Project Bermuda
Emily, Stephen, Alex, Jake
Software Design Specification

1. System Overview

This software is intended to be an access portal to the cybersecurity club's laboratory environment. It will allow users to access the lab via a web terminal, via ssh, as well as present the user with information about the available challenges in the environment.

The system is designed to operate on a single server to keep club costs down, and it uses Microsoft authentication to ensure that only users from the University of Oregon can access the system. It's composed of a Flask backend, with a raw web frontend. The Flask system connects to a MariaDB database for state, and manages a set of Podman containers using the Podman API that the users use to access the lab. The SSH server also runs alongside the Flask backend - it's written in Golang and also connects to the database to authenticate users.

2. Software Architecture

2.1. Components and their functionality

2.1.1. Web Interface

2.1.1.1. Landing page

2.1.1.1.1. Will show welcome information as well as a link that directs users to the authentication page

2.1.1.2. Authentication page

2.1.1.2.1. Will start the Microsoft OAuth flow

2.1.1.2.2. After successful Microsoft login, it will ensure that the user has an allowed email, such as an @uoregon email. This will be customizable to allow other schools access if necessary

2.1.1.2.3. It will check whether the user is already in the database, if not, it will add an entry in the database

2.1.1.2.4. After a successful authentication/signup flow, it will redirect the user to the configuration page.

2.1.1.3. Configuration page

2.1.1.3.1. Will show information about using the site and basic rules

2.1.1.3.2. Will link to the terminal page

2.1.1.3.3. Will link to the challenges page

2.1.1.3.4. Will have a series of textboxes, beginning with a box for an ssh key, that the user can submit and save the content to the database

2.1.1.4. Challenges page

2.1.1.4.1. Will pull a list of all active challenges from the database

2.1.1.4.2. Will display each challenge to the user, along with information about it, whether it's been solved, and if it hasn't been solved, a text box allowing the user to submit a flag.

2.1.1.4.3. If a user submits a flag, it'll check whether it's correct. If it is, it'll display a message to the user and update the database

2.1.1.5. Terminal page

2.1.1.5.1. On load, will display a terminal

2.1.1.5.2. Will pull information about any active containers from the database, if so, connect to the active container from the user
2.1.1.5.3. Else, will create a new container and connect

2.1.2. SSH Server
2.1.2.1. Authentication module
2.1.2.1.1. Will connect to the database and find the ssh key associated with the user
2.1.2.1.2. Will check that the user's key is associated with their account
2.1.2.2. Terminal module
2.1.2.2.1. Will pull information about any active containers from the database, if so, will connect, else, create a new container for the user

3. Software Modules

○ 3.1. Web Interface

3.1.1. Primary Role and Function

The web interface provides users with a universal way to access the lab environment, authenticate using their UOregon accounts, and view available cybersecurity challenges. The key pages ensure that anyone, regardless of technical ability can complete challenges with the club. It's written using flask templates and raw web technologies, aside from the terminal page, which is written with xterm.js, since it's the industry standard for web-based terminals.

3.1.2. Interface

- **Landing Page:** Displays welcome information with a link to the Authentication page.
- **Authentication Page:** Initiates Microsoft authentication for users with authorized emails (e.g., @uoregon.edu). If the user is new, an entry is added to the database.
- **Configuration Page:** Shows usage guidelines, links to the terminal and challenges pages, and allows users to submit their SSH keys.
- **Challenges Page:** Displays a list of active challenges, indicates solved challenges, and allows users to submit flags.
- **Terminal Page:** Creates or connects to an active container based on the user's session.

