Verification and Validation

Verification is checking that the software does what we expect it to do without having any bugs. Validation is checking that the software does what the requirements ask it to do.

For this project, at each stage of the implementation, I will first consider validation as it is pointless to test and debug only to realise afterwards that you built the wrong thing. I will think through the feature that I want to add, both before and after adding it, to ensure that it is what is required. Remember, the key goal for this project is to make it easier for students and staff to meetup on graduation day, so I must be convinced that a feature will help to do this in order to add this feature.

Once a feature has passed initial validation, I will then begin the implementation. In order to verify a feature it will undergo 2 sets of testing, manual debugging and automated unit testing. The server will also undergo load testing in order to verify it can handle the expected number of concurrent users.

Manual Debugging

As I implemented every feature, I tested them extensively with a variety of inputs to verify that the software is acting as expected. This level of testing focused on the larger scale of the feature and so did not send specific inputs to the individual methods. However, this level of testing does include interacting with the UI and making sure that it reacts to user input appropriately, but also testing the server endpoints to make sure they return current data from the database. This level of testing uses the live server and not a mock of the methods and so could be unpredictable in exactly the results presented, however is reflective of how the app would be used and so gets a good perspective on the app. To be clear, this perspective is not as good as the feedback that would be given by actual users.

Automated Unit Testing

For each public method implemented in the client side of the app, barring a few exceptions, unit tests have been written to automatically give the method a variety of inputs to check it returns the expected result. For this level of testing, mocks have been used to return data from the database and from local storage, this is to ensure consistency in the inputs to the test. Due to this reason any method that’s sole purpose is to retrieve data from the server unedited, has not had tests written for it as these tests would not accomplish anything. Unit testing is rigid and predictable and doesn’t reflect how actual users would interact with the program, however it does help you check you haven’t inadvertently broken an existing part of the program. The act of writing unit tests also makes you think about what the software is doing and how it accomplishes its task, this both helps with validation and verification. There have been many times when I have fixed a bug I didn’t know existed through writing unit tests, not through the output of the test being wrong, but through getting another perspective of the code written.

Load Testing

In order to test if the server can support the desired number of concurrent users, I have designed a load testing experiment that can be run efficiently and repeated as the server is updated. I have written a poster on this experiment and the results that I got when I conducted the experiment. To summarise the findings of this experiment, here is a quote from the conclusion: “This performance evaluation has shown that the Graduation Gathering server functions properly up to 300 concurrent users and seamlessly up to 70 concurrent users when subjected to realistic test conditions. However, this is far from the 2000 concurrent users that the Graduation Gathering server is expected to handle.”