Whitepaper - Project Zulu

Jacob Eva, Connor Bryan

April 1, 2023

Contents

1 Introduction

1.1 Explanation of project

CTF challenges, or capture the flag challenges, are challenges which users download and complete in order to submit the discovered ‘flag‘ to the CTF platform they got the challenge from. The topics of these challenges can vary massively. Here are some examples:

- Web exploitation
- Cryptography
- Reverse engineering
- Steganography
- OSINT (open source intelligence)

Usually, when users submit the flags in these challenges to a CTF platform, they are given the amount of points assigned to the challenge, as a reward. On many CTF platforms, there are site-wide leaderboards which allow you to view the ranking of each user according to the number of points they have.

Project Zulu is free and open source CTF platform for use within societies and organisations which wish to publish challenges and track users’ scores in a leaderboard. In order to complete challenges, users submit ‘flags‘ which they find within the challenges, and are rewarded with points accordingly, much like other CTF platforms.

Project Zulu is produced by:

- Jacob Owen Eva - st20202526
- Connor Bryan - st20201795

1.2 Overview of project plan

The planned features of this project are clearly laid out within the README.md file on our git repository here. They can also be seen plotted against their delivery dates within a Gantt chart in Figure 1.

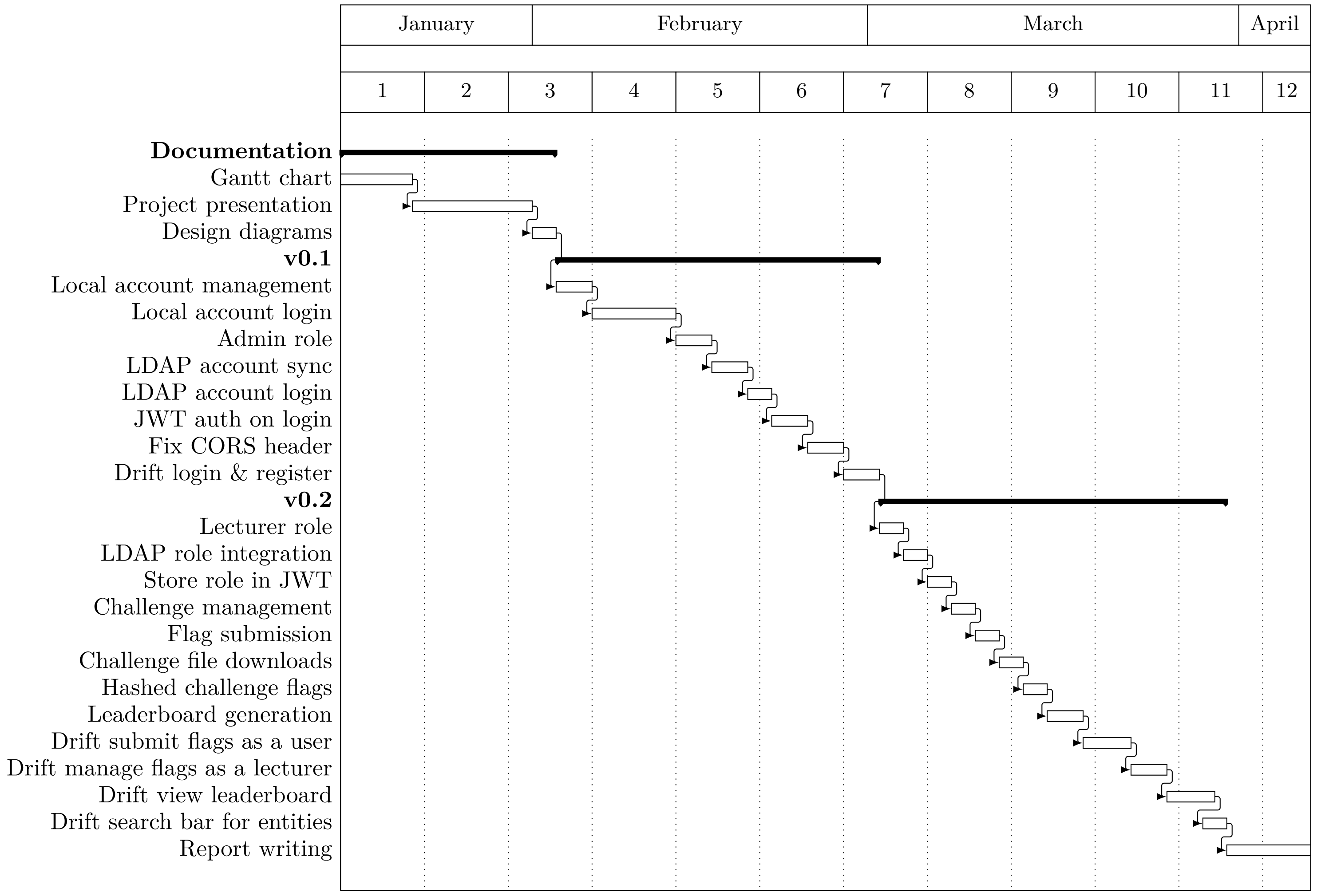

Figure 1: The Gantt chart for planning the development of Project Zulu

Much of the conceptual planning and design was performed prior to this module, hence the low amount of time allocated to such tasks in the Gantt chart in Figure 1.

2 Background

2.1 Selection of project

Jacob originally had the idea for Project Zulu over the summer whilst on an internship placement, where he observed the fragmentation of the various CTF platforms in action, with private companies each developing their own CTF platforms, rather than using a common self-hosted platform, which can promote interoperability between different instances, along with vastly reducing the cost of development. Wordpress is an excellent community-maintained content management system, which is used by 42% of the web (Wordpress 2023). It allows people to create websites very easily, but still allows for deep customisation and modification thanks to a rich feature set and open source nature. Therefore, open source community projects are excellent for benefiting organisations self-hosting their own infrastructure, so he asked himself why a feature-rich and interoperable one didn't exist for CTF platforms.

