5 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.

The testing plan should connect the requirements and the design to the adopted test strategy and instruments. In this overarching introduction, given an overview of the testing strategy and your team's overall testing philosophy. Emphasize any unique challenges to testing for your system/design.

In the sections below, describe specific methods for testing. You may include additional types of testing, if applicable to your design. If a particular type of testing is not applicable to your project, you must justify why you are not including it.

When writing your testing planning consider a few guidelines:

- Is our testing plan unique to our project? (It should be)
- Are you testing related to all requirements? For requirements you're not testing (e.g., cost related requirements) can you justify their exclusion?
- Is your testing plan comprehensive?
- When should you be testing? (In most cases, it's early and often, not at the end of the project)

In this section we discuss our plan for conducting various testing procedures to validate the implementation of the project. We note that the requirements (functional and non-functional) were already translated into the architectural components discussed in Section 3 of this document, and we refer to corresponding figures from that section here.

Given that the project involves large (simulation based) datasets, along with execution of algoritms and visualization of the output, there are certain distinct characteristics of our testing.

We note that the main objective of this project is to develop a proof-of-concept implementation, not a complete commercially available prototype. As such, the cost component is not a major testing factor.

5.1 Unit Testing (Rowan Collins)

There are a handful of aspects in our proposed design that require unit testing to ensure correct functionality of all systems. Unit testing is essential to scale functionality of our project. We will be testing in parallel with new code development. Below is listed our proposed ideas for unit testing:

- We need to ensure consistent algorithm output. By running JUnit tests on an algorithm given specific parameters, we can ensure that output is always in our generalized file format. While we can't test the output, as algorithms being used may vary, we can test that an output file is formatted correctly.

- We need to make sure the system can detect a simulation with parameters that have already been run by running JUnit tests to grab data from an output file and make sure it matches with preset parameters.

- Our frontend system needs to function properly and translate the proper data to the server for drone visualization. In order to ensure correct functionality we need to ensure:
    - parameters are passed as selected by the client
    - the account responsible for creating the request is passed with the parameters
- Our notification system needs to function properly in order to notify the correct users that the simulation data is up. In order to verify the functionality we will need to make sure that:
    - the notification system correctly identifies the user who submitted the request
    - the notification email contains the formatted file attatched
- Our user registration system will need to be tested in order to be certain that duplicate users cannot be created.

While this list isn't completed, it's an outline for what's to come. Unit testing will be vital in order to provide a functioning and reliable system, so new items will be added in the future. To provide a scalable system our group will aim for near perfect line and case coverage of our application. Test input will be small in comparison when compared to data being computed in production. Using small sample sizes when testing will help us predict with higher accuracy when unit testing.

Tools: JUnit, Mockito

5.2 Interface Testing (Joