

# **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 unis 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 uni 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 unis 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

AberLink contains 2 components; a website which is used to link up Discord [1] accounts to students' accounts and a Discord bot called AberLink that is used for verifying students' Discord accounts. This bot is also responsible for marking students' attendance during practicals that have the flag of "Discord" on their student record.

This system was developed on the Linux distribution Debian 10 (Buster) [2]. It uses the open source HTTP server Apache2.0 [3] to web-host along with the Python framework Django [4] for the website. The 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.

Information is saved to a back-end PostgreSQL [6] database using primary keys to link the uni and Discord accounts together. The Discord bot (AberLink) using the Python library Discord.py [7] then verifies Discord accounts by checking if that account is present in the database and then grant users access to the Discord server. AberLink also has an API endpoint for marking attendance on Student Record and uses the database to perform a reverse search on Discord users to get their uni information and send that to the endpoint.

The AberLink Discord bot is based on two previous bots called Aber Verify Bot [8] and I am here! [9], both created by 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

<b>1</b>	<b>Background &amp; Objectives</b>	<b>1</b>
1.1	Background . . . . .	1
1.1.1	Research . . . . .	1
1.2	Analysis . . . . .	2
1.2.1	Objectives . . . . .	2
1.2.2	Possible Security Issues . . . . .	3
1.3	Process . . . . .	4
1.4	Functional Requirements . . . . .	5
<b>2</b>	<b>Design</b>	<b>7</b>
2.1	Overall Architecture . . . . .	7
2.2	Website Architecture and Design . . . . .	8
2.3	Discord Bot Architecture and Design . . . . .	10
2.3.1	Database Interaction . . . . .	11
2.3.2	Discord Commands Interactions . . . . .	12
2.3.3	Complicated Behaviours . . . . .	13
2.4	Database design . . . . .	14
2.5	User Interface . . . . .	14
2.5.1	Website . . . . .	15
2.5.2	Discord bot . . . . .	16
2.6	Version Control System and IDE . . . . .	17
<b>3</b>	<b>Implementation</b>	<b>18</b>
3.1	Code Implementation & Third-Party libraries . . . . .	18
3.1.1	Website Building and Final Design . . . . .	18
3.1.2	Discord Bot . . . . .	21
3.1.3	Database . . . . .	25
3.2	Configuration and Setup . . . . .	28
3.3	Unforeseen Issues . . . . .	28
3.4	Review . . . . .	29
3.4.1	Review Against Planned Requirements . . . . .	29
3.4.2	Review Against Project Process . . . . .	29
<b>4</b>	<b>Testing</b>	<b>31</b>
4.1	Sample Data . . . . .	31
4.2	Automated Testing . . . . .	31
4.2.1	Unit Tests . . . . .	31
4.2.2	Stress Testing . . . . .	33
4.3	User Interface Testing . . . . .	34
4.4	User Testing . . . . .	34
4.4.1	Attendance API Endpoint Testing . . . . .	34
4.5	Functional Requirements . . . . .	35
<b>5</b>	<b>Evaluation</b>	<b>36</b>
5.1	Were the requirements correctly identified? . . . . .	36
5.2	Were the design decisions correct? . . . . .	36

5.3	Could a more suitable set of tools have been chosen? . . . . .	36
5.4	How well did the software meet the needs of those who were expecting to use it? . . . . .	37
5.5	Time Management of the project . . . . .	37
5.6	Project Meetings and Blog . . . . .	37
<b>Annotated Bibliography</b>		<b>38</b>
<b>Appendices</b>		<b>41</b>
<b>A</b>	<b>Third-Party Code and Libraries</b>	<b>43</b>
<b>B</b>	<b>Ethics Submission</b>	<b>45</b>
<b>C</b>	<b>Functional Requirements Table</b>	<b>48</b>

## List of Figures

2.1	Architecture diagram of the overall system . . . . .	7
2.2	Architecture diagram of website . . . . .	8
2.3	Website flowchart for authentication of accounts and display of user data . . . . .	9
2.4	Architecture diagram of Discord bot . . . . .	10
2.5	Tree diagram showing Discord file structure . . . . .	10
2.6	AberLinkDiscord UML diagram . . . . .	11
2.7	Sequence diagram for Discord bot to database . . . . .	12
2.8	Sequence diagram for Discord bot to database . . . . .	13
2.9	Entity relationship diagram for database . . . . .	14
2.10	Website mock-up for 0 users . . . . .	15
2.11	Website mock-up for 1 user . . . . .	15
2.12	Website mock-up for 2 users . . . . .	16
2.13	Discord Embed example for getting users information . . . . .	16
2.14	Discord Embed example for source command . . . . .	17
2.15	Example of GitLab repository commits . . . . .	17
3.1	Final website with 0 linked Discord accounts . . . . .	19
3.2	Final website with 1 linked Discord accounts . . . . .	19
3.3	Final website with 2 linked Discord accounts . . . . .	20
3.4	Final website Admin page for Aber accounts . . . . .	20
3.5	Final website Admin page for Discord accounts . . . . .	21
3.6	Snippet of code from AberLink.py main Discord bot file . . . . .	22
3.7	Example of AberLink's ping command code . . . . .	23
3.8	Example of ping command in Discord . . . . .	23
3.9	Example of Discord help command to display list of Discord commands for AberLink . . . . .	24
3.10	Example of AberLink's on_command_error() checking statements . . . . .	24
3.11	Django database model of OpenID Connect Aber users . . . . .	26
3.12	Final Entity Relationship diagram for database . . . . .	27
3.13	Updated Entity relationship diagram for database . . . . .	29
3.14	GitLab graph of commits over time . . . . .	30
4.1	Django URL render test . . . . .	32
4.2	Django database render test . . . . .	32

## List of Tables

3.1	Aberystwyth user table example . . . . .	28
3.2	Discord user table example . . . . .	28
3.3	Project iterations and what occurred . . . . .	30
C.1	Functional Requirements testing table . . . . .	50



# 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 and further weeks to decide what can be used I began work on the background and spike work as soon as possible. By the end of the first week I had already had completed most of the spike work required to form the Project Outline and send it out to IS and CS-support. 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 and I encourage that you have a look there too when reviewing this section.

#### 1.1.1 Research

**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. I also used HTML, CSS and some Javascript to develop the website pages.

**Databases** - Early on I decided that a relational database was best suited for the data as

I am 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) [6] was used as I am already familiar with using it in a second year project and it has native support for the web framework that I will discuss later.

**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 I settled on Django. I chose this 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 [6] so that was a big bonus too.

## 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 uni data and how to login users who are only on the uni 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 [6] and make it secure from outside attacks.
- **Version control, documentation and setup.**
  - Create a container on the uni 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 [6] 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 uni 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 aren't.
  - 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

Throughout this project I have attempted to minimise the amount of security risks that can occur. The main identified weakpoints are as follows:

- **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 [6] 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 uni'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 doesn't 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 [12]. 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 run usually from Monday to Friday as I try 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

- **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 doesn't 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 doesn't 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 verify 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 whether 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 whether database connection is working correctly.
  - **FR7.7** - Bot has a search tool to get information on Aber uni 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 up-time.
  - **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

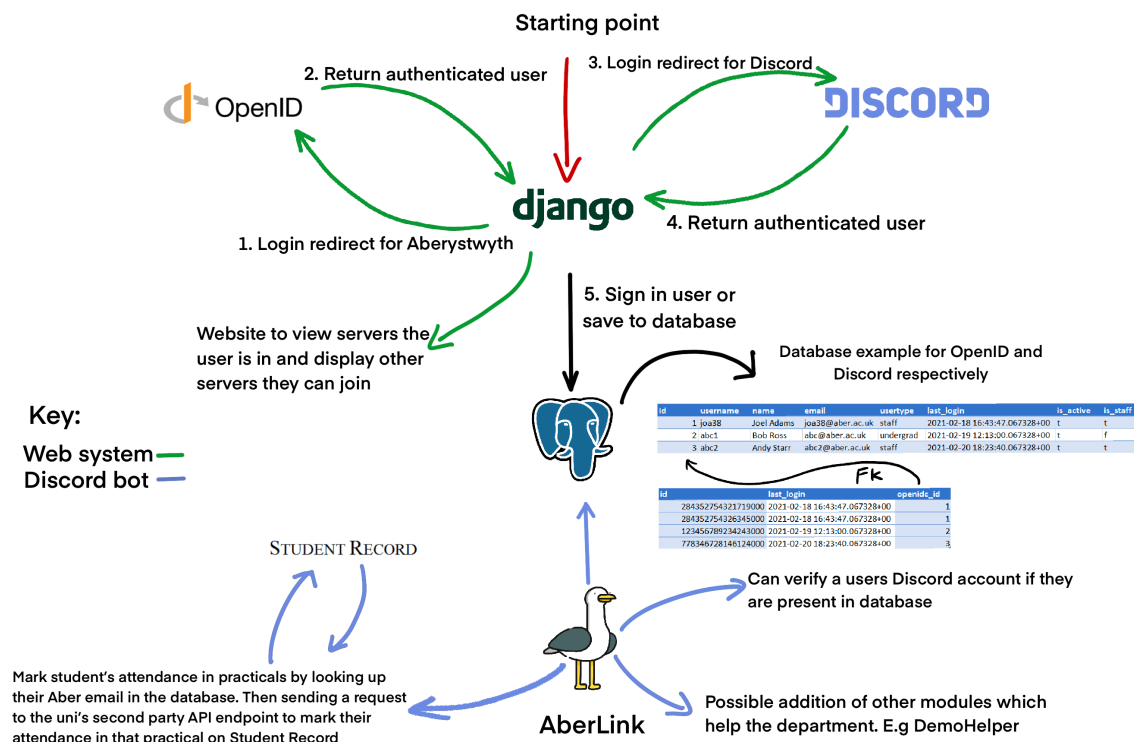


Figure 2.1: Architecture diagram of the overall system

As seen above here is a brief overview of the main components used in this project which I go into 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 of 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 [13], Discord bot and Django [4].