2.2 Why is this project needed?

The current issue with CTF platforms is that many of them are proprietary and built for private companies, which means users are locked into platforms which they cannot control. A lot of these CTF platforms also have premium features which users can pay to access, increasing the bar for users who want to be able to use all the features of the platform freely. Unfortunately, this also includes a free and open source CTF platform, CTFd. CTFd charges a monthly subscription for features such as web shells, certain game types and integration with external identity provider services, such as LDAP (LLC, Kevin Chung // CTFd 2023). Additionally, CTFd also scales poorly, as it is written in Python, a programming language which is unsuitable for web development due to its incredibly slow speed and high memory usage (Shanika, Wickramasinghe 2022).

2.3 How is Zulu different?

We have chosen to write Zulu using the Rust programming language. The Rust programming language is designed with memory safety in mind, along with other safety features, in order to ensure programs written in Rust are immune to common memory vulnerabilities such as buffer overflows (Sasidharan, Deepu 2022). This is especially important for a CTF server, as these are at much higher risk of being targeted for exploitation, because if they are compromised, competitors could cheat and modify their scores.

Additionally, Zulu is designed to be free software, in both the senses that it is designed to respect users' freedom, but also be completely free for users to access, and organisations to deploy on their own infrastructure.

Zulu will be designed to encourage decentralisation within the CTF server community, allowing, and in fact encouraging, organisations and university societies to host their own dedicated instance of Zulu on their infrastructure. This allows them to have total control over their CTF server, rather than being accountable to an external entity, who may attempt to exert control over the users of their platform. For example, on a proprietary CTF platform known as ”HackTheBox”, any challenges that are retired after being released for a period of time are locked behind a membership paywall. This is unfair to users, as not everybody may be able to afford such a membership, especially university students (*Hacking Training For The Best* 2023).

Therefore, we believe it is important that organisations are able to deploy their own CTF server, in their sole control, without the potential for the financial interests of external bodies to interfere with such learning.