3.1.3. Static Model

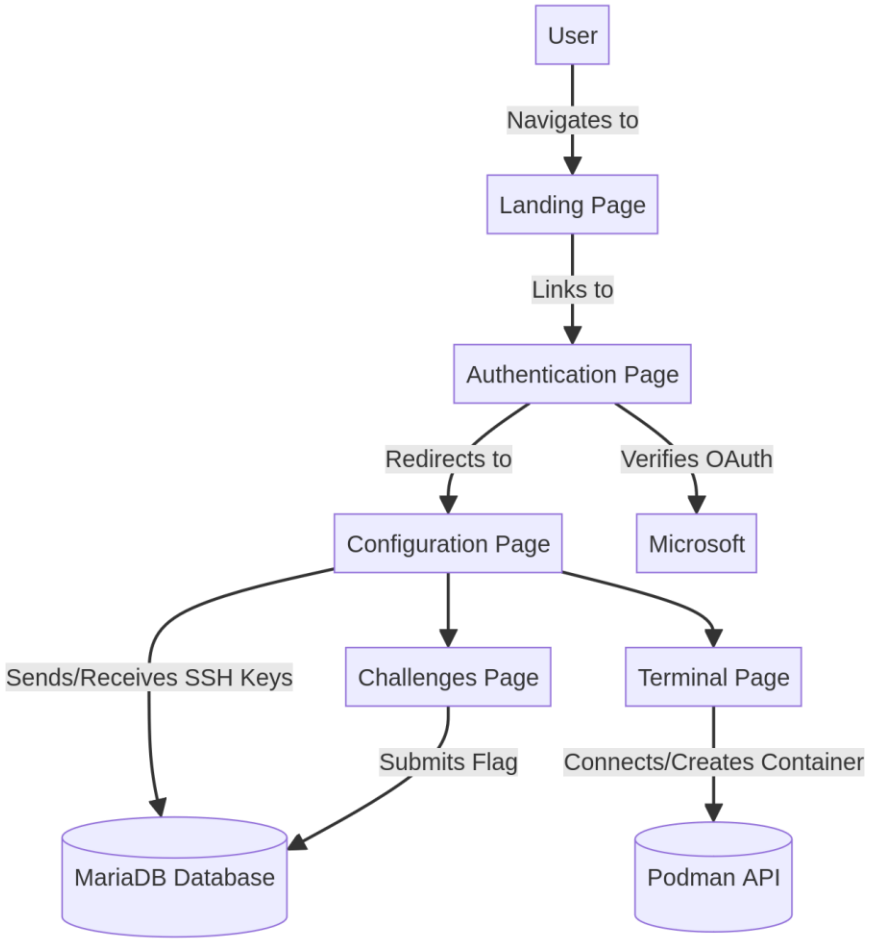


Figure 3.1. shows a static model of the web interface, detailing interactions between pages and the MariaDB database for user and challenge data.

3.1.4. Dynamic Model

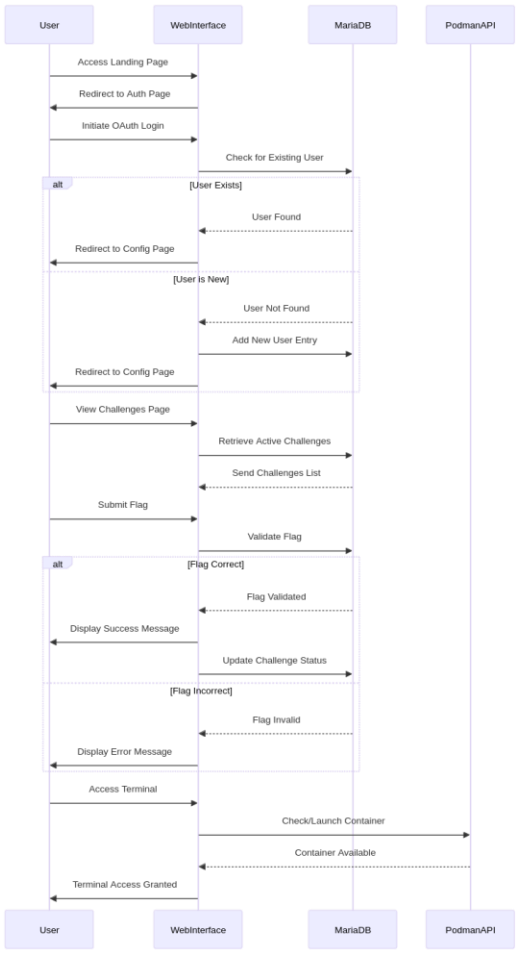


Figure 3.2. provides a sequence diagram of the authentication flow, including user redirection and database updates upon successful login and flag submission.

3.1.5. Design Rationale

The design of the web interface prioritizes maintainability, picking technologies and flows that will allow future developers to modify it as necessary, adding additional challenges, pages or config options as the club's needs change over time. Hence the reasoning for picking a flask and raw html based app.

