

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.

NameJoel 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.

NameJoel Luca Adams.....

Date

Acknowledgements

I'd like to thank Dan Monaghan for giving me the original inspiration for this project and for helping me along the way with understanding key concepts.

I am grateful to have been working along side Dr. Neal Snooke for this project. Over the last year we have worked collaboratively on multiple Discord bots and improved the Uni's online Discord presence. I'd also like to thank him for allowing me to re-create and use the Discord bots that he created and have been the main motivation behind this project.

I'd like to thank Alun Jones for his continued support throughout this project and his resourcefulness in providing instructions on how to setup and use specific uni services.

I'd like to thank Leslie Johns for having setup an API endpoint that has been crucial in getting this project off the ground. He was very fast at setting this up and provided good documentation along with testing links which have helped in the testing section of this document.

Abstract

AberLink contains 2 components; this 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 foreign keys to link the uni and Discord accounts together. A 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]. These bots verified students' Discord accounts and helped to mark attendance during practicals respectively but have been greatly modified.

Contents

1	Background & Objectives	1
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
2	Design	6
2.1	Overall Architecture	6
2.2	Website Architecture and Design	7
2.3	Discord Bot Architecture and Design	9
2.3.1	Database Interaction	10
2.3.2	Complicated Behaviours	11
2.4	Database design	12
2.5	User Interface	13
2.5.1	Website	13
2.5.2	Discord bot	14
3	Implementation	16
3.1	Code Implementation & Third-Party libraries	16
3.1.1	Website Building	16
3.1.2	Discord Bot	16
3.2	Unforeseen Issues	17
3.3	Review Against Planned Requirements	17
4	Testing	19
4.1	Sample Data	19
4.2	Automated Testing	19
4.2.1	Unit Tests	19
4.2.2	Stress Testing	21
4.3	User Interface Testing	22
4.4	User Testing	22
4.5	Integration Testing	22
5	Evaluation	23
	Annotated Bibliography	24
	Appendices	25
A	Third-Party Code and Libraries	27
B	Ethics Submission	28
C	Code Examples	29
3.1	Random Number Generator	29

List of Figures

2.1	Architecture diagram of the overall system	6
2.2	Tree diagram showing overall folder structure	7
2.3	Architecture diagram of website	7
2.4	Website flowchart for authentication of accounts	8
2.5	Architecture diagram of Discord bot	9
2.6	Tree diagram showing Discord file structure	9
2.7	Discord UML diagram	10
2.8	Sequence diagram for Discord bot to database	11
2.9	Entity relationship diagram for database	12
2.10	Website mock-up for 0 users	13
2.11	Website mock-up for 1 user	14
2.12	Website mock-up for 2 users	14
2.13	Discord embed example for getting users information	15
2.14	Discord embed example for source command	15
3.1	Updated Entity relationship diagram for database	17
4.1	Django URL render test	20
4.2	Django database render test	20

List of Tables

2.1	Aberystwyth user table example	12
2.2	Discord user table example	13

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 the student can use 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 much of the spike work required to form the Project Outline and send it out to IS and CS-support. The 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 is not listed below and I encourage that you have a look there too.

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. Linux 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 support from CS-support and documentation available online.

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. This helped me to build up 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 that would act as the primary login system and a table 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 support 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 sign users in who are only on the uni network using OpenID Connect [5].
 - How to build the attendance API so that it is secure and only marks students for current practicals.
 - Setting up PSQL [6] and how to 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’.
- **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 someone 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 with the database or website.

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 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 (see later in this document for details).
- **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 chapter of this document. There is also a board of issues on the GitLab page that can be used to review the timestamps and work completed.