3 Systems Analysis and Design

3.1 Project scoping

CTF platforms can have a very wide range of features, such as spawnable victim virtual machine / container instances, formation of teams, team-based challenges, time-based events, etc.

However, we were careful to take the liberty to perform proper scoping for the expected functionality and aims of Project Zulu, to create an initial release. This can be seen within the planned tasks within Section 1.2, as the planned featureset for this release of the project is rather small, in comparison to the features many mature CTF platforms contain.

We have chosen to limit the functionality of the initial release of the project to the following:

- Creation, modification and deletion of accounts
- Auto-import of accounts from LDAP
- Creation, modification and deletion of challenges
- Ability to upload and download challenge files
- Ability to view leaderboard
- Ability to search for entities

It should be noted of course, that this is a high level overview and lacks detail. We have defined this small and manageable feature set to prevent feature creep, and ensure the project can be completed in a timely manner with minimal delay.

3.2 Analysis of requirements

Our user stories for the project are as follows. It should go without saying, that as we list different groups, the user stories from the previous groups also apply. Meaning that, lecturers will also want to register accounts as much as users, etc.

3.2.1 Users

A user is any authorised person accessing the system who does not have special privileges.

1. As a user, I want to register an account, so that I can make an account on the platform
2. As a user, I want import my LDAP account, so that I can make an account on the platform seamlessly

3. As a user, I want to login, so I can access the platform
4. As a user, I want to modify my account details, so that I can keep my information up to date
5. As a user, I want to delete my account, so that I can revoke my consent for my data to be stored
6. As a user, I want to search for challenges, so that I can browse them
7. As a user, I want to download files for challenges, so that I can try them
8. As a user, I want to submit flags, so that I can complete challenges
9. As a user, I want to view the leaderboard, so that I can see my ranking

3.2.2 Lecturers

A lecturer is an authorised person accessing the system who has privileges to release challenges.

1. As a lecturer, I want to create challenges, so that users can try them
2. As a lecturer, I want to modify my challenges, so that I can keep their information up to date
3. As a lecturer, I want to delete my challenges, to remove unsuitable challenges
4. As a lecturer, I want to upload a file for my challenges, so that users can download them

3.2.3 Admins

An admin is an authorised person accessing the system who has the privileges to manage the Zulu instance.

1. As an admin, I want to modify challenges, so that I can keep their information up to date
2. As an admin, I want to delete challenges, to remove unsuitable challenges
3. As an admin, I want to modify accounts, so that I can keep their information up to date
4. As an admin, I want to delete accounts, to remove unsuitable accounts

The hardware and software requirements for Zulu will be extremely minimal, with any modern portable or desktop computer and operating system (Linux, BSD, Windows, etc.) being easily capable of running the software and serving a small number of clients (1-20). However, Zulu is primarily intended for use on servers, due to its intended nature to serve as a persistent CTF platform, which will be able to serve much larger numbers of clients (100+).

3.3 Prioritisation of requirements

Throughout the development of this project, we have had to consider which requirements to keep in this release, and which to delegate to a future release instead. We have many requirements which we plan to satiate in the future, including:

- Federation between different instances with ActivityPub (*ActivityPub Rocks!* 2023) (**Mastodon**)

- Community-made challenges, shared over the federated network
- 4 letter identifiers for instances on the federated network
- Challenge tags
- Challenge descriptions

And many more features. These would be excellent to include into Project Zulu in the future, however for the sake of getting an initial version of the project completed, we chose to prioritise the functionality documented in Section 3.2, lest we forever remain in development hell due to scope creep.

