection is lost.
FR8.3	Yes	*get_discord_user()* gets Discord information from database using Discord ID.
FR8.4	Yes	*get_openid_user()* gets information from database about aber account using their generated ID.
FR8.5	Yes	*get_connection_status()*, *get_polling_status()* and *get_connection_latency()* get information from database.

Table C.1: Functional Requirements testing table

Appendix D

REAMDME.md Configuration File

4.1 Building the project

This directory contains some sample config files that can be used a reference for setting up the project. Feel free to use them and change the variables accordingly.

4.1.1 Setting up Apache2

1. `sudo apt install apache2 -y` - Install apache2
2. `sudo apt install libapache2-mod-auth-openidc` - Installs library for running openid
3. Enable the following mods: `ssl`, `auth_openidc`, `wsgi`, `rewrite`
4. `sudo nano /etc/apache2/auth_openidc.conf` and copy the following to the file

```
OIDCProviderMetadataURL https://openidc.dcs.aber.ac.uk/auth/realms/MMP-IMPACS/.well-known/openid-configuration
OIDCClientID MMP-IMPACS
OIDCRedirectURI /oauth2callback #Replace with link to website e.g. "discord.dcs.aber.ac.uk/oauth2callback"
OIDCCryptoPassphrase some-random-string-for-encrypting-cookies
OIDCScope "openid basic"
OIDCRemoteUserClaim preferred_username
OIDCSessionInactivityTimeout 86400
```

5. `sudo cp ~/aberlink/config/aberlink.conf /etc/apache2/sites-available/` - copy the config file to the directory
6. Edit the file to match the file structure, below is an example of what to change the settings to:

```
<VirtualHost *:80>
        ServerName discord.dcs.aber.ac.uk
        ServerAlias discord.dcs.aber.ac.uk
        ServerAdmin cs-support@aber.ac.uk
        DocumentRoot /home/joel/aberlink/src/

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

7. `sudo a2ensite aberlink` - Enable the website
8. Go to `sudo nano /etc/apache2/apache2.conf` and add the following:

```
<Directory /home/joa38/aberlink/src/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>
```

4.1.2 Setting up SSL

1. Install the certbot package: `sudo apt intall certbot`
2. Run this command to create the SSL certificates: `sudo certbot certonly --webroot -w ~/aberink -d discord.dcs.aber.ac.uk`
3. `sudo a2enmod rewrite` - Enable the rewrite mod for website redirect
4. `sudo a2enmod ssl` - Enable the ssl mod for SSL to work on the website
5. `source /etc/apache2/envvars` - fixes errors in apache2
6. Open the conf file again (e.g. `aberlink.conf`) and change it match the domain name:

```
<VirtualHost *:443>
        ServerAdmin cs-support@aber.ac.uk
        serverName discord.dcs.aber.ac.uk
        ServerAlias discord.dcs.aber.ac.uk
        DocumentRoot /home/joa38/aberlink/src/
   AberLinkAuthentication

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

```

```
10        Alias /static /home/joa38/aberlink/src/
   AberLinkAuthentication/static
11        <Directory /home/joa38/aberlink/src/AberLinkAuthent/
   static>
12                Require all granted
13        </Directory>
14
15        <Directory /home/joa38/aberlink/src/
   AberLinkAuthentication/AberLinkAuthentication>
16                <Files wsgi.py>
17                        Require all granted
18                </Files>
19        </Directory>
20
21        <Directory /home/joa38/aberlink/src/
   AberLinkAuthentication>
22                Options Indexes FollowSymLinks
23                AllowOverride None
24                Require all granted
25                Allow from all
26        </Directory>
27
28        <Location />
29                AuthType openid-connect
30                Require valid-user
31        </Location>
32
33        WSGIScriptAlias / /home/joa38/aberlink/src/
   AberLinkAuthentication/AberLinkAuthentication/wsgi.py
34        WSGIDaemonProcess django_app python-path=/home/joa38/
   aberlink/src/AberLinkAuthentication python-home=/home/joa38/
   aberlink/src/AberLinkAuthentication/venv
35        WSGIProcessGroup django_app
36
37        SSLEngine on
38        SSLCertificateFile /etc/letsencrypt/live/discord.dcs.
   aber.ac.uk/fullchain.pem
39        SSLCertificateKeyFile /etc/letsencrypt/live/discord.dcs.
   aber.ac.uk/privkey.pem
40        SSLProtocol all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
41 </VirtualHost>
```

7. `sudo apachectl restart` - Restart Apache2 to enable the new mods

4.1.3 installing the dependencies for the bot

1. `sudo apt install libpq-dev python-dev` - needed to install `psycopg2-binary`
2. Navigate to the folder `aberlink/src/AberLinkDiscord` and type `pipenv install`
3. Create a discord bot token by visitng `https://discord.com/developers/applications` and creating a new bot called AberLink along with the supplied photo `/img/AberLink_logo_cropped.png`. Then head on over to the Bot panel and create a new bot with the same information as above. Finally copy the token underneath the bots name for the next section.
4. While inside the folder `/aberlink/src/AberLinkDiscord` create a new file using `nano .env` and add the following (or copy the example provided in this folder):