- **dev** - Contains the spike work completed throughout this project and I strongly encourage that you review it when reading through this document.
- **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 [14] 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

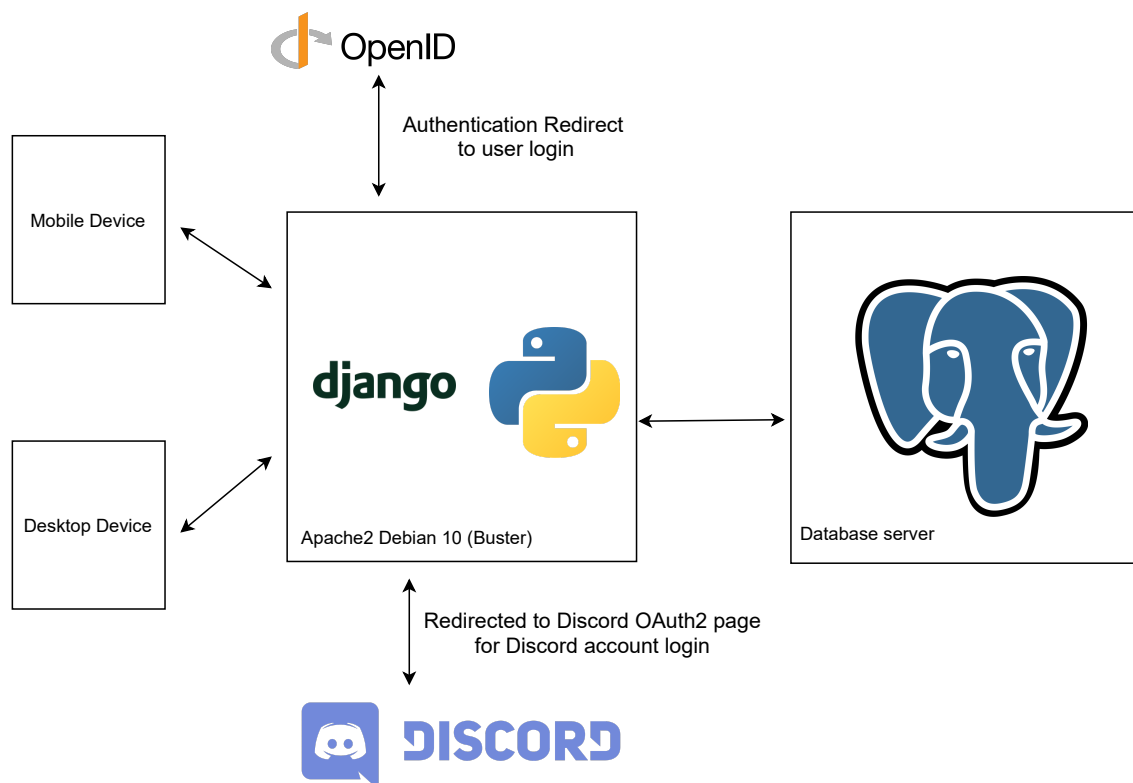


Figure 2.2: Architecture diagram of website



The website is built using the Python framework Django [4]. This framework has been used to generate a template of the code required and only requires minimal tweaking to setup. When you visit the website you then get redirected to the OpenID Connect [5] authentication framework which helps to prevent unwanted access by users who aren't permitted to view the website. Once authenticated 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've logged in with a discord account then it redirects them back to the my website and saves that information to the database as well. I have included a flow chart below to explain the process.

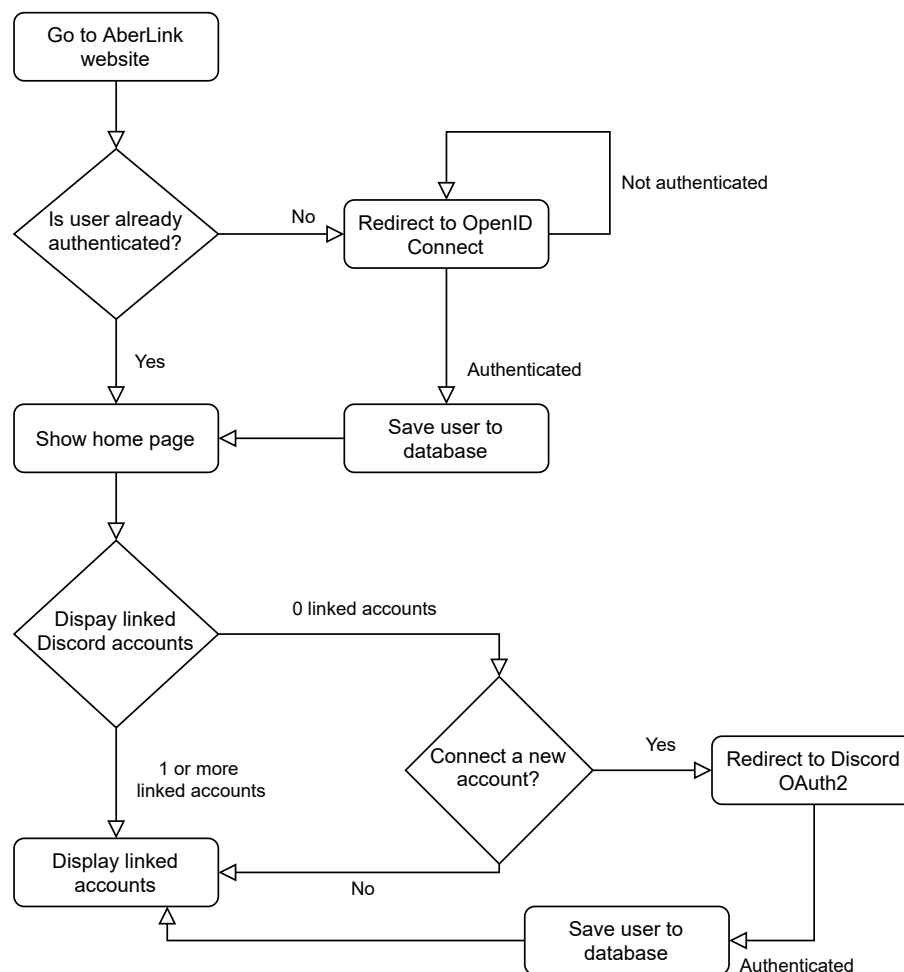


Figure 2.3: Website flowchart for authentication of accounts and display of user data

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

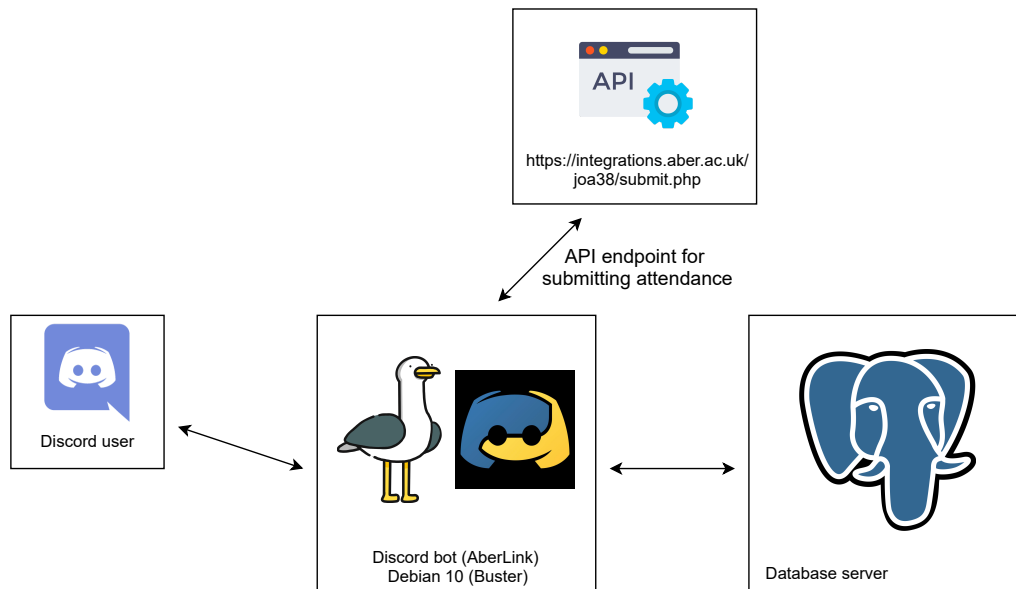


Figure 2.4: Architecture diagram of Discord bot

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 [7] 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're called here. The .env file is used 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 [7], requests [15], psycpg2 [16] and the version of Python required to run the bot. The Pipfile can be initialised using the Python virtual environment pipenv [14].