Chapter 2

Design

2.1 Overall Architecture

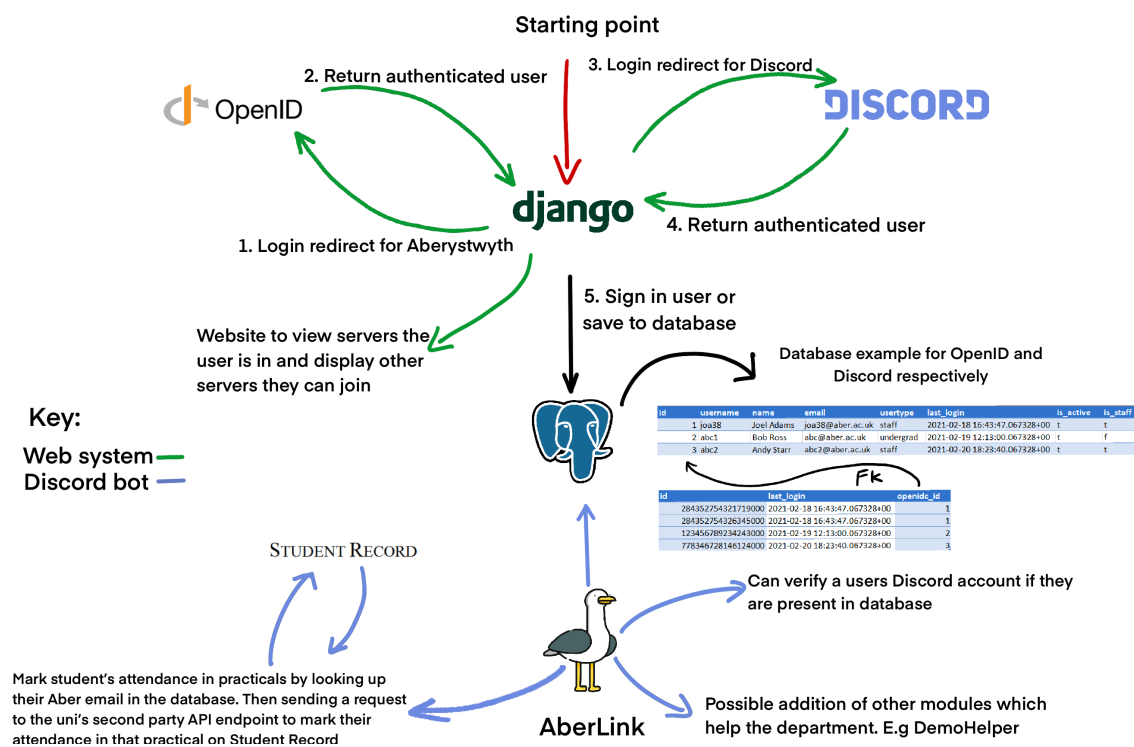


Figure 2.1: Architecture diagram of the overall system

Below is a diagram displaying the overall file structure used for the project and was inspired by the second year group project. It has a config folder which describes how to setup the whole project, a folder called dev which contains all of the projects spikework and a folder called docs that contains the latex documents required for this project such as this document. There is also a folder called src that contains two sub folders AberLinkAuthentication and AberLinkDiscord. These two folders are in charge of the website and

Discord bot respectively and have been split up as the underlying architecture for each one is vastly different and helps greatly with code maintainability.

```
aberslink
├── config
├── dev
├── docs
├── src
│   ├── AberLinkAuthentication
│   └── AberLinkDiscord
```

Figure 2.2: Tree diagram showing overall folder structure

2.2 Website Architecture and Design

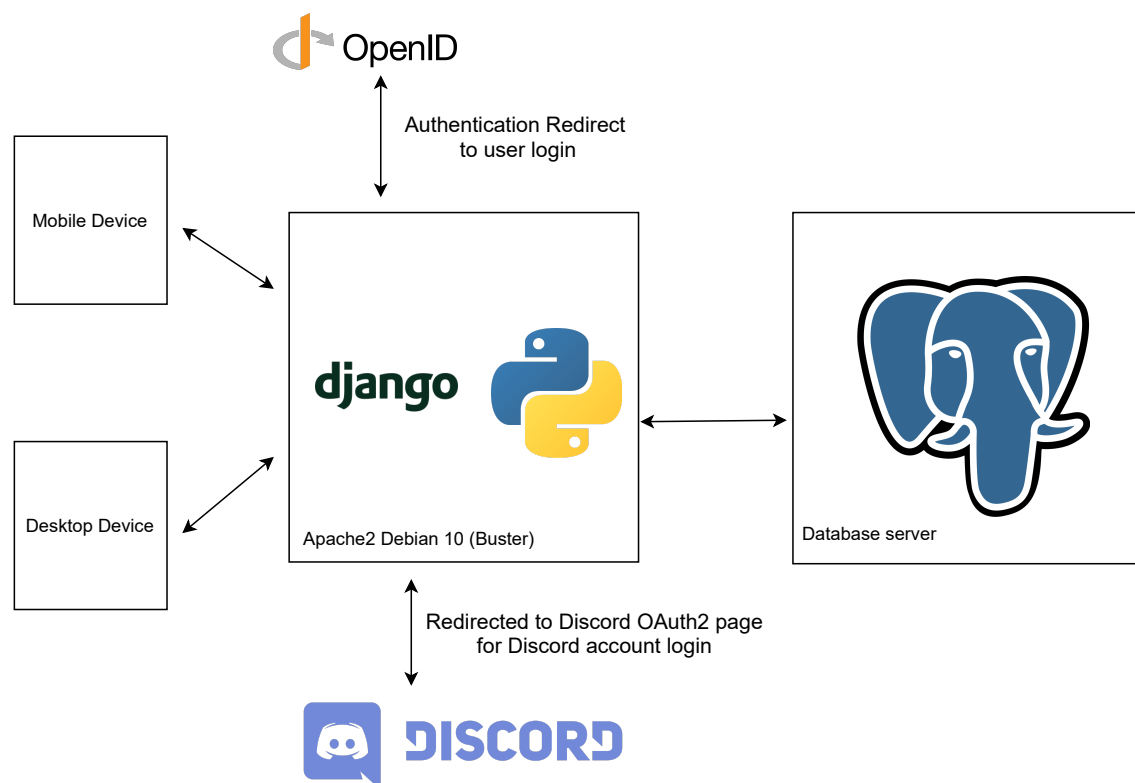


Figure 2.3: Architecture diagram of website

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.

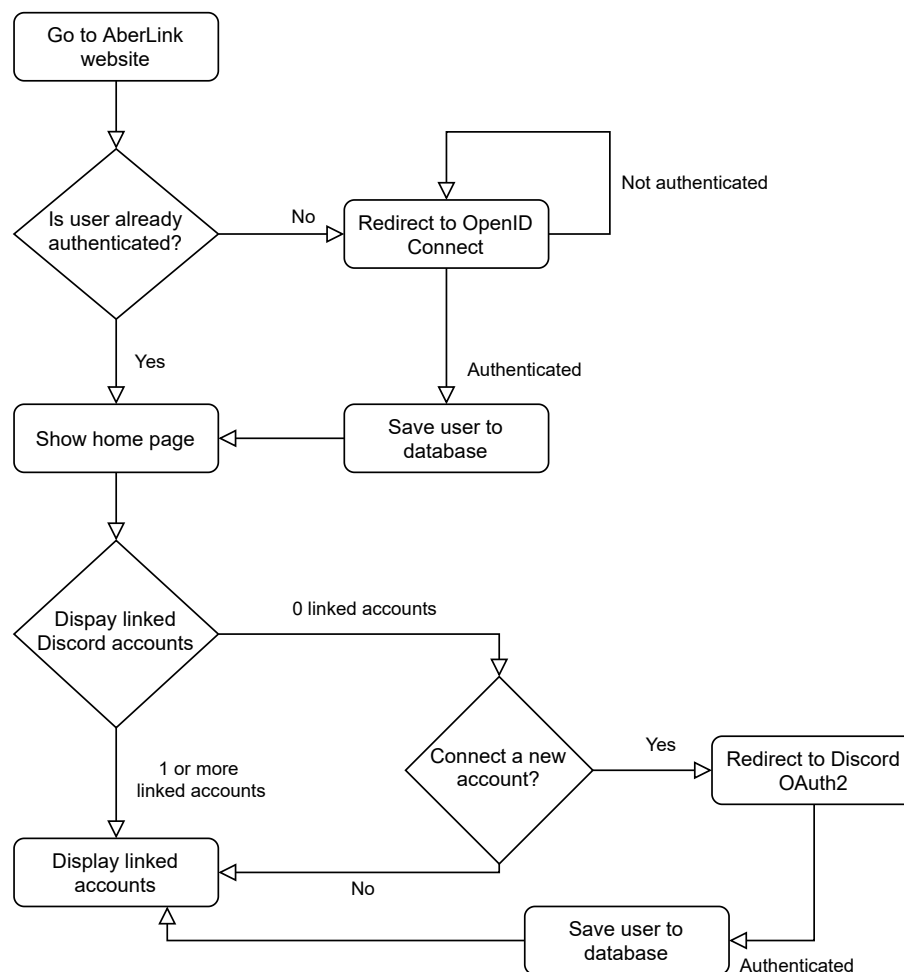


Figure 2.4: Website flowchart for authentication of accounts

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

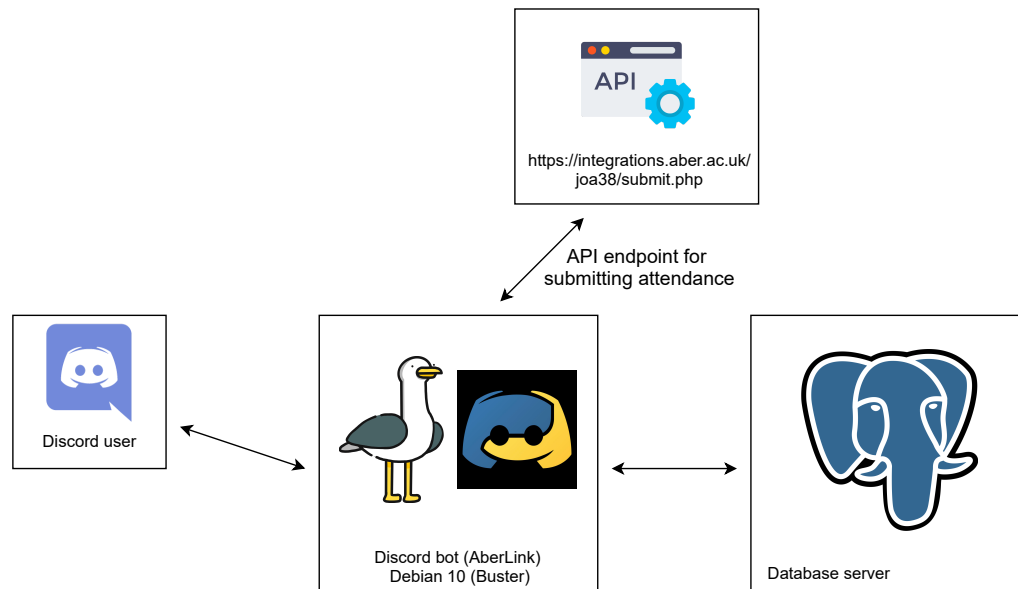


Figure 2.5: Architecture diagram of Discord bot

The Discord bot (AberLink) is interactable through the Discord application available on mobile, desktop and 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. The basic premise however is that it can access the database to retrieve information about a user; it never saves any information to the database however. The file structure goes as follows:

```

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