```
DISCORD_TOKEN= # Discord token found on bot's page
DATABASE_NAME= # Postgres database name
USER= # Postgres user
PASSWORD= # Postgres password
HOST= # Postgres host
PORT=5432 # This is the default port
WEBSITE_URL= # The website for AberLinkAuthentication e.g. https
   ://joa38-mmp.dcs.aber.ac.uk/
```

5. Check that the .env file has been configured correctly by typing in the command `pipenv run python3 -m AberLink`.
6. Invite the bot to one of your servers by creating an invite link in the tab OAuth2 and selecting `bot` in the `scopes` section and then in the `bot permissions` tick `Administrator`. Copy the link generated by it and paste it into the browser and invite the bot to your server.
7. In the OAuth2 tab `Redirects` section add your website uri e.g. `https://discord.dcs.aber.ac.uk/oauth2/login/redirect`.
8. On the same page scroll down to the `OAuth2 URL Generator` and select the web address you entered in the `Redirects` section.
9. Scroll down again and in the `scopes` section select `identify` and copy the url. Then head on over to the file `/aberlink/src/AberLinkAuthentication/login/views.py` and find the `discord_oauth2` function and replace the redirect URL with your own.

4.1.4 Installing and running the webserver

1. navigate to the folder `aberlink/src/AberLinkAuthentication` and type `virtualenv venv` to create a virtual enviornment folder for the data.

2. `source venv/bin/activate` - activates the virtuanenv
3. `pipenv install` installs dependencies from the project into the file
4. Make sure that that the venv file path is correct inside of the `aberlink.conf` file.

4.1.5 Django setup

1. `sudo cp config/config.json /etc/config.json` - copy the template file for the django config and fill out the details below (email joa38@aber.ac.uk for `SECRET_KEY`):

```
{
    "SECRET_KEY": "",
    "DATABASE_NAME": "",
    "USER": "",
    "PASSWORD": "",
    "HOST": "",
    "PORT": "",
    "DISCORD_CLIENT_SECRET": "",
    "DISCORD_TOKEN": "",
    "WEBSITE_URL": ""
}
```

Note: The `WEBSITE_URL` is the name of the website that is going to be used. e.g. `https://joa38-mmp.dcs.aber.ac.uk`

The `DISCORD_CLIENT_SECRET` and `DISCORD_TOKEN` can be found by visitng the page created earlier for the discord bot.

1. Navigate to the `General Information` tab and copy the client secret which is located below the Description on the right hand side. Copy and save this variable to the `DISCORD_TOKEN`
2. Naviage to the `Bot` tab and copy the name token located below the username and save it to the `DISCORD_CLIENT_SECRET`

After setting up the config file open the command shell and type the following commands to configure the database for Django:

1. `python3 manage.py makemigrations`
2. `python3 manage.py sqlmigrate`
3. `python3 manage.py migrate`

Finally once everything has been ensured to be fully functioning open the file `src/AberLinkAuthentication/AberLinkAuthentication/settings.py` and find the line `DEBUG = True` and set the variable to `False`