3.4 System diagrams

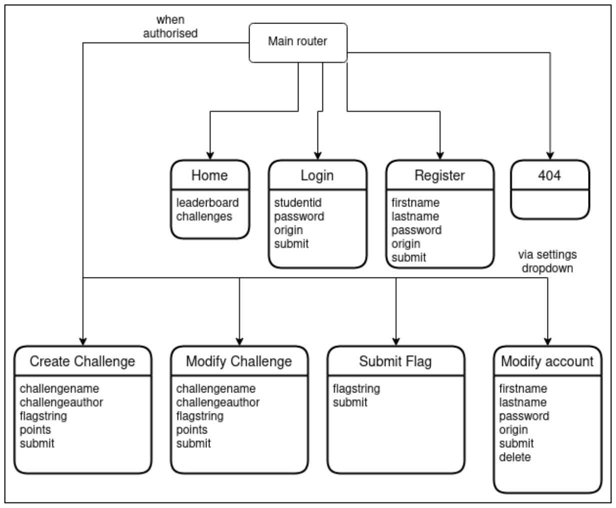

Figure 2: A diagram of all the routes provided by Drift

Within Figure 2, the various routes provided by Drift can be clearly seen. Drift provides these various different routes in order to enable users a user-friendly way to interact with the backend, Zulu, without having to send unintuitive API requests, as these are generated by the page when the user fills out and submits the provided fields.

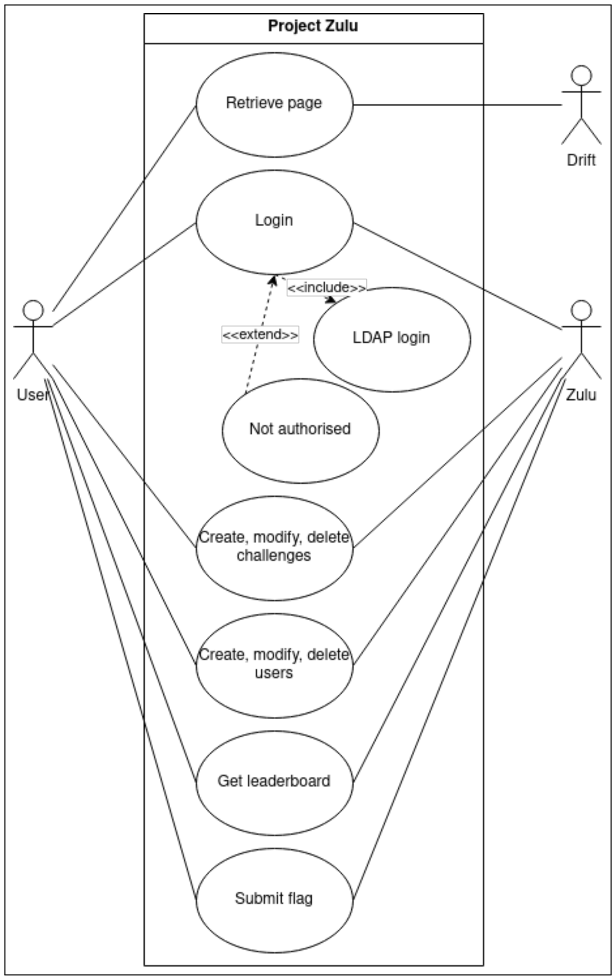

Figure 3: A use case diagram of the features provided by Zulu

Within Figure 3, an example use case diagram is provided for all the functionality provided by Zulu. This use case diagram very much helps illustrate the place of Drift and Zulu in providing a fully developed user experience. Drift, the frontend, will serve a webpage to the user, which displays information the user is instructed to retrieve from the backend, Zulu. When the user attempts to login, the LDAP login functionality will be invoked if Zulu deems it necessary, and will return a JWT or an unauthorised message, if the user provided incorrect credentials.

Zulu is also designed to interact with LDAP (lightweight directory access protocol), which can allow for much more seamless integration with organisational or university society networks. LDAP is an application protocol for accessing directory information over a network, such as user accounts. Within data objects, a variety of data can be stored, in the case of user accounts, this includes account emails, passwords, first and last names, user groups, etc. (Ernesto, Luiz 2007)

One popular implementation of LDAP in the modern world is Microsoft Active Directory, which is used by a large amount of educational institutions in the United Kingdom, as well as worldwide (alvinashcraft 2020). Therefore, designing Zulu with LDAP integration baked-in is core to ensuring adoption and usability of the CTF server.