```

Figure 2.6: 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) but instead separated out 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, requests and the version of Python required to run the bot. It can be initialised using the Python virtual environment pipenv.

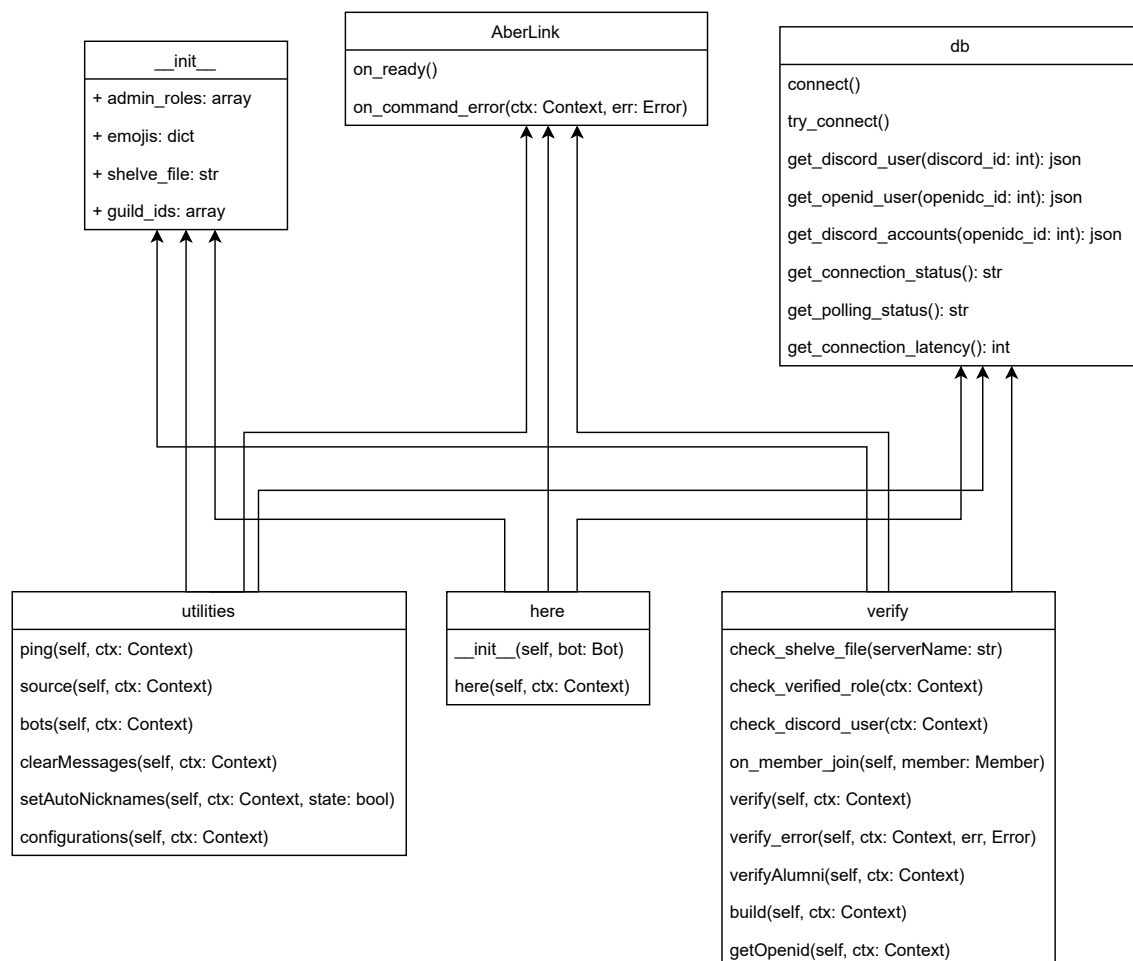


Figure 2.7: Discord UML diagram

2.3.1 Database Interaction

Below is a sequence diagram explaining how AberLink connects to database and sends requests back and forth. It uses the Python library psycopg2 to connect and send SQL requests back and forth; the detail 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.

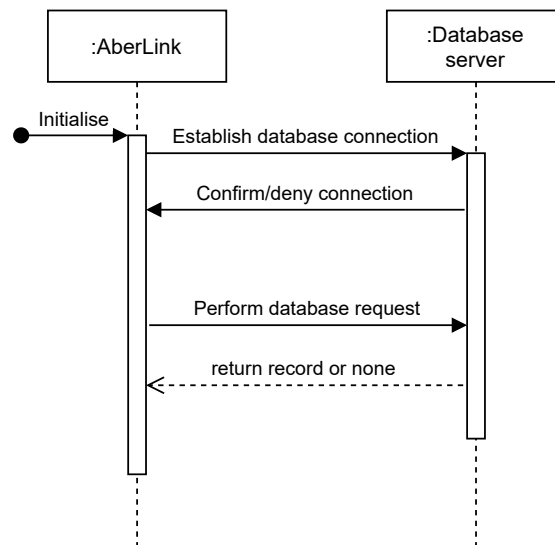


Figure 2.8: Sequence diagram for Discord bot to database

2.3.2 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 `/cogs/verify.py`. This function and command is used in discord to configure the server for verification and uses the following steps:

1. Begins by simulating that the bot is typing
2. Then searches for the `@everyone` and verify roles, the `verify_channel`
3. The `@everyone` role is then stripped of it's permissions entirely so as to stop channel viewing
4. If the verify role doesn't exist then it is created and it is set with all the default permissions that `@everyone` used to have
5. The bot is then given the verify role so that they can view the server channels
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 then adds the ability for the `@everyone` role to 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
9. Finally once completed the bot stops typing and sends the message "authentication complete"

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.

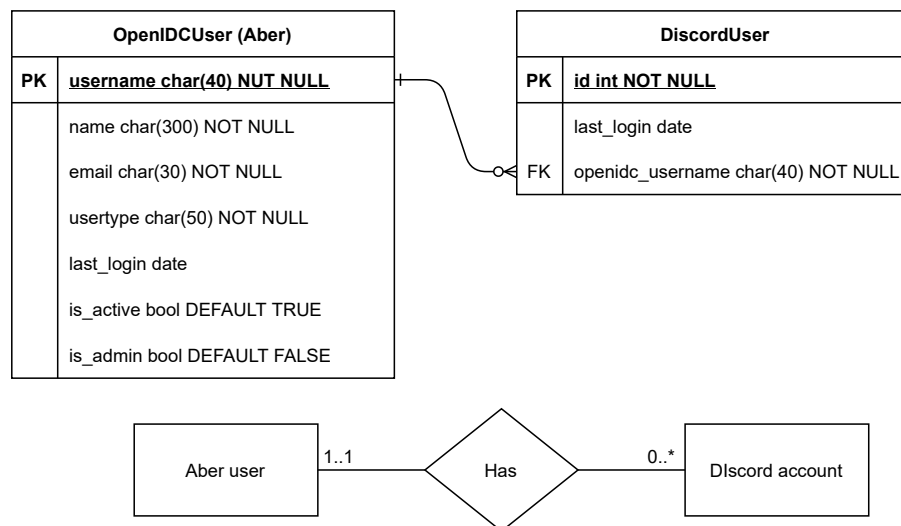


Figure 2.9: Entity relationship diagram for database

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. 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.

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 help with possible problems down the line and lastly a `openidc_id` that is a foreign key from the **OpenIDCUser**'s id.

Below is an example of the data stored in the database.

<u>id</u>	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 2.1: 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 2.2: Discord user table example

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 and are pictured below.