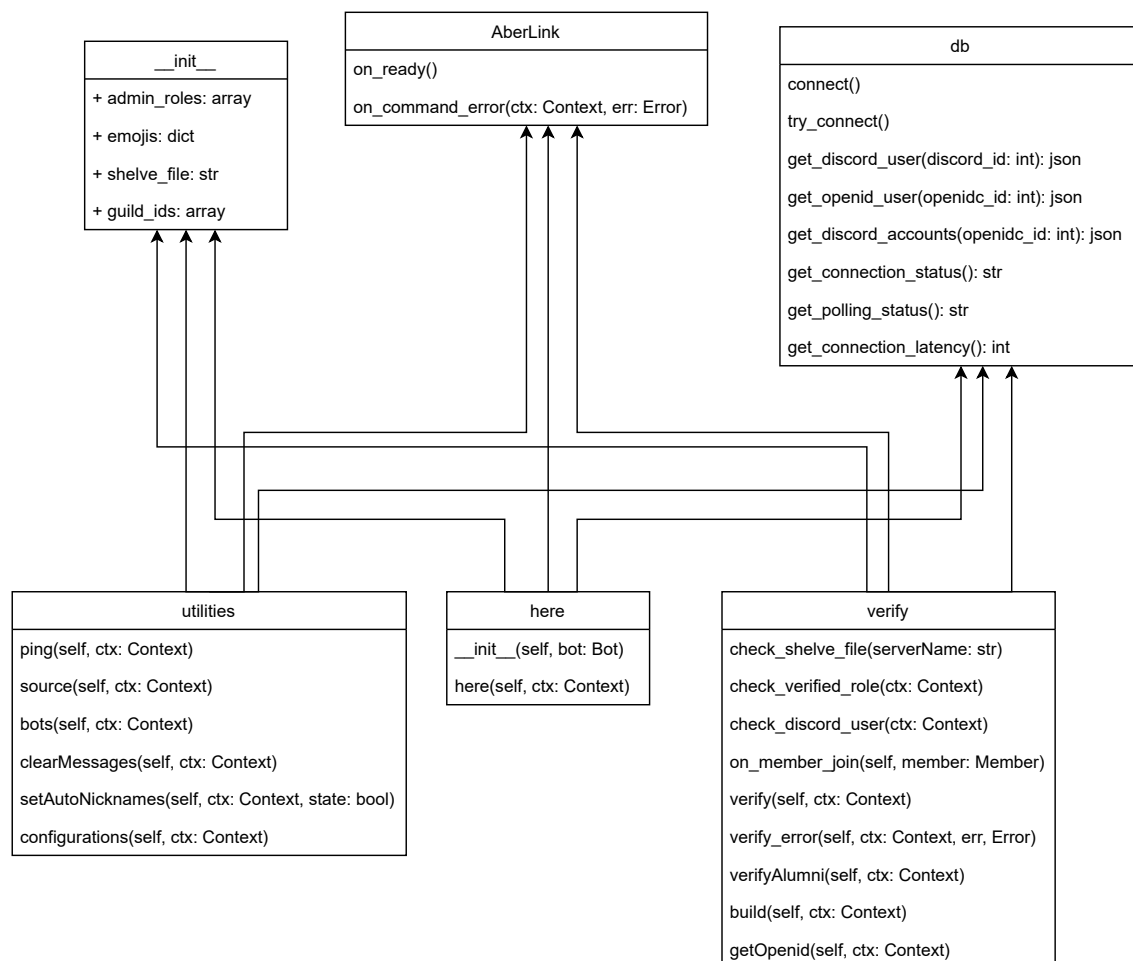


Figure 2.6: AberLinkDiscord UML diagram

### 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 psycpg2 [16] 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 `psycpg2`.

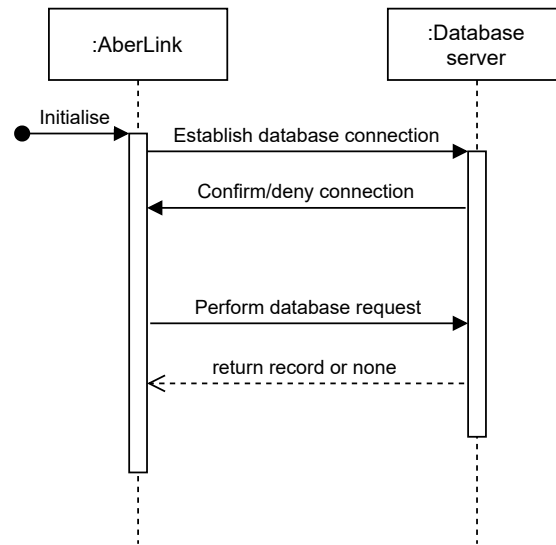


Figure 2.7: Sequence diagram for Discord bot to database

### 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. `!help` would display the information for the help command. This option has been around since the beginning of Discord bots and is a reliable and simple to use method however the end user needs to remember what the commands are to use them.

The new workaround for this system is to use ‘slash commands’ which are a new feature in Discord that allow the user to type a `/` and Discord 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 in relevant libraries. `Discord.py` [7] does not support the feature at all as it would require an entire library re-write so external libraries have emerged to support this feature such as `discord-py-slash-command` [17]. These libraries however are currently very early in development and therefore subject to change and restructuring. As this project will be deployed after it’s creation then we want it to be robust and not break after a new update to the slash commands library.

Despite it’s infancy, there is basic support for slash commands in `AberLink` but are commented out. Below is a screenshot to show a working example of the `AberLink` source command appearing as a Discord `/` command.

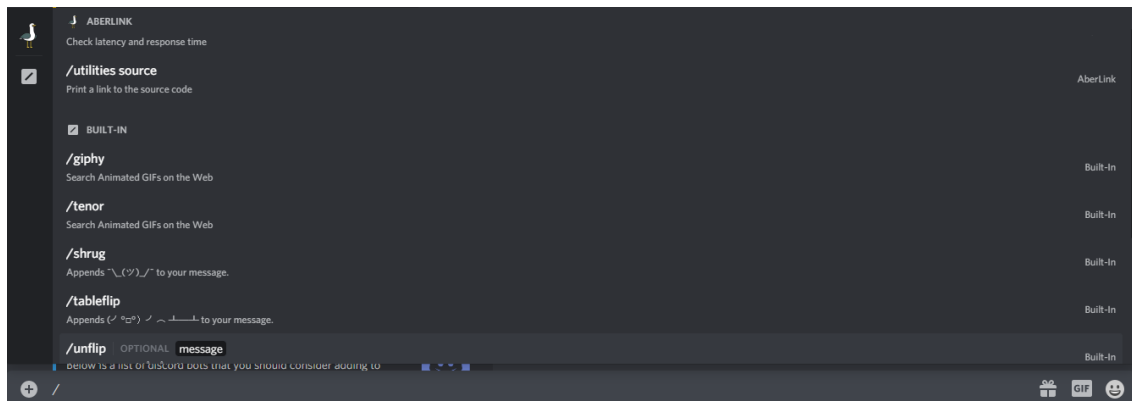


Figure 2.8: Sequence diagram for Discord bot to database

### 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 function and Discord command is used to configure a Discord server for verification. 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 it's 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 doesn't 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` doesn't 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 each other.

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 [6] database to fit those requirements.

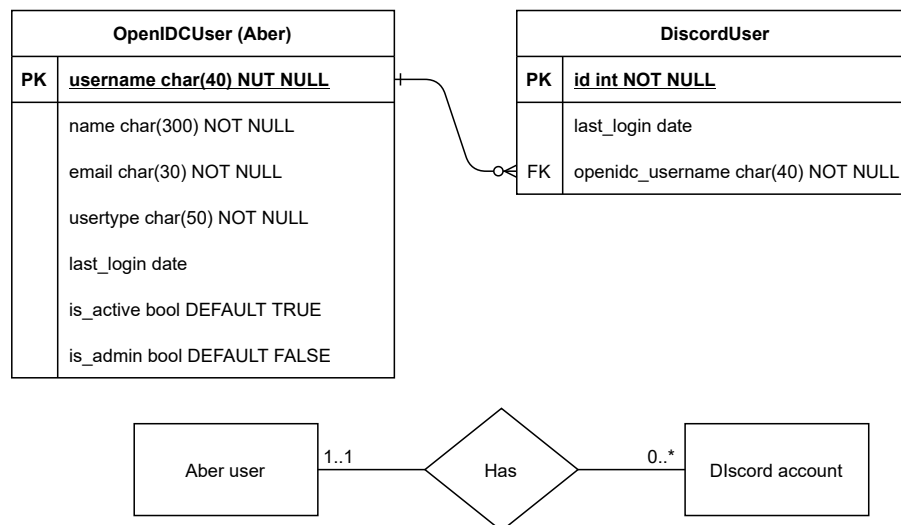


Figure 2.9: Entity relationship diagram for database

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

Before beginning work on the website I designed some website mock-ups that I have used as reference when working on the design of the webpages. I created these mockups using the popular design application Figma [18] and they are located below.