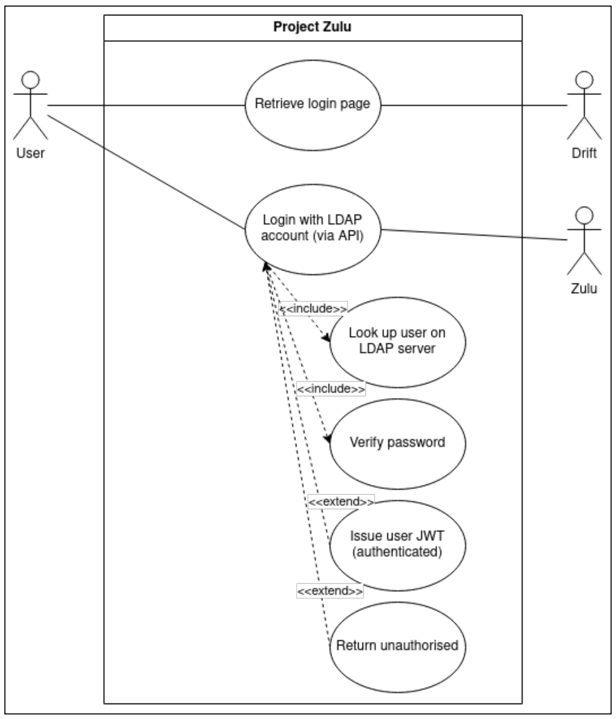

Figure 4: A use case diagram of a user signing in using an existing LDAP account

Within Figure 4, a more detailed use case diagram of a user logging in with an LDAP account can be seen. When the user attempts to log in using a user account which is not found in the database, an attempt is made by Zulu to look up the user on LDAP. If their user account object can be retrieved, then Zulu also attempts to log in as them on the LDAP server using the user's provided credentials. If this is successful, then finally a record is created in Zulu's database with the details it is able to retrieve about the user from the LDAP server. The user is then logged in. Again, if the username or password provided was incorrect, then Zulu responds that the user is unauthorised.

4 Application Development

4.1 Tools

4.1.1 SQL database

Zulu is designed to use MariaDB, an SQL database, to store data regarding users and challenges on the instance. We have chosen to use an SQL database for storage, as support for it is easy to implement within the context of a Rust program, since there are excellent Rust libraries for interacting with SQL databases, such as sqlx. In the case of Zulu, we utilised sqlx for the database storage functionality, as it's written in pure Rust, and therefore should not introduce many security vulnerabilities. sqlx also has excellent support for different query types, along with parsing records retrieved from the database in question (LaunchBadge 2023).

The following tables are present in the database in this version:

- accounts - contains user accounts
- challenges - contains information about challenges
- solves - contains what challenges users have solved

Many of these tables have cross referencing foreign keys, which allow for records to reference other records in seperate tables, but in such a fashion as to be linked. This allows, within the challenges table for example, to store the account ID of the author of the challenge.

4.1.2 Yew

Yew is a modern Rust framework for the creation of client-side web applications utilising web assembly (yewstack 2023). It operates based on the MVC (model, view, controller) architecture, which is an architectural pattern used for application development. The model is the component that contains the logic for the application, alongside facilitating accessing, manipulating and storing data for the application. The view is the components of the application which the user can see, namely the UI. Finally, the controller is an intermediary between the controller and the view, which interprets the changes the user makes in the view and perfoms modifications to the model accordingly (**RalFerrerGarca2023iOSArchitecturePatte**).