3.1.6. Alternative Designs

Writing the web interface in a more advanced web technology, such as React, Next.js, or Angular, but it was decided against because we want the site to be maintainable by future club members who may not be versed in the latest technologies, and may not have learned those frameworks.

○ 3.2. Flask Backend

3.2.1 Primary Role and Function

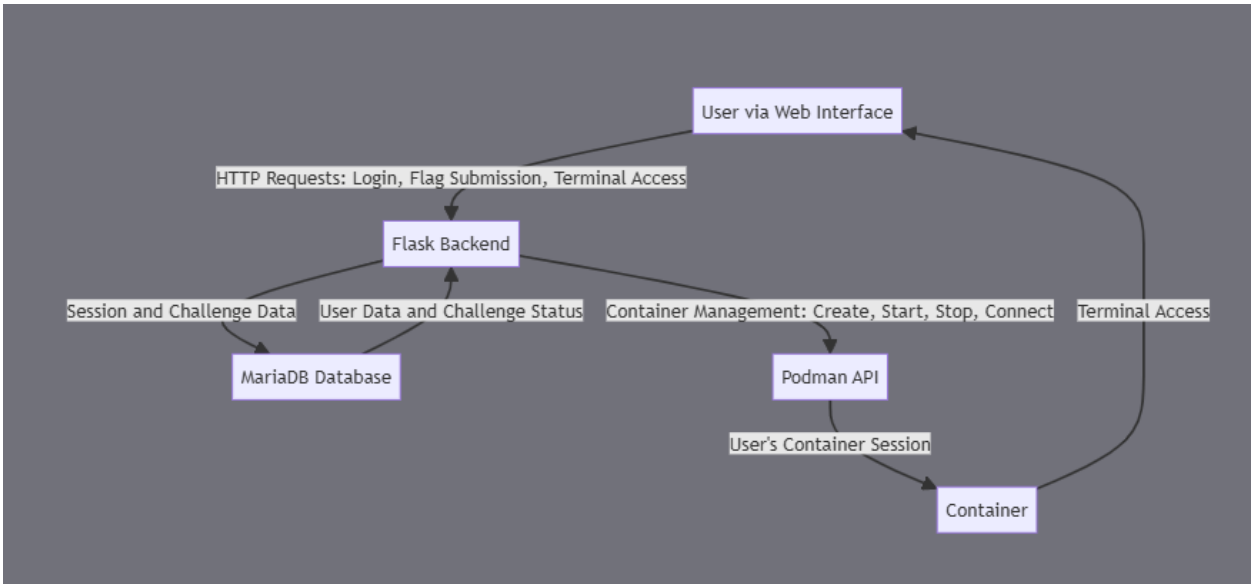
The Flask backend serves as the core intermediary between the web interface, database, and container management. It processes and routes requests from users, such as accessing the terminal, submitting flags, or retrieving challenge data. When a user accesses the terminal webpage, the Flask backend communicates with the Podman API to create or manage containers, ensuring each user connects to their session. It also interfaces with MariaDB to manage user sessions,, track challenges, and update user information.

3.2.2 Interfaces

- **Web Interface:** Receives HTTP requests from the web interface and handles user interactions such as login, flag submission, and terminal access.
- **Podman API:** Manages container lifecycle operations (create, start, stop, and connect) for individual user sessions.
- **MariaDB:** Stores user data, challenge statuses, and container session information.

