Implementation Discussion

We will be running the backend using Python with the Flask framework alongside JWT and Flask-SQLAlchemy. Flask allows us to quickly and easily set up multiple web endpoints. JWT will provide access control for the Flutter mobile app; specifically, the authorization tokens will be created on the Flutter app and sent in the URL request to Flask.

After the endpoint has verified that the JWT is valid, it can make a request to the SQL database using SQLAlchemy, extract the relevant data, and return it via a REST API. The entire server will be run within a Docker container so it can be easily deployed onto a VM or VPS. The frontend will be developed entirely in Flutter, utilizing persistent storage to keep the user logged in using the shared storage system. When the app opens, a JWT will be created for the session, allowing the app to download the list of users' societies and check for new notifications.

We also explored alternatives for this stack. We considered using Firebase, which would remove the need for a dedicated server and SQL database by utilizing Google's serverless backend and built-in authentication. We also looked at React Native as a frontend alternative due to its ease of use and extensive set of libraries to help optimize development; however, the major advantage of Flutter is that all our members already know Dart.

Testing Discussion

The none-functional requirements will be tests by several methods, including timing, and testing inputs.

Non-functional requirements 1, 2, 3 and 5 will be tested by timing the speed of the system, passing if they do not exceed the time limit

requirement 4 will be tested by simulating logging in and ending the session then begining another, if the user is still logged in the test passes

requirement 6 will be tested by trying to upload an image over the max file size, if it fails the test passes

requirement 7 will be tested by trying to mutate a post while unauthenticated, if the system does not allow the mutation the test passes

requirement 8 will be tested by trying to execute XSS/SQL injection attacks if the attacks fail the test passes

the tests will be automatically run via flutter unit tests; other testing methods would be harder to integrate into the system as we are already using flutter, and as it’s the front end of the system, the other components can be tested through it.

Critical Analysis

This semester we have identified a key gap in the university experience for students - one which we believe our application can fill.

Through defining our methodology for gathering requirements, developing user requirements from these and then translating these into system requirements we were able to acutely identify the problem we aimed to solve.

We then developed an architectural model, to support our eventual implementation.

We accompanied these with 5 key use cases.

To prepare for implementation we then investigated the tools we could make use of, and the ways we would test our design.

Completing this was not a straightforward task as we encountered complications which challenged our teamwork skills - offering us opportunity to learn and achieve.

We initially we began with 7 team members, and using a simple table in a word document to track attendance.

As the first few weeks went by as we brainstormed draft ideas for our app, and explored how we designed our questions, we realised there was an

At the start we began with 7 team members but found that 4 of them wouldn't show up to pre agreed coursework meeting due to them oversleeping alarms and found they missed labs regularly due to sport societies. To deal with this we started agreeing to tasks over messages that they could do to contribute without having to come to group meeting. We ended up deciding the best course of action was splitting into two group. This allows the larger group to work around their sports while out new smaller group can follow a regular set timetable.