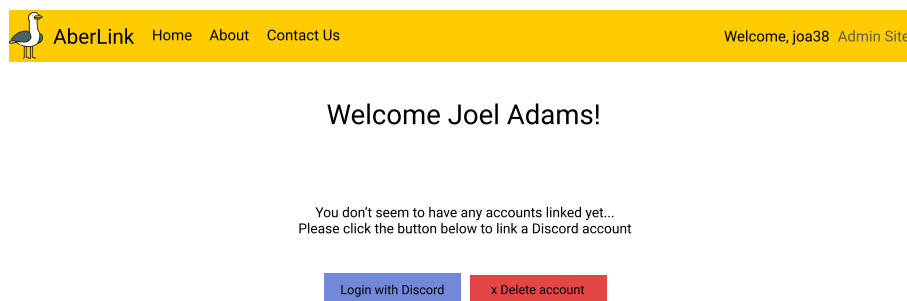


Figure 2.10: Website mock-up for 0 users

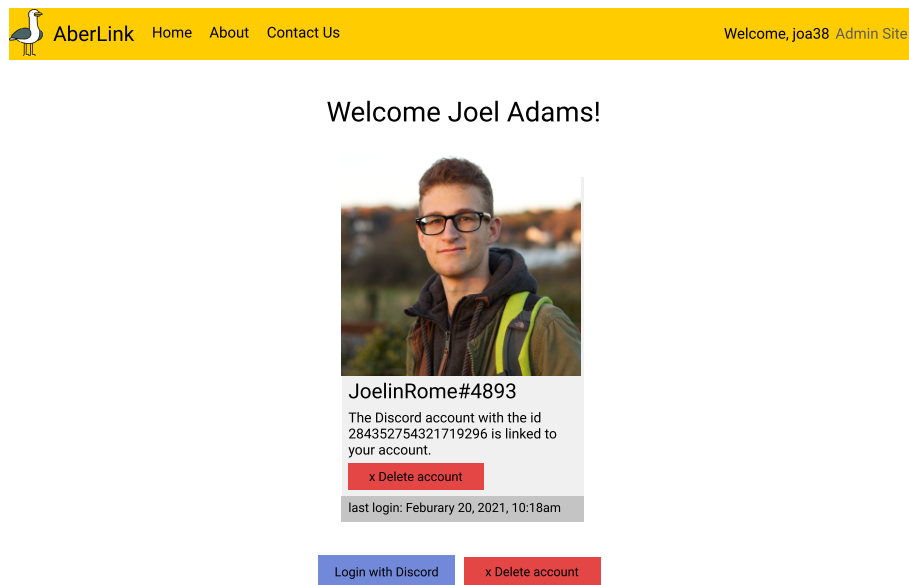


Figure 2.11: Website mock-up for 1 user

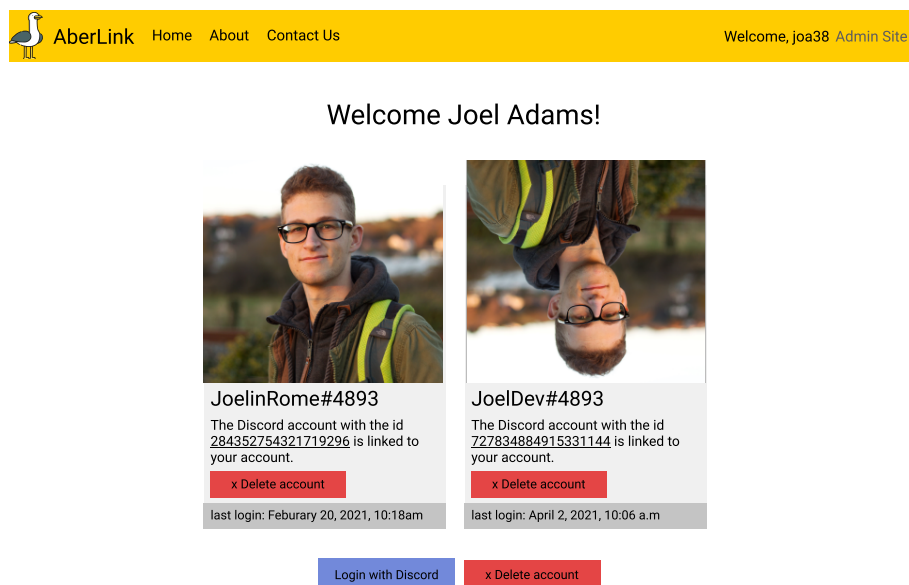


Figure 2.12: Website mock-up for 2 users

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.