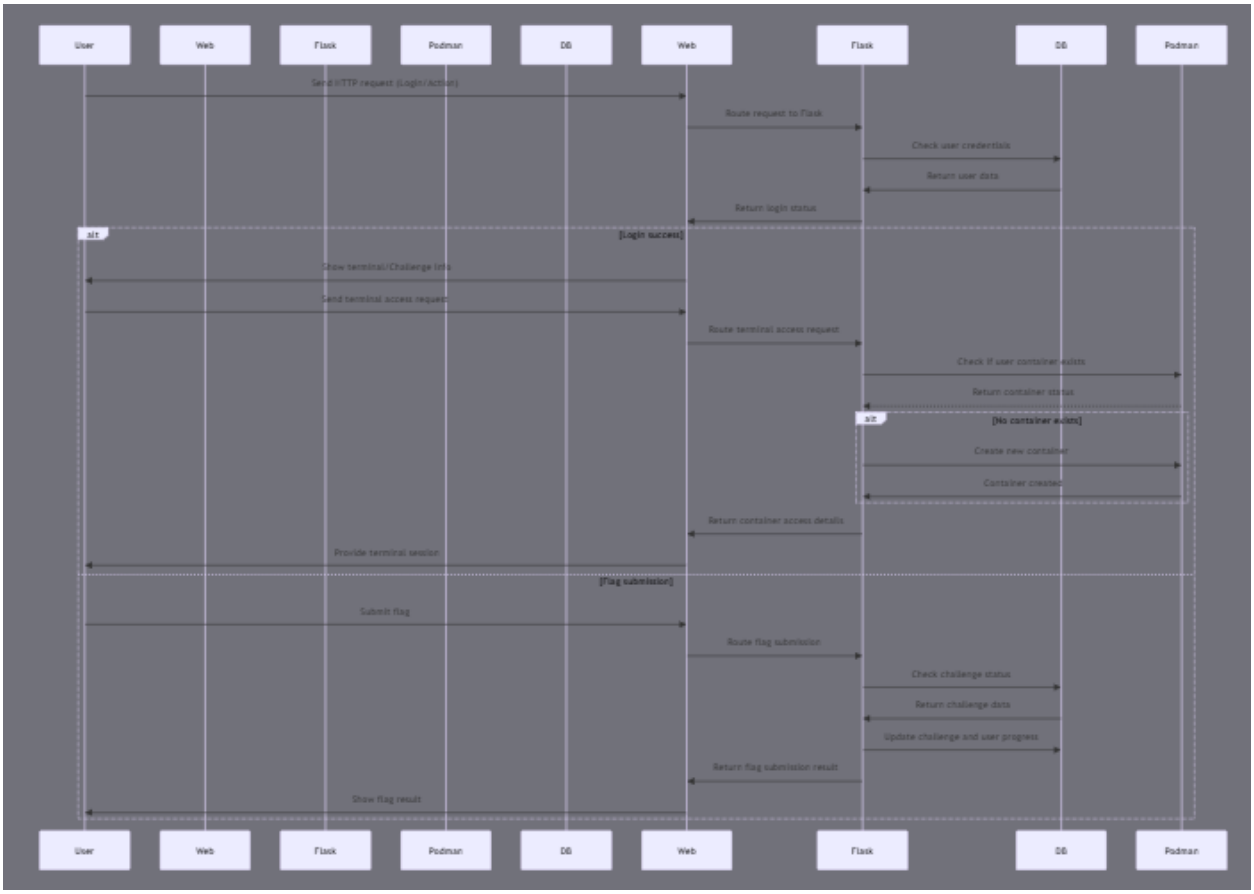
3.2.3. Static Model

Figure 3.3. illustrates the static relationships among the Flask backend, MariaDB, and the Podman API, highlighting data flows for user authentication and container management.



3.2.4. Dynamic Model

Figure 3.4. shows a sequence diagram detailing the container lifecycle from user login to container connection, including checks for existing containers and creation of new ones if necessary.



3.2.5. Design Rationale

Flask was chosen for its lightweight and modular design, in addition to it being taught at the University and has an easy to pick up architecture. allowing rapid development and integration with Microsoft and the Podman APIs. Flask's simplicity enables the backend to remain easily maintainable and cost-effective, crucial for a university-based club setting.

3.2.6. Alternative Designs

We considered using the podman command line tools, but decided that having our flask app parse and run that many arbitrary commands was too complicated and a security risk.

We also considered writing the backend in Golang, using the Gin web framework, but due to the requirements for maintainability, this was rejected, since there isn't enough experience with Golang at this University.

○ 3.3. SSH Server

3.3.1. Primary Role and Function

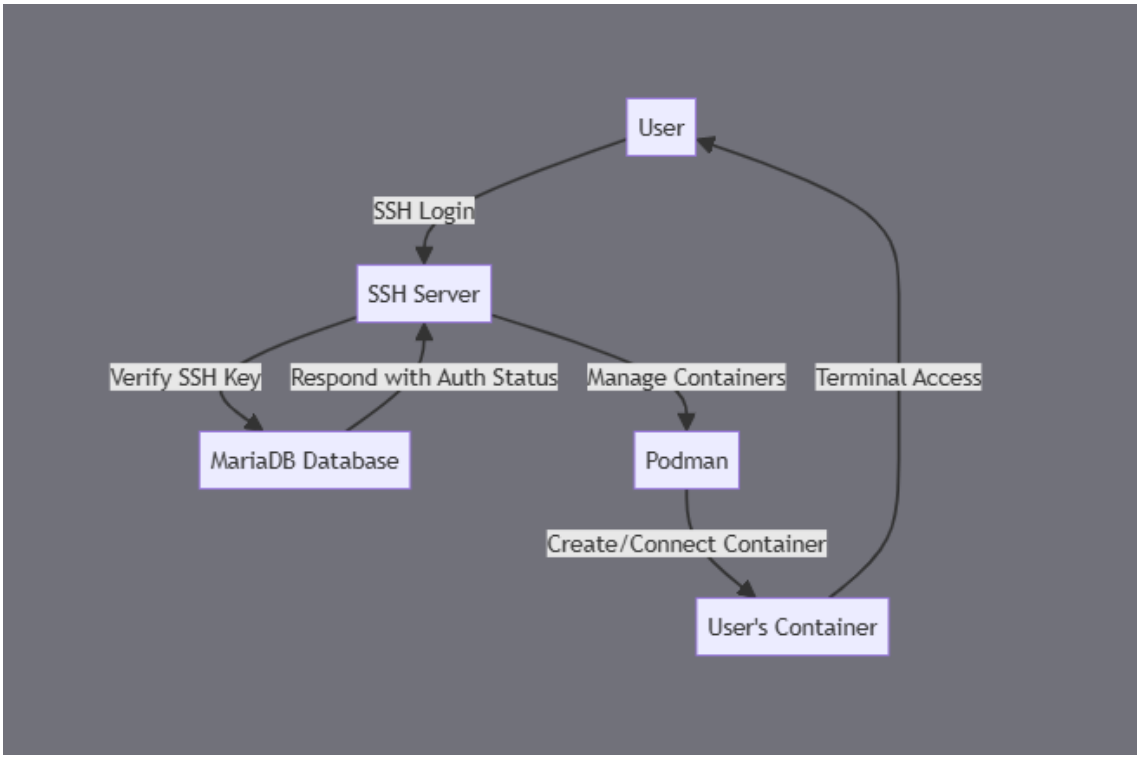
The SSH server provides direct terminal access to users via SSH, linking their credentials to container access and maintaining session security. It's written in Golang to take advantage of the rich SSH libraries, and uses the Golang Podman bindings to manage the containers.

3.3.2. Interface

- **Authentication Interface:** Verifies users' SSH keys against stored credentials in the MariaDB database.
- **Container Interface:** Interacts with Podman to provide access to the appropriate active container, creating one if none exists.

