Development Testing:

Our development testing strategy follows the principle that testing is used to show the presence of bugs and not the absence of bugs. We will go about adhering to this principal by following the protocol of first writing the tests before writing the code, making sure these tests revolve around identifying if the code is conforming to the specification and if the product is actually being built correctly. After doing this, we will continuously re-run, refactor and optimize the tests. The correctness of these tests will be ensured by using synthetic data that is chosen using methods such as equivalence partitioning to provide complete coverage of test cases without needing to try all possibilities. All automated tests will be implemented using JUnit.

A core function of the project is to have an application that allows users to find, create, delete, update and register for events that are organised by the university Health and Wellbeing Department that aren't currently integrated into the university Legend system. A core component of the application is to have different levels of user privileges and that users who had the administrator privilege had the ability to create, delete and update these events while more importantly be able to register attendees. This was identified as a core component by establishing that this was the main aspect of the clients vision of a 'minimum viable product' which aimed to solve the problem of not being able to efficiently track analytics of events that aren't currently integrated into the system.

Test Plan:

Test	Type	Input	Reason	Expected Outcome
Event creation	Unit Test: - Automatic - In-memory database and JUnit	User with admin privileges creates an event using application interface. - Date - Location - Time - Capacity	Required in client specification. Testing for input validation and prevention of duplication of events.	If event doesn't already exist/overlap with events at that time and that location then an event is created and displayed on application and imported into database.
Event deletion	Unit Test - Automatic - In-memory database and JUnit	User with admin privileges deletes an event using application interface or event is deleted after it expires.	Required in client specification. Making sure application doesn't continue to display out of date events but information of past events remain.	If event exists, when manually/automatically deleted it is removed from application interface but not from database.
Event attendee registration.	Unit Test - Automatic - In-memory database and JUnit	User with admin privileges registers attendees using application interface.	Required in client specification. Making sure number of attendees doesn't exceed capacity.	After registering attendees, information of event is stored in database.
Separation of user privileges and restrictions on user access.	Unit Test/Security Test - Automatic - MockWebService	User interfaces and privileges are displayed differently depending on account.	Required in client specification. Making sure non-admin users can modify events without permission.	Different application interfaces and actions are displayed depending on account privilege.
Check all user inputs/event data is reflected in database.	Integration Test: - Automatic - In-memory database and JUnit	Integration of all unit tests.	Making sure all unit tests /inputs are reflected in database records.	Database records are updated according to user input.
Application event limit handling.	Stress/Load Test: - Automatic - In-memory database and JUnit	N/A	Making sure database can synchronize with multiple event creation/deletion/registration at the same time.	Information is synchronized in the database.

Release Testing:

Our release testing strategy follows the principle that a test should identify if the current product/a new feature is meeting a user story or client requirement and ultimately that the product is the correct solution to the client's problem and meets all client requirements and user needs. We will go about adhering to this principal by emphasising user experience testing, constantly getting feedback from the client and running both automatic and manual tests using methods such as acceptance testing and automated API calls. After doing this, we will continuously re-run, refactor and optimize the tests based on real user assessment and the products reliability, performance, usability and robustness.

A core user story of the product is for students to be able to access an application that compiles all university wide and society events that are available to them and act as a focal point of information regarding the societies they are interested in/apart of, their upcoming events and a way to navigate and register to such events. This is as a core user story as we identified along with the client that there is currently no coherent way for students to find, register, show interest and get details of university and society wide events other than navigating Facebook and searching for these events and specific societies.

Test Plan:

Test	Type	Input	Reason	Expected Outcome
Application works on different environments and displays properly on all Android devices.	Production/System Testing - Manual	N/A	Important to make sure the application works on all relevant devices so all users can access it.	Application works coherently across all relevant devices.
Automated API/CRUD Calls	Production/System Testing: - Automatic	Automated calls using JUnit.	To make sure system and features work as they should when handling api/crud calls.	API/CRUD calls are handled correctly.
Implementation of additional features and release verification.	Regression/System Testing - Automatic and Manual	MockMVC	Making sure that additional features interact correctly with system even if the system isn't fully functional/developed.	New features don't cause errors in fundamental features.
Basic Release Test	Smoke Test - Manual	Manual user input/interaction	Making sure most important functions work and allows us to verify the stability of product and allows us to continue testing other components. Allows us to monitor server and application metrics.	Product complies with most basic user requirements and meets user story.
User/Client Interaction	Deployment test		Provides testing environment that can't be replicated in an automated/lab environment.	Reliability, usability and robustness of application is confirmed. - Feedback is taken from user.
Client/User Audit	QA Test: - Manual	Client/User manual interaction with application.	To make sure user/client needs are satisfied and that the correct processes are being taken to fulfill the client/user story.	Product complies with most basic user requirements and meets user story. Feedback is given for continuous improvement.