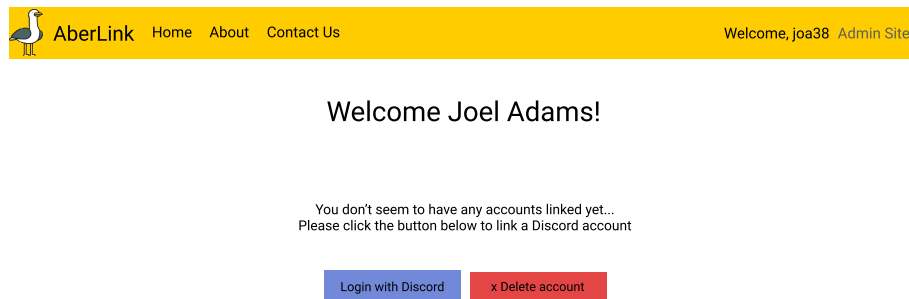


Figure 2.10: Website mock-up for 0 users

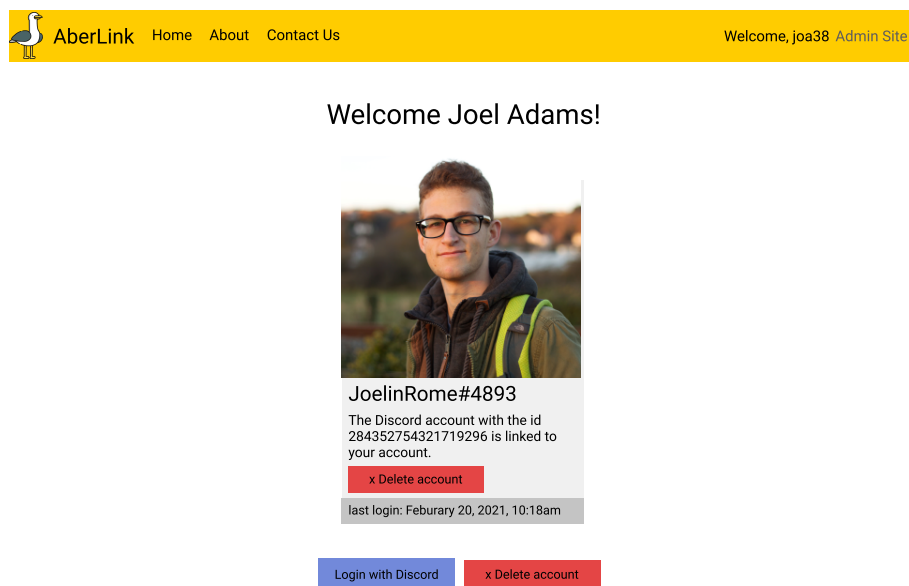


Figure 2.11: Website mock-up for 1 user

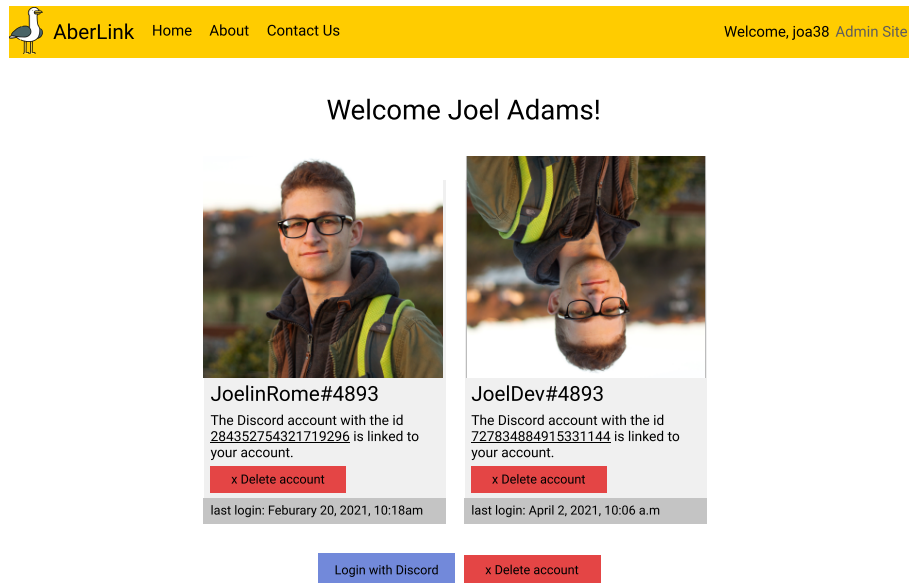


Figure 2.12: Website mock-up for 2 users

### 2.5.2 Discord bot

Discord provides its own interface for Discord bots and doesn't vary much from the regular interface provided to users. Discord bots however have the additional feature of being able to use Embeds which contain far better formatting than regular messages and are used extensively throughout AberLink. Below are some examples of the output of bot commands that I have created using Embeds.

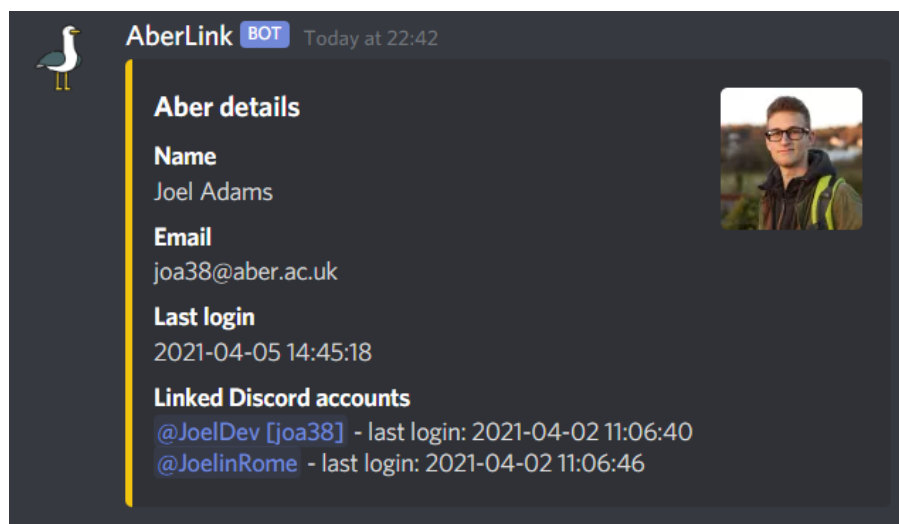


Figure 2.13: Discord Embed example for getting users information



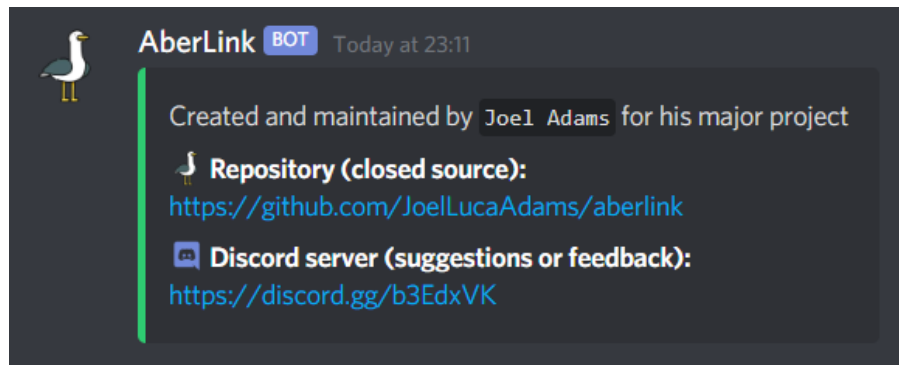


Figure 2.14: Discord Embed example for source command

## 2.6 Version Control System and IDE

Over the last year of working in Python I have used JetBrains Python IDE PyCharm [19] to code my Python projects but recently switched to Visual Studio Code [20] due to the amount of extensions it supports and the versatility it has. I chose this because it has extensions for querying a psql [6] database, compile Python programs, use virtual environments and write LaTeX documents like this one. It was also chosen as it allows you to remotely connect to a host over SSH which is very useful as I have developed the majority of this program on a Linux (Debian 10 Buster [2]) container. This last feature is not supported in PyCharm hence my choice of Visual Studio Code.

For Version Control system I have used the GitLab instance provided by the university to store my code. I have also added mirroring of the project to another repository called aberlink on my primary GitHub account in case of emergencies. I've included below an example of the commits that are made to the repository.

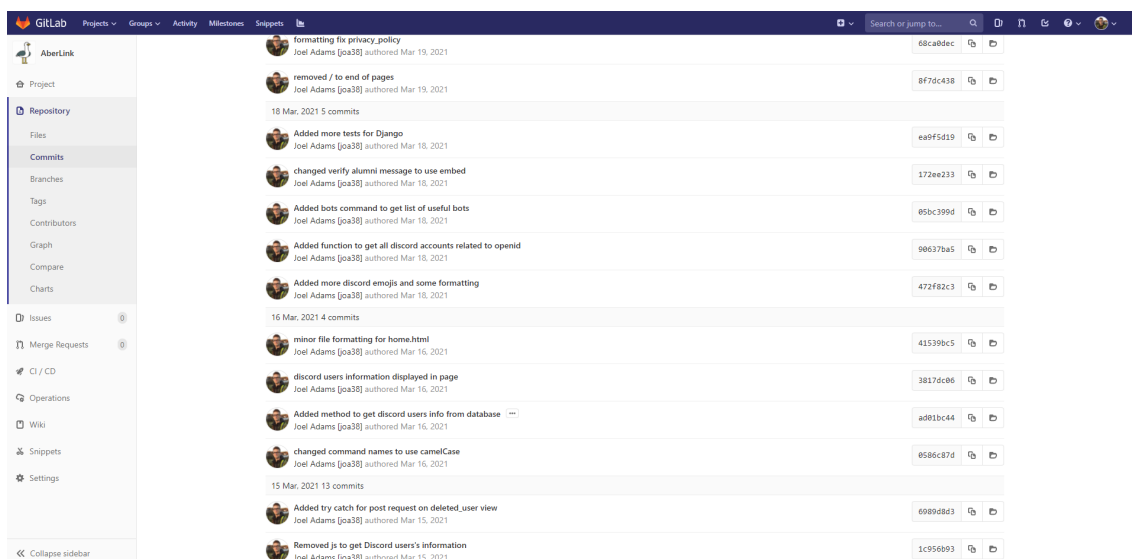


Figure 2.15: Example of GitLab repository commits

## 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 I decided to make 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.

#### 3.1.1 Website Building and Final Design

To create and run the website I've used a set of different libraries to perform certain tasks. Firstly I've used the Apache2 [3] web hosting framework to host 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 I have used the linux tool certbot [21] 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 general template I created 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 shouldn't fall under UAP.

The website pages also use a third party library called bootstrap [22] 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 I 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.

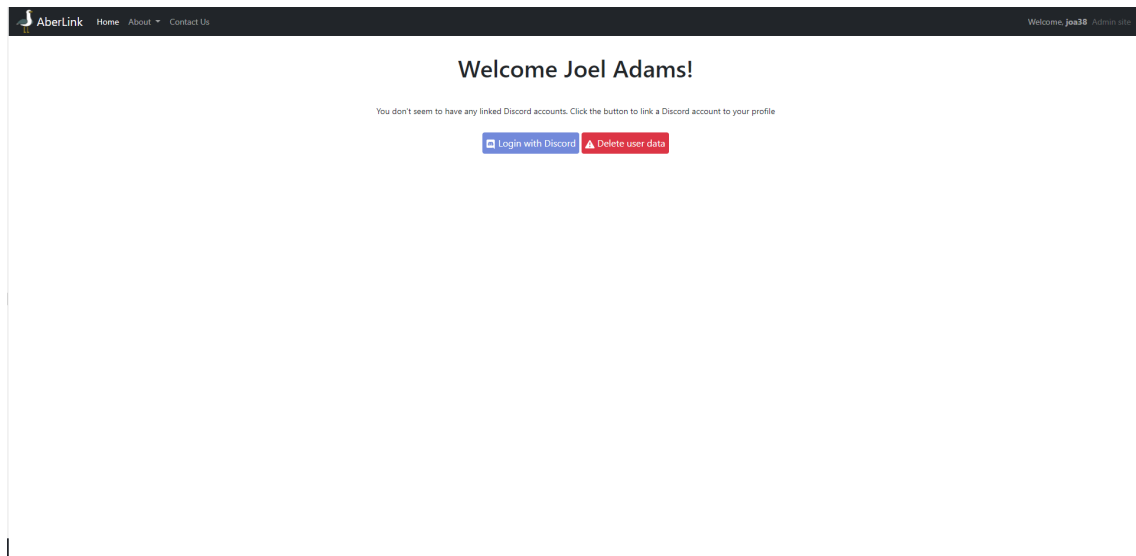


Figure 3.1: Final website with 0 linked Discord accounts

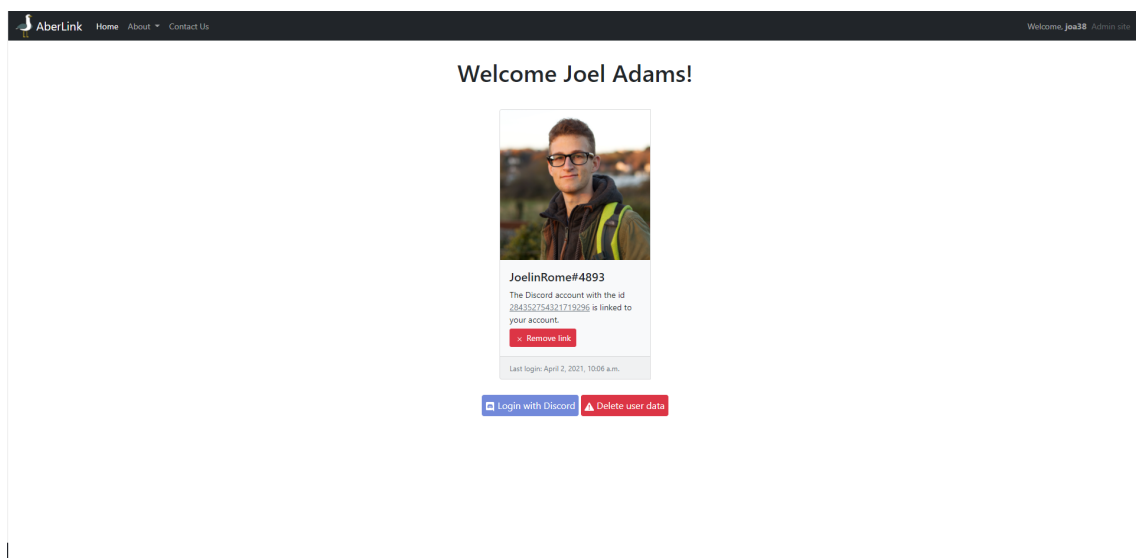


Figure 3.2: Final website with 1 linked Discord accounts

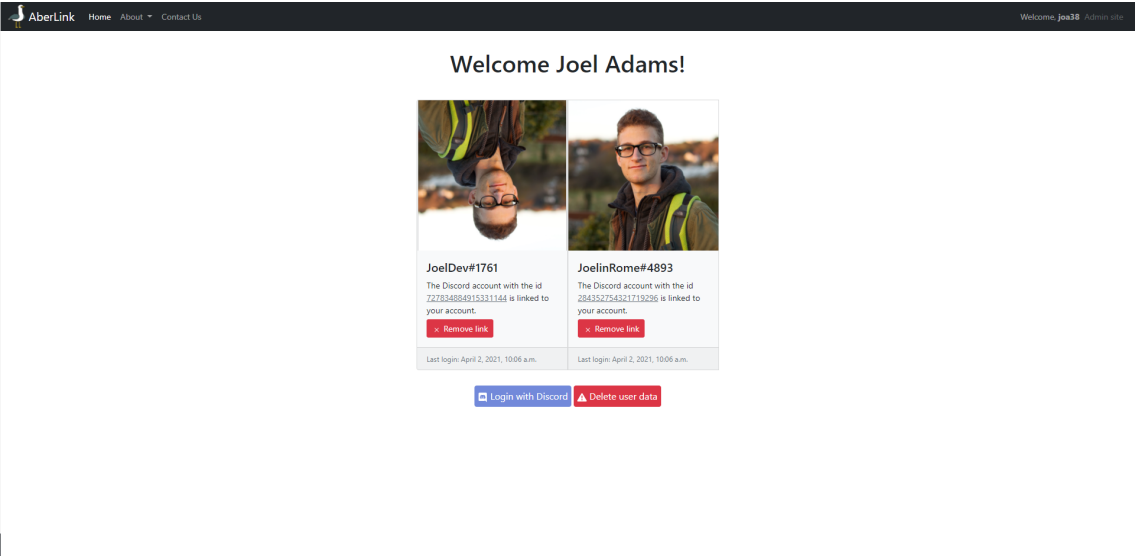


Figure 3.3: Final website with 2 linked Discord accounts

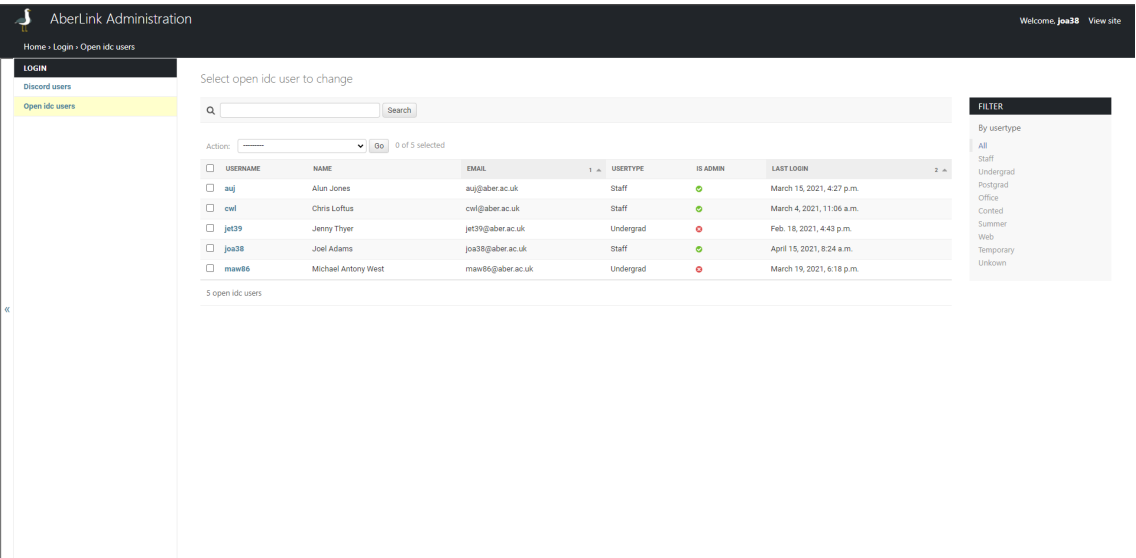


Figure 3.4: Final website Admin page for Aber accounts

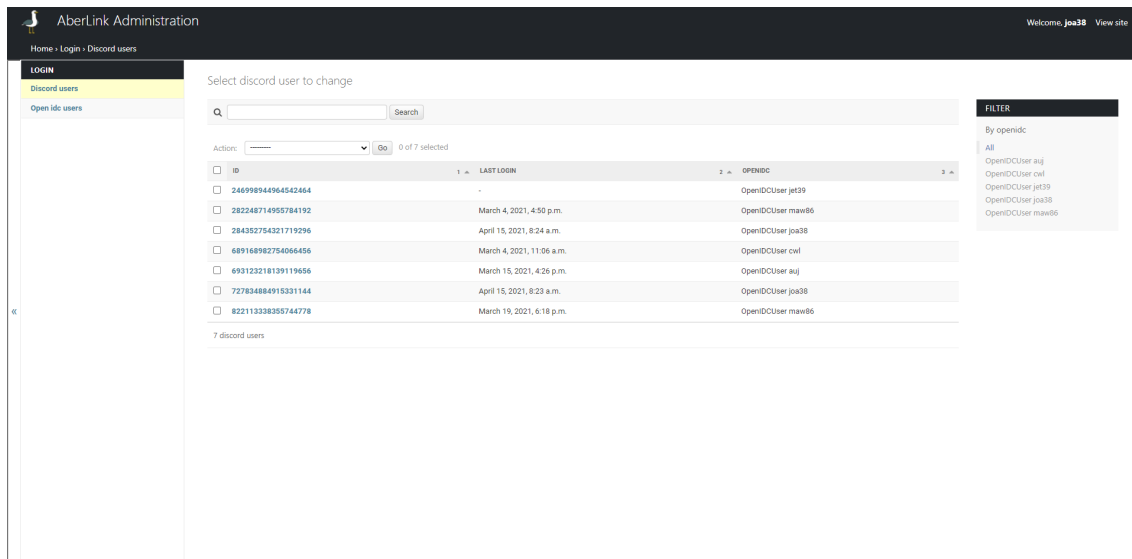


Figure 3.5: Final website Admin page for Discord accounts

### 3.1.2 Discord Bot

The Discord bot (AberLink) implementation is exactly as described in the section 2.3 of the design. AberLink uses the Python library `Discord.py` [7] to make calls to and from the Discord API and to interact with the database it uses the Python library `Psycopg2` [16]. Both of these pieces of software are open-source and free to use in personal projects.

#### 3.1.2.1 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` [23]. 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.

```
1 # load the private discord token from .env file.
2 load_dotenv()
3 TOKEN = os.getenv('DISCORD_TOKEN')
4 WEBSITE = os.getenv('WEBSITE_URL')
5
6 # Initialise the database connection
7 PostgreSQL.connect()
8
9 # Initialise the Bot object with an accessible help Command
   object
10 helpCommand = DefaultHelpCommand()
11
12 bot = commands.Bot(
13     command_prefix="!",
14     help_command=helpCommand,
15     intents=discord.Intents.all(),
16 )
17
18 bot.run(TOKEN)
```

Figure 3.6: 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 in important section called `commands.Bot` that tells the bot key information such as which prefix to use that determines how a command is called. E.g. `!` means that a command is invoked using `!<command_name>`, by changing this then the command is invoked using that prefix. It is also not limited to a single character so can be `AberLink` and then command would be invoked using `AberLink<command_name>`.

### 3.1.2.2 Discord Command Example

The bot creates commands that can be used in a Discord server using the `Discord.py` [7] 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.

```

1 @commands.command(aliases=['p'])
2 async def ping(self, ctx: Context):
3     """
4     Returns latency and response time of Discord and the database
5     """
6     start_time = time()
7     message = await ctx.send(f' pong `DWSP latency: {str(round(ctx
8         .bot.latency * 1000))}ms`')
9     end_time = time()
10    db_latency = PostgreSQL.get_connection_latency()
11    db_poll = PostgreSQL.get_polling_status()
12    await message.edit(content=f' pong \n{emojis["discord"]} `DWSP
13        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 3.7: Example of AberLink's ping command code

The commands response is then sent to Discord and I have included an example of the command output below.

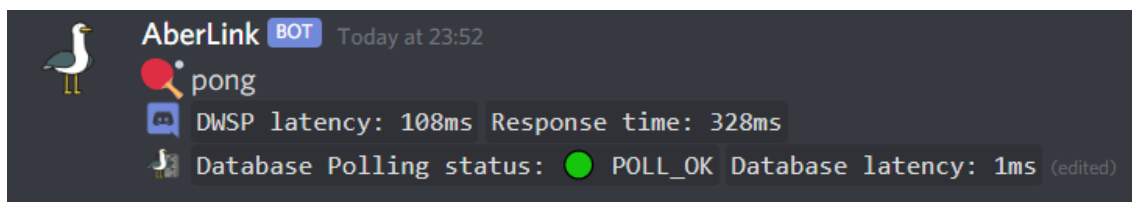


Figure 3.8: Example of ping command in Discord

### 3.1.2.3 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.

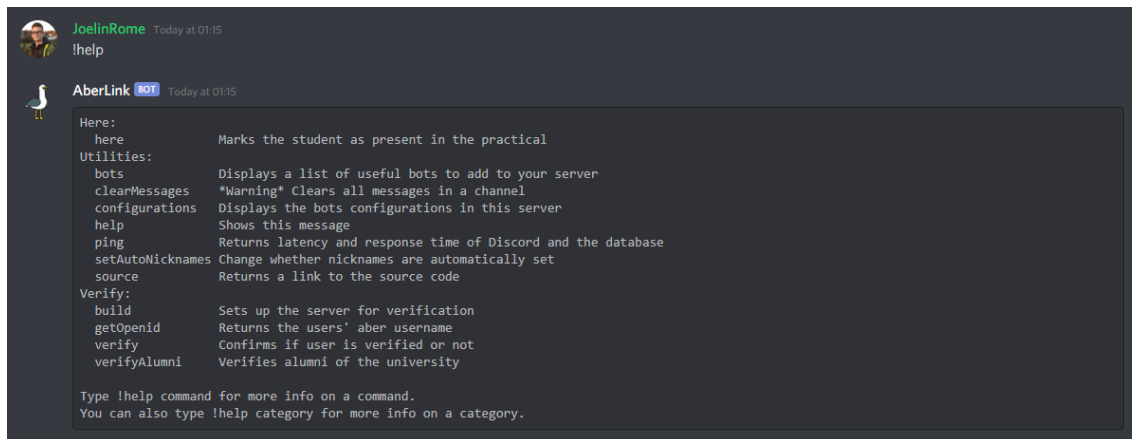


Figure 3.9: Example of Discord help command to display list of Discord commands for AberLink

### 3.1.2.4 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 don't exist in the case where two Discord bots use the same command prefix but have different commands.

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

Figure 3.10: Example of AberLink's `on_command_error()` checking statements

### 3.1.2.5 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 [24] that creates a file stored in the same directory as the Python file. This creates a persistent storage database so I have included a feature in



the bot that allows lecturers to enable or disable the automatic nickname feature.

### 3.1.3 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.