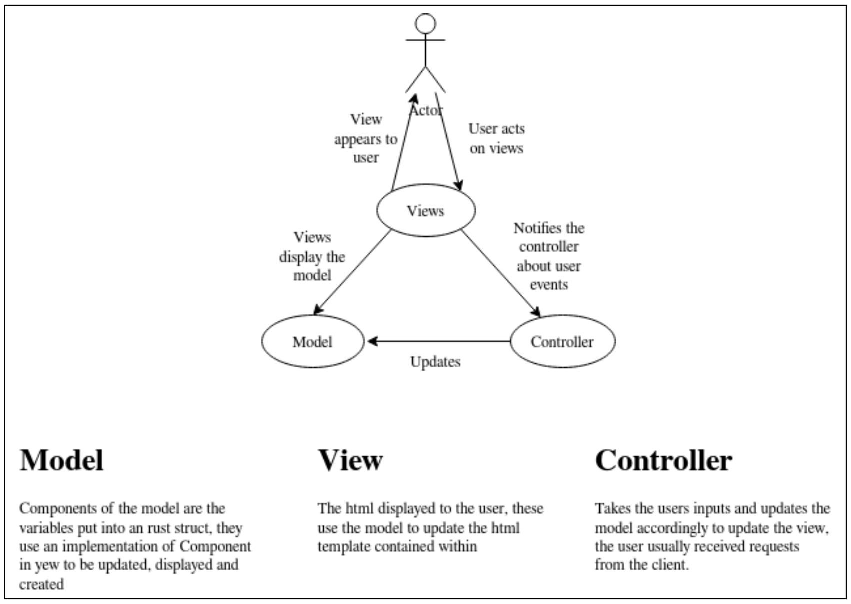
Actor
View appears to user
User acts on views
Views
Views display the model
Notifies the controller about user events
Model
Controller
Updates
Model
Components of the model are the variables put into an rust struct, they use an implementation of Component in yew to be updated, displayed and created
View
The html displayed to the user, these use the model to update the html template contained within
Controller
Takes the users inputs and updates the model accordingly to update the view, the user usually received requests from the client.

Figure 5: An illustration demonstrating the use of the MVC architecture within Drift (ibid.)

Within Figure 5, the relationship between the view, model and controller within the architecture is illustrated, along with explanations of how the MVC architecture relates to how Drift itself was developed.

4.1.3 Rocket

Rocket is a web framework for Rust, which is designed mainly for server side rendering and with safety and security as critical aspects (Rocket 2023). This makes it an excellent fit for use as the framework used for Zulu, as the security of Zulu is critical to the overall project. If Zulu can be controlled by an attacker, the confidentiality, integrity and availability of the data stored therein can be critically affected, which would be a severe failure in the overall project. Additionally, clients accessing Zulu could also be compromised, as it could be used to serve malware to users if compromised itself.

Rocket has excellent features included within the framework, such as request guards. Request guards were incredibly useful in the development of Zulu in order to ensure the security of aspects of the application, such as creating challenges. A request guard was added to ensure that unauthenticated users, or users with privileges lower than lecturer could not make API calls to create challenges, through inspection of the user's JWT, which they send with each request as an authorisation header. This JWT is also signed, albeit insecurely for the moment, which will prevent users from forging their own JWTs with incorrect levels of access. These JWT operations were performed with the Rust library 'jwt'.

4.2 Feedback

We approached two members of the Cardiff Met CTF society for their thoughts as users on what features they like, and what features they think we may have missed in Project Zulu. Additionally, we also asked them for feedback on the UI design of Drift.

4.2.1 Subject 1

Subject 1 commented that they were impressed overall by the UI design of Drift, however they did note it is slightly rough in places. For example, they suggested that the bottom bar be locked at the bottom of the page, instead of floating, as it overlapped in places. They also suggested including a description for challenges would be helpful, to which we responded we plan to do this in a later version.

4.2.2 Subject 2

Subject 2 said they enjoyed their experience with the UI of Drift. They mentioned that the ability to add images to challenges would be a good addition, to which we replied that we plan to do this in the future. They also expressed great enthusiasm for integrating different instances of Zulu with one another, by for example allowing the sharing of challenges. We also stated that too is a planned feature for a future version.

4.2.3 Analysis

Generally, features both the subjects suggested for Project Zulu were already planned to be added in the future, or have been considered in some capacity at least, which therefore leads us to believe we have done a good job of understanding user needs, especially considering their experience was very positive overall. We plan to make adjustments to the UI of Drift in the future due to the concerns raised in Section 4.2.1.

4.3 User Guide

The user guide can be viewed within Appendix A.

5 Testing

Extensive testing was performed on the program in order to ensure the functions performed their tasks properly and without unintended functionality. To this end, unit tests were integrated with the program, which could be run with ‘cargo test‘(*Unit testing - Rust By Example* 2023), as Rust has integrated testing support. Within Figure 6, the a table of the unit tests we created are clearly laid out.

Test No.	Type of test	Purpose of test	Expected outcome	Test data	Outcome
1	Unit	Check if the test account is in the database	The record is returned	Select account id from database	Success
2	Unit	Register test account to database	The user account is added	Insert account information into database	Success
3	Unit	Select user from database using username	The record is returned	Select username from database	Success
4	Unit	Modify existing account	The record is modified	Create user and modify password after creation	Success
5	Unit	Delete a user account	The record is deleted	Delete a record by referencing the username	Success
6	Unit	Check database connection	The connection is successful	Connect with the username zulu and password zulu	Success
7	Unit	Check if a valid token can be decoded	The token can be decoded and matches	A token with claims ”sub” and ”test”	Success
8	Unit	Check if an invalid token is rejected when decoded	The token is rejected	A token with invalid claims ”user” and ”test”	Success
9	Unit	Check if a hashed password can be verified	The password is verified	A password ”test”, verified with the same string	Success
10	Unit	Check if a hashed password can be verified	The password is verified	A password ”test”, verified with the same string	Success
11	Unit	Check if the LDAP config file can be parsed	The file can be parsed	The stock default.toml config file	Success
12	Unit	Check if an LDAP user account can be logged into and imported	The account can be logged into successfully	An example test account on LDAP	Success

Figure 6: An illustration demonstrating the use of the MVC architecture within Drift

Considering all of the unit tests we wrote were successful when run, we believe that the software we have produced has very good code quality, with minimal unintended functionality within the functions we have written. Overall the user experience for the software is excellent, with minimal changes needing to be implemented in order to polish the existing functionality for the project.

6 Evaluation

We've developed a working version of Zulu that includes all the features mentioned above, as well as incorporating the suggestions from the feedback we gained, such as fixing the footer at the bottom of the page and cleaning up the UI. We faced a plethora of challenges during development such as:

- WASM - Since WASM is still new it lacks some functionality such as accessing cookies straight from the browser (*Introduction - Rust and WebAssembly* 2023)
- LDAP - Understanding how to pull the information about the user and verifying their authenticity from the server
- Rocket - A lot of the functionality was recycled from an older project, but was modified to work with the new requirements

However, eventually many of these challenges were overcome, thanks to adequate time allocated within the project plan.

7 Conclusion

Zulu was developed in response to the need for a free and open source CTF server for the educational sector, not only for university CTF societies but also for organisations. Overall, this project has been very successful in producing a functioning CTF server that will serve many users and CTF societies across the country, allowing students and professionals to hone their skills using our free software. Zulu will continue to be maintained in the future, with more features planned to be implemented in the future, such as federation.

A User guide

A.1 Prerequisites

These instructions are designed for a Linux-based OS. First, ensure you have the following installed:

- Rust (rustc, cargo, rustup)
- MariaDB
- Git
- SSL development libraries

On debian these and their dependencies can be installed with the command:

```
$ sudo snap install rustup --classic
$ sudo apt install mariadb-server git libssl-dev pkg-config
$ rustup default stable
```

A.2 Compiling and running Zulu

In order to install Zulu, you must first download the code for the project in order to be able to compile it, as we do not provide compiled binaries at this time. In order to download the source code, run the command below.

```
$ git clone https://git.metctf.org.uk/jacob.eva/zulu
```

Once you have downloaded the code for the project, you must then compile both Zulu and Drift, so that they can both be run.