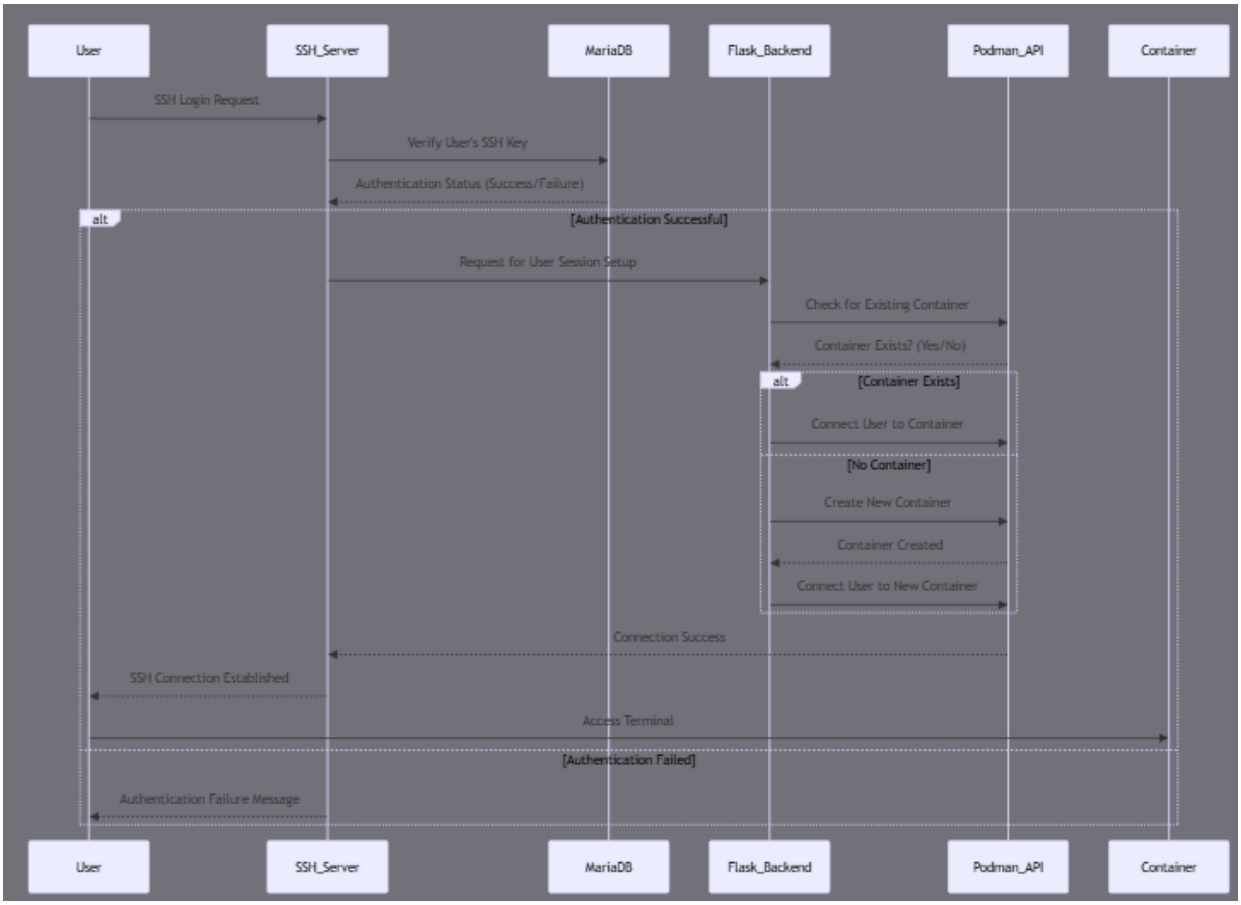
3.3.3. Static Model

Figure 3.5. displays the SSH server's static model, showing data flow between the server, database, and container environment for secure authentication and session management.



3.3.4. Dynamic Model

Figure 3.6. outlines the dynamic authentication and container connection process, illustrating interactions among the SSH server, Flask backend, and Podman API during a user's SSH login.



3.3.5. Design Rationale

Golang was chosen for the SSH server due to its speed and efficiency in handling concurrent connections, essential for supporting multiple users in the lab environment. Integrating the SSH server with the database enables a unified authentication system, enhancing usability and security. The benefits of library and database support, in this case, outweighed the benefits of Python.

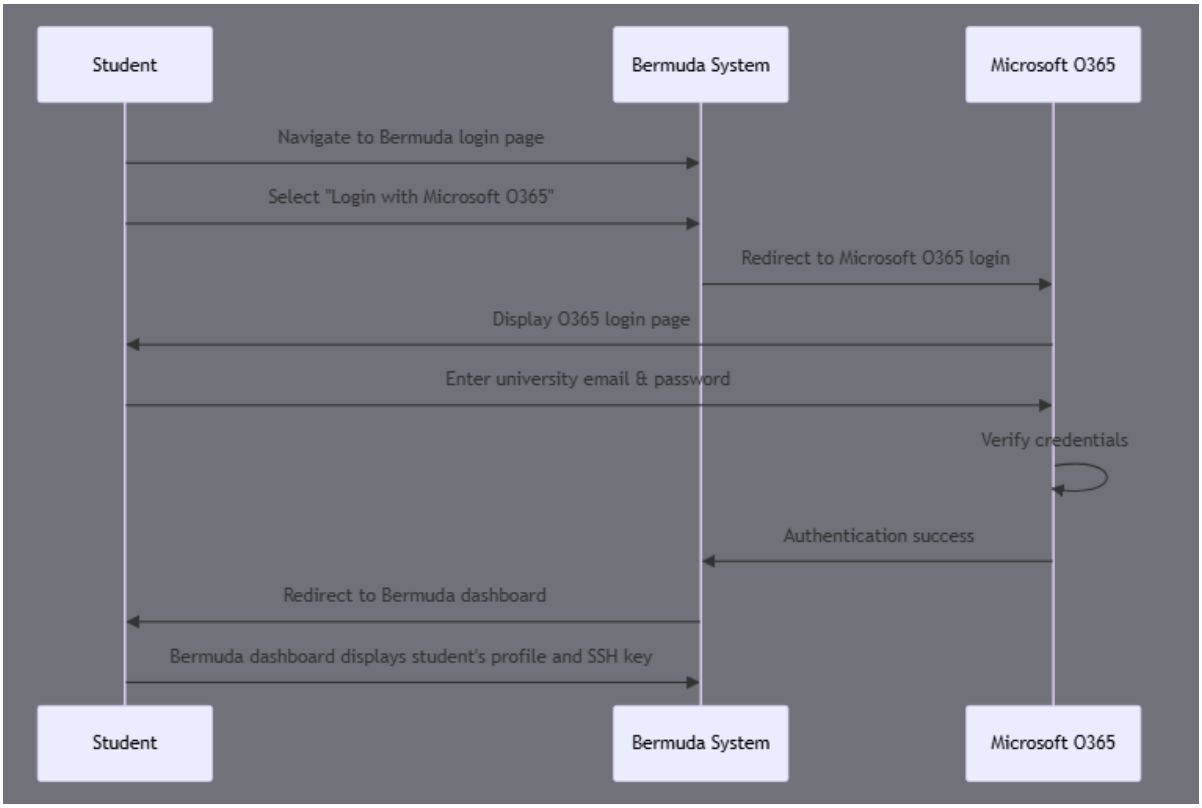
3.2.6. Alternative Designs

Two other options were considered for this ssh server. First, using the prebuilt BSD or Linux servers, but we found that they didn't have appropriate bindings or authentication modules to allow access from the MaraiDB and also limiting access to only specific Podman containers.

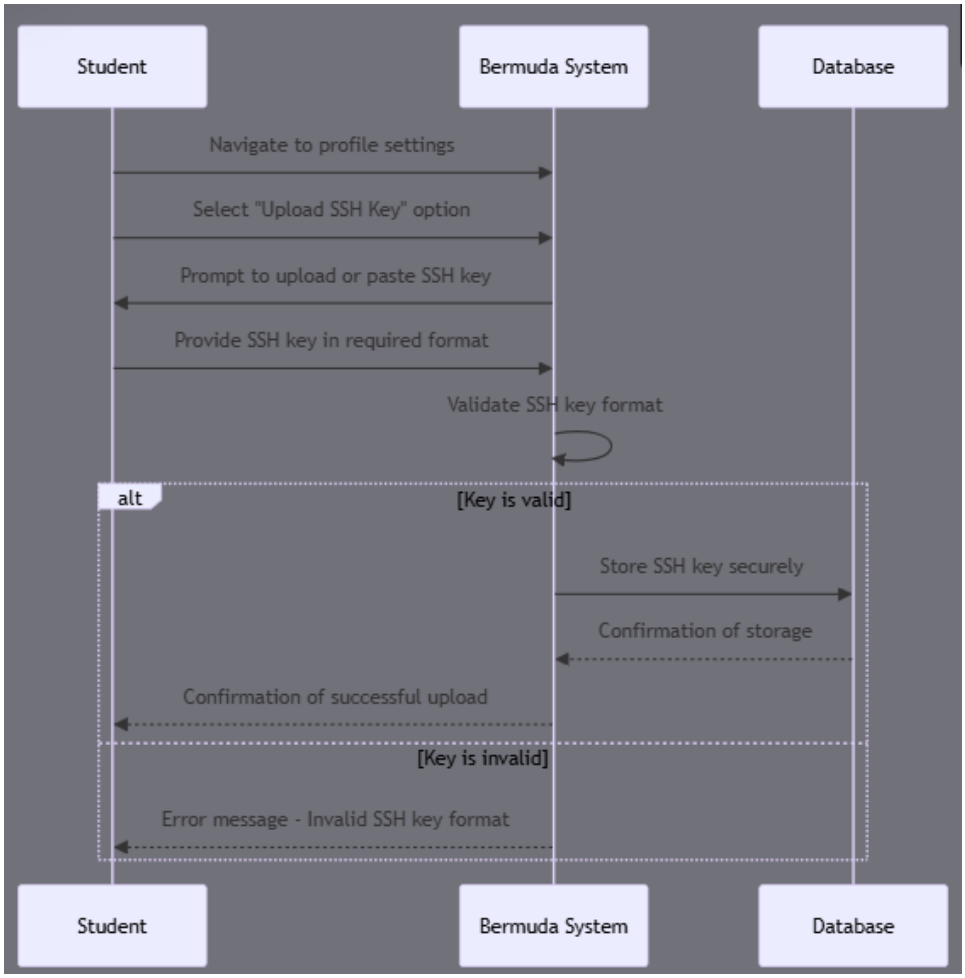
We also considered writing the server in Python, but found that library support for a robust SSH server was lacking, and so if we did it in python, much of the protocol would've had to have be written by hand, which presented significant maintenance and security concerns.