```

1 class OpenIDCUserManager(BaseUserManager):
2     def create_user(self, user, password=None):
3         new_user = self.model(
4             username = user['OIDC_CLAIM_preferred_username'],
5             name = user['OIDC_CLAIM_name'],
6             email = user['OIDC_CLAIM_email'],
7             usertype = user['OIDC_CLAIM_usertype']
8         )
9         if user['OIDC_CLAIM_usertype'] == "staff":
10             new_user.is_admin = True
11             new_user.save(using=self._db)
12             return new_user
13
14 class OpenIDCUser(AbstractBaseUser):
15     objects = OpenIDCUserManager()
16
17     class usertypes(models.TextChoices):
18         STAFF = 'staff'
19         UNDERGRAD = 'undergrad'
20         POSTGRAD = 'postgrad'
21         OFFICE = 'office'
22         CONTED = 'conted'
23         SUMMER = 'summer'
24         WEB = 'web'
25         TEMPORARY = 'temporary'
26         UNKNOWN = 'unknown'
27
28     id = models.AutoField(auto_created=True, primary_key=True,
29                           serialize=False)
30     username = models.CharField(max_length=40)
31     name = models.CharField(max_length=300)
32     email = models.CharField(max_length=30)
33     usertype = models.CharField(max_length=50, choices=usertypes
34                               .choices)
35     last_login = models.DateTimeField(null=True)
36     password = None
37     is_active = models.BooleanField(default=True)
38     is_admin = models.BooleanField(default=False)
39
40     USERNAME_FIELD = 'id'
41     REQUIRED_FIELDS = ['username', 'name', 'email', 'usertype']

```

Figure 3.11: 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. 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. I have included information below the figure to explain what the tables mean.

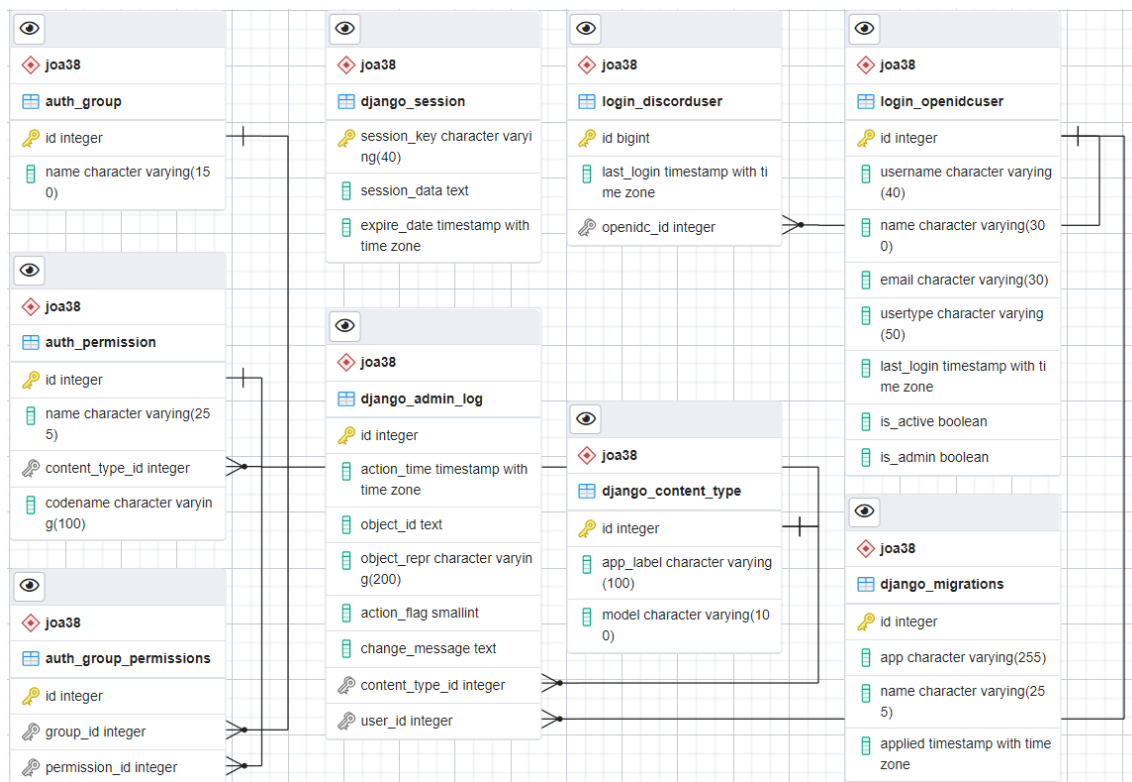


Figure 3.12: Final Entity Relationship diagram for database

- **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

## 3.2 Configuration and Setup

After completion of this project there is a plan to setup this service permanently for the department so I have created a folder called *config* containing information on how to set up this project. This folder contains a `README.md` file that explains how and what is needed to be configured to get the system up and running properly. I have also added 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 [7].

## 3.3 Unforeseen Issues

An issue that I encountered was with creating the custom user model in Django that would have been used to model the database. The documentation and videos I found online about implementing user models were rather cryptic and difficult to understand, however I eventually found a video explaining how to implement a good custom user model. Once this was completed I realised that Django isn't happy with modelling the primary key of a table using a char so I had to switch 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.

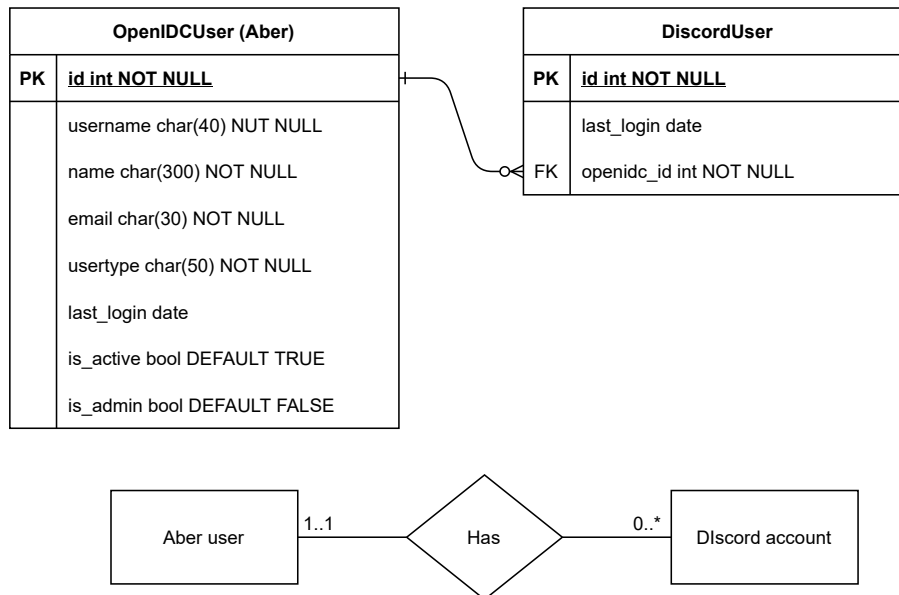


Figure 3.13: Updated Entity relationship diagram for database

## 3.4 Review

### 3.4.1 Review Against Planned Requirements

Most of the planned requirements haven't 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 it's complexity and make it much more difficult to maintain. I have also not added Welsh language support as I do not speak the language fluently nor understand it.

### 3.4.2 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 uni container. Lots of meetings to discuss what data I can use 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 [6].
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

I've included a graph below of my GitLab commits over time below to help backup my table described above.

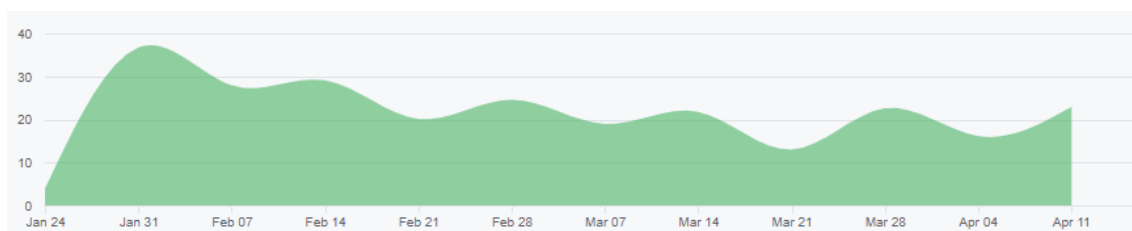


Figure 3.14: GitLab graph of commits over time

## Chapter 4

# Testing

As previously mentioned in section 1.3 I have used a Feature Driven Development (FDD). I worked in one week iteration windows to build and test the software as I went; this included all of the testing sections mentioned below apart from automated testing. This was completed at the end of the project instead.

The Discord.py Python framework doesn't 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 into it so I have tried to use some of these and they are detailed further on this section. Django tests are used to cover both the website and the database.

### 4.1 Sample Data

I have considered using sample data for my project however this is very difficult because the OpenID Connect [5] system used to authenticate and log users in relies on a proper database. This would mean I would have to mock the database which would take considerable work to create. The only workaround that I have found for this system has been to rely on automated and user testing.

### 4.2 Automated Testing

Automated testing has been difficult for this project as mentioned above. I was originally planning on including automated testing and building DevOps frameworks in Git, however as I ended up using the universities GitLab 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 and I have used these tests where I deemed most appropriate. All of the following tests can be found here `src\AberLinkAuthentication\login\tests.py`.

I've created two testing classes here; 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:

```
1 def test_url_discord_redirect(self):
2     url = reverse('Discord-response')
3     self.assertEqual(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

```
1 def test_model_openidc_user_staff(self):
2     self.openidc_user2 = OpenIDCUser.objects.create(
3         username="abc123",
4         name="Bob Ross",
5         email="abc123@aber.ac.uk",
6         usertype="staff"
7     )
8     self.assertEqual(self.openidc_user2.username, "abc123")
9     self.assertEqual(self.openidc_user2.name, "Bob Ross")
10    self.assertEqual(self.openidc_user2.email,
11        "abc123@aber.ac.uk")
12    self.assertEqual(self.openidc_user2.usertype, "staff")
13    self.assertEqual(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 I used ApacheBenchmark [25] which is a built in Apache2 [3] library used to request a webpage and measure response time. To test this I temporarily disabled the OpenID Connect [5] authentication system so that HTML gets served to tge incoming requests. Due to the nature of my websites implementation I cannot query the main page because the user is not authenticated however I can query one of the subpages that doesn't require authentication. I used the following line of code to complete the request and uses 1000 requests with 100 concurrent threads to attempt to tank website performance:

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

From this command I received the following output 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 with the department I can't see more than 300 requests happening at the exact same time.

```
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.

When building the UI of the website I decided to rely on the CSS library Bootstrap [22] as it is built from the ground up with responsive design in mind. This meant that website would scale very nicely and uniformly depending on screen size. I have however conducted testing and scaling of all the website pages on mobile devices, iPads and laptops with screens ranging from 11in to 40in using Safari, Chrome, Edge and Firefox. I have also tested the website on HTML validation website <https://validator.w3.org/> and fixed the warnings and errors where possible.

### 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 design choice that I made because I realised that this 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 couldn't 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 3.1.3 I didn't account for Django creating its own tables used for running the website.

I also didn't 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 3.3.

### 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 3.4.2 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] 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.

- [7] 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.

- [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. Joelin-Rome#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] Blaine Cook, “OAuth 2.0,” accessed April 2021. [Online]. Available: <https://oauth.net/2/>

OAuth 2.0 is the industry-standard protocol for authorization.

- [13] 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.

- [14] —, “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.

- [15] graffatcolmingov, Lukasa and nateprewitt, “Requests: HTTP for Humans™,” accessed April 2021. [Online]. Available: <https://docs.python-requests.org/en/master/>

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

- [16] Various authors, “Pyscopg2,” accessed April 2021. [Online]. Available: <https://www.pyscopg.org/docs/>

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

- [17] 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.

- [18] Evan Wallace & Dylan Field, “Figma,” accessed April 2021. [Online]. Available: <https://www.figma.com/>

Figma is a design website used to build application mockups.

- [19] JetBrains, “PyCharm,” accessed April 2021. [Online]. Available: <https://www.jetbrains.com/pycharm/>

A professional IDE for Python built by JetBrains.

- [20] 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.

- [21] Electronic Frontier Foundation (EFF), “Certbot,” accessed April 2021. [Online]. Available: <https://certbot.eff.org/>

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

- [22] Various authors on Github, “Bootstrap 5.0,” accessed April 2021. [Online]. Available: <https://getbootstrap.com/>

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

- [23] 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.

- [24] 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.

- [25] Apache Software Foundation, “Apache HTTP server benchmarking tool (ApacheBench),” accessed April 2021. [Online]. Available: <http://httpd.apache.org/docs/current/programs/ab.html>

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



- [26] 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.

- [27] aalbrecht, “sqlparse,” accessed April 2021. [Online]. Available: <https://pypi.org/project/sqlparse/>

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

- [28] Various authors, “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 web servers and frameworks.

- [29] 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.

- [30] 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.

- [31] 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 unmodified.

- **Pipenv** [14] - 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** [13] - 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** [21] - This package was only briefly required to setup HTTPS on the website using Let's Encrypt [29]. 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** [16] - 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 [30] and python-dev [31]. This library is licensed under the GNU Lesser General Public License.
- **Discord.py** [7] - 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** [15] - 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** [26] [27] [28] - 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 <i>Aber-LinkAuthentication/login/views.py</i>
FR1.1	Yes	OpenID Connect only authenticates aber users.
FR1.2	Yes	function <i>openidc_response()</i> gets metadata from OpenID Connect login and usertype.
FR1.3	Yes	<i>openidc_response()</i> 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 <i>Aber-LinkAuthentication/login/views.py</i>
FR2.1	Yes	<i>discord_oauth2_redirect()</i> gets metadata by getting authorization from url code and calls <i>exchange_code()</i> to query Discord API for metadata on user.
FR2.2	Yes	<i>discord_oauth2_redirect()</i> contains a line to get the <i>openidc_user</i> account and link the Discord account to it using the foreign key relationship.
FR2.3	Yes	Tested connecting 3 Discord accounts to the same <i>openidc_user</i> with no errors.
FR3	Partially	Most requirements passed. Using functions from file <i>Aber-LinkAuthentication/login/views.py</i>
FR3.1	Yes	<i>openidc_response()</i> displays main webpage along with information on the linked user from the metadata.
FR3.2	Yes	<i>openidc_response()</i> displays linked Discord accounts linked to <i>Openidc_user</i> account.
FR3.3	Yes	<i>openidc_response()</i> contains function <i>get_discord_users()</i> 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 <i>Aber-LinkAuthentication/login/admin.py</i>
FR4.1	Yes	When user is logged in Django checks that they have <i>is_admin</i> 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	<i>OpenIDCAdmin()</i> displays table containing aber (openidc) accounts.
FR4.3	Yes	<i>DiscordAdmin()</i> displays table containing Discord accounts and their linked openidc accounts.
FR4.4	Yes	Both <i>DiscordAdmin()</i> and <i>OpenIDCAdmin()</i> contain the line <i>readonly_fields</i> 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 <i>Aber-LinkDiscord/cogs/verify.py</i>
FR5.1	Yes	<i>build()</i> 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	<i>verify.py/verify()</i> allows uses to call a command and manually verify, alternatively users are automatically verified when they join using <i>on_member_join()</i> .
FR5.3	Yes	<i>verifyAlumni()</i> is a callable command that tells alumni how to verify by send an email to <a href="mailto:afc@aber.ac.uk">afc@aber.ac.uk</a> .
FR5.4	Yes	Both <i>verify()</i> and <i>on_member_join()</i> contain a function called <i>check_shelve_file()</i> that checks to see if the server has auto nickname enabled.
FR6	Yes	Passed all requirements. Using functions from file <i>Aber-LinkDiscord/cogs/here.py</i>
FR6.1	Yes	<i>here()</i> calls functions from <i>AberLinkDiscord/cogs/db.py</i> to do reverse lookup using discord ID to get Aber username.
FR6.2	Yes	<i>here()</i> gets information as described in FR6.1 and sends it to the university API endpoint <a href="https://integration.aber.ac.uk/joa38/submit.php">https://integration.aber.ac.uk/joa38/submit.php</a> . 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 <i>Aber-LinkDiscord/cogs/</i>

FR7.1	Yes	File called <i>shelve.file.shelve.db</i> exists or is created and is used in function <i>verify.py/verify()</i> to check whether to give nickname automatically.
FR7.2	Yes	Function called <i>utilities.py/configurations()</i> displays configurations available for the current server.
FR7.3	Yes	Function called <i>utilities.py/ClearMessages()</i> clears the last 14 days of messages.
FR7.4	Yes	Function called <i>utilities.py/source()</i> displays message containing link to source code and website.
FR7.5	Yes	Function called <i>utilities.py/bots()</i> displays message with useful bots that can be added to server.
FR7.6	Yes	Function called <i>utilities.py/ping()</i> displays bot and message edit latency. It also gets information on the database connection from functions in <i>db.py</i> .
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 <i>Aber-LinkDiscord/cogs/db.py</i>
FR8.1	Yes	<i>connect()</i> connects to the database, returns information on connection and defines global variable for connection.
FR8.2	Yes	<i>try_connection()</i> reconnects to database if connection is lost.
FR8.3	Yes	<i>get_discord_user()</i> gets Discord information from database using Discord ID.
FR8.4	Yes	<i>get_openid_user()</i> gets information from database about aber account using their generated ID.
FR8.5	Yes	<i>get_connection_status()</i> , <i>get_polling_status()</i> and <i>get_connection_latency()</i> get information from database.

Table C.1: Functional Requirements testing table