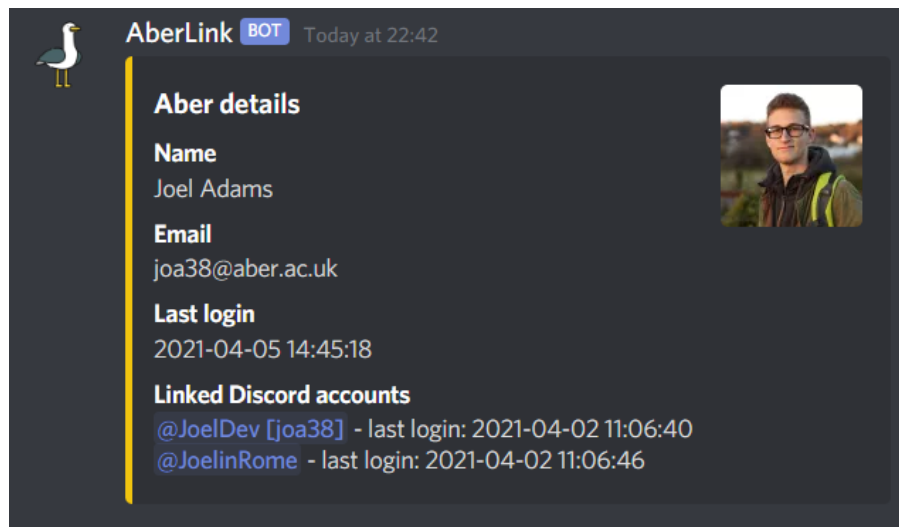


Figure 2.13: Discord embed example for getting users information

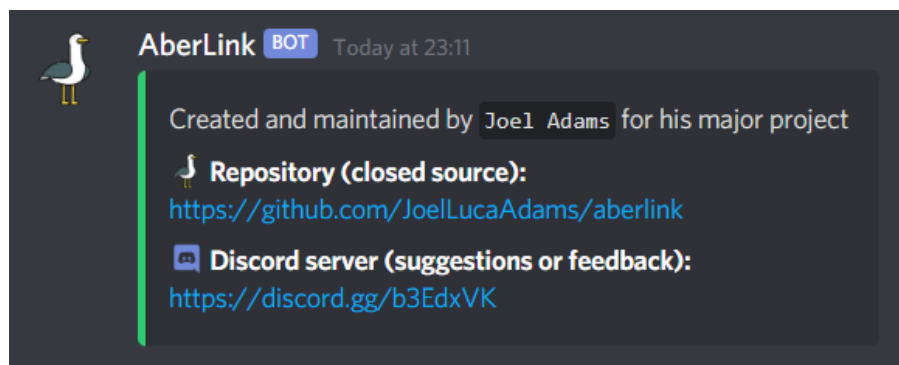


Figure 2.14: Discord embed example for source command

Chapter 3

Implementation

3.1 Code Implementation & Third-Party libraries

As seen below the Website and Discord bots 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.

3.1.1 Website Building

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 [13] 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 (Django Trademark License Agreement). 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 and it's respective services.

The website pages also use a third party library called bootstrap [14] that is used to generate responsive mobile-first CSS on the website.

3.1.2 Discord Bot

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 [15]. Both of these

pieces of software are open-source and free to use in personal projects.

3.2 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.

3.3 Review Against Planned Requirements

Most of the planned requirements haven't changed during implementation however as discussed in the section above 3.2 the database model has changed slightly and the updated diagram is included below.

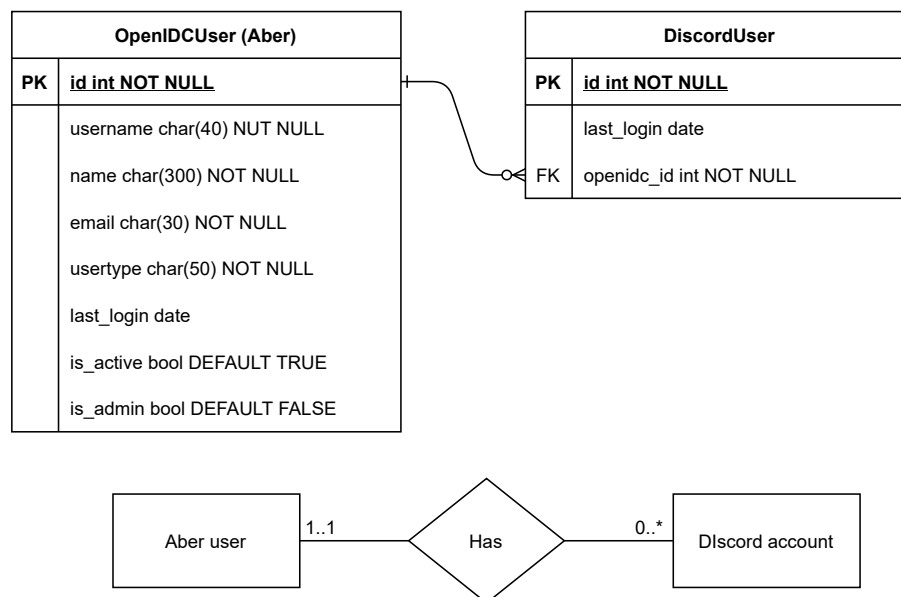


Figure 3.1: Updated Entity relationship diagram for database

The planned requirements also included a section called **Interface for lecturers and students on website** has also only been partially fulfilled. I have completed the second part of this task which was to create admin pages to view the connected staff and students however the first part where users can view what servers they are in is missing. 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.

In the final section of the planned requirements **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.