4. Dynamic Models of Operational Scenarios (Use Cases)

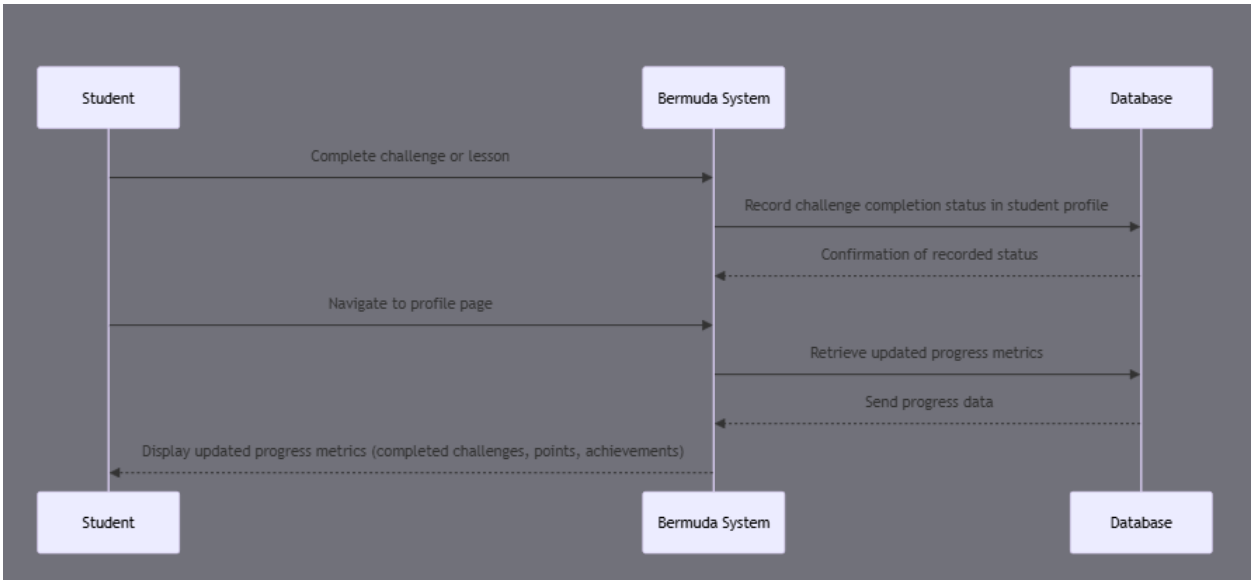
Use Case 1: Microsoft Authentication



Use Case 2: SSH Credentials



Use Case 3: Challenge Progress



5. References

IEEE Std 1016-2009. (2009). IEEE Standard for Information Technology—Systems Design—Software Design Descriptions. https://ieeexplore.ieee.org/document/5167255

6. Acknowledgments

SRS template was provided by Juan Flores.
Diagramming was done in Mermaid using this website https://www.mermaidflow.app/editor