However, there are some prerequisites to this. First, you must create an account for zulu to access the database with. Connect to mysql, create the new account, and set the root password with the following commands:

```
$ sudo mysql
MariaDB [(none)]> create user zulu@localhost identified by 'zulu';
MariaDB [(none)]> alter user root@localhost identified by 'password';
MariaDB [(none)]> \q
```

Next, the database for Zulu must be imported in order to allow for it to run. The database can be imported by running:

```
$ cd zulu
$ sudo mysql -p < zulu.sql
```

Then, you need to declare to cargo the root credentials to connect with the database:

```
$ mkdir .cargo
$ echo -e "[env]\nDATABASE_URL='mysql://root:password@127.0.0.1/zulu'"
  > .cargo/config.toml
```

You also need to create the directory challenges will be stored in:

```
$ mkdir -p static/challenge
```

After importing the database, you are ready to compile Zulu. To do so, run the following command:

```
$ cargo run --release
```

After waiting several minutes for cargo to download the packages, Zulu should be compiled and will launch on your machine.

Next, you need to run Drift, to provide a UI for Zulu.

A.3 Compiling and running Drift

Open a new terminal. Then run:

```
$ git clone https://git.metctf.org.uk/connor.bryan/drift
$ cd drift
$ cargo install --locked trunk
$ rustup target add wasm32-unknown-unknown
$ /home/$USER/.cargo/bin/trunk serve --release
```

Then, access Drift in the web browser by connecting to ‘http://localhost:8080‘. Congratulations! You have successfully deployed a complete Zulu instance.

References

ActivityPub Rocks! (2023). URL: https://activitypub.rocks/ (visited on 03/28/2023).

alvinashcraft (2020). *Directory System Agent - Win32 apps.* URL: https://learn.microsoft.com/en-us/windows/win32/ad/directory-system-agent (visited on 03/24/2023).

Chapter 1. Introduction (2023). en. ISBN: 978-0-201-63361-0. URL: https://learning.oreilly.com/library/view/design-patterns-elements/0201633612/ch01.html (visited on 03/28/2023).

Ernesto, Luiz (2007). *LDAP backends, objects and attributes.* URL: https://tldp.org/HOWTO/LDAP-HOWTO/ldapbackends.html (visited on 03/24/2023).

Hacking Training For The Best (2023). en. URL: https://www.hackthebox.euhttps://www.hackthebox.com (visited on 03/28/2023).

Introduction - Rust and WebAssembly (2023). URL: https://rustwasm.github.io/book/ (visited on 03/28/2023).

LaunchBadge (2023). *The Rust SQL Toolkit.* URL: https://github.com/launchbadge/sqlx (visited on 03/26/2023).

LLC, Kevin Chung // CTFd (2023). *CTFd : The Easiest Capture The Flag Framework.* URL: https://ctfd.io/ (visited on 03/23/2023).

Rocket (2023). *Introduction - Rocket Programming Guide.* URL: https://rocket.rs/v0.5-rc/guide/introduction/ (visited on 03/26/2023).

Sasidharan, Deepu (2022). *Why Safe Programming Matters and Why a Language Like Rust Matters.* URL: https://developer.okta.com/blog/2022/03/18/programming-security-and-why-rust (visited on 03/23/2023).

Shanika, Wickramasinghe (2022). *Rust vs Python: Which One Is Best for Your Project?* URL: https://kinsta.com/blog/rust-vs-python/ (visited on 03/23/2023).

Unit testing - Rust By Example (2023). URL: https://doc.rust-lang.org/rust-by-example/testing/unit_testing.html (visited on 03/28/2023).

Wordpress (2023). *WordPress.com: Build a Site, Sell Your Stuff, Start a Blog & More.* URL: https://wordpress.com/ (visited on 03/26/2023).

yewstack (2023). *Rust / Wasm client web app framework.* URL: https://github.com/yewstack/yew (visited on 03/26/2023).