Chapter 4

Testing

As previously mentioned in section 1.3 I have used a Feature Driven Development (FDD) for this project to develop and test features as I developed the project. 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 deemed it to be impossible as due to the implementation of authentication. As stated previously the website uses the OpenID Connect [5] system to authenticate and log users in; as I don't have a dummy dataset that can authenticate with that system there is no other way for me to authenticate users and perform sample testing. 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 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:

```
def test_url_discord_redirect(self):
    url = reverse('Discord-response')
    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

```
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.assertEqual(self.openidc_user2.username, "abc123")
    self.assertEqual(self.openidc_user2.name, "Bob Ross")
    self.assertEqual(self.openidc_user2.email,
"abc123@aber.ac.uk")
    self.assertEqual(self.openidc_user2.usertype, "staff")
    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 ApacheBench [16] 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 to do no testing on that front.

When I came to building the UI of the website I decided to rely on the CSS library Bootstrap [14] 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 helps to find a few errors that I had previously missed in the code.

4.5 Integration Testing

Following the FDD approach I ran integration tests on this project as it contained many working parts. I used the Figure 2.1 as a reference guide when performing these tests.

For the AberLink Discord bot I only had two components to test; the bot requests and responses from the database for the commands I had. I used some user tests to verify that all these commands worked and attempted to account for all possible scenarios.

Chapter 5

Evaluation

Examiners expect to find a section addressing questions such as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

The questions are an indication of issues you should consider. They are not intended as a specification of a list of sections.

The evaluation is regarded as an important part of the project report; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things in the work and aspects of the work that could be improved. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

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] 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.

- [14] 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.

- [15] 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.

- [16] —, “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.

Appendices

The appendices are for additional content that is useful to support the discussion in the report. It is material that is not necessarily needed in the body of the report, but its inclusion in the appendices makes it easy to access.

For example, if you have developed a Design Specification document as part of a plan-driven approach for the project, then it would be appropriate to include that document as an appendix. In the body of your report you would highlight the most interesting aspects of the design, referring your reader to the full specification for further detail.

If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification. Perhaps you use stories to keep track of the functionality and the 'future conversations'. It might not be relevant to include all of those in the body of your report. Instead, you might include those in an appendix.

There is a balance to be struck between what is relevant to include in the body of your report and whether additional supporting evidence is appropriate in the appendices. Speak to your supervisor or the module coordinator if you have questions about this.

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what your original work is and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

The following is an example of what you might say.

Apache POI library - The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client's existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [?]. The library is released using the Apache License [?]. This library was used without modification.

Include as many declarations as appropriate for your work. The specific wording is less important than the fact that you are declaring the relevant work.

Appendix B

Ethics Submission

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.

Appendix C

Code Examples

For some projects, it might be relevant to include some code extracts in an appendix. You are not expected to put all of your code here - the correct place for all of your code is in the technical submission that is made in addition to the Project Report. However, if there are some notable aspects of the code that you discuss, including that in an appendix might be useful to make it easier for your readers to access.

As a general guide, if you are discussing short extracts of code then you are advised to include such code in the body of the report. If there is a longer extract that is relevant, then you might include it as shown in the following section.

Only include code in the appendix if that code is discussed and referred to in the body of the report.

3.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [?].

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
```

```

#define RNMX (1.0 - EPS)

double ran2(long *idum)
{
    /*-----*/
    /* Minimum Standard Random Number Generator      */
    /* Taken from Numerical recipies in C             */
    /* Based on Park and Miller with Bays Durham Shuffle */
    /* Coupled Schrage methods for extra periodicity   */
    /* Always call with negative number to initialise  */
    /*-----*/

    int j;
    long k;
    static long idum2=123456789;
    static long iy=0;
    static long iv[NTAB];
    double temp;

    if (*idum <=0)
    {
        if (-(*idum) < 1)
        {
            *idum = 1;
        }else
        {
            *idum = -(*idum);
        }
        idum2=(*idum);
        for (j=NTAB+7; j>=0; j--)
        {
            k = (*idum)/IQ1;
            *idum = IA1 *(*idum-k*IQ1) - IR1*k;
            if (*idum < 0)
            {
                *idum += IM1;
            }
            if (j < NTAB)
            {
                iv[j] = *idum;
            }
        }
        iy = iv[0];
    }
    k = (*idum)/IQ1;
    *idum = IA1*(*idum-k*IQ1) - IR1*k;
    if (*idum < 0)
    {

```

```
    *idum += IM1;
}
k = (idum2)/IQ2;
idum2 = IA2*(idum2-k*IQ2) - IR2*k;
if (idum2 < 0)
{
    idum2 += IM2;
}
j = iy/NDIV;
iy=iv[j] - idum2;
iv[j] = *idum;
if (iy < 1)
{
    iy += IMM1;
}
if ((temp=AM*iy) > RNMx)
{
    return RNMx;
}else
{
    return temp;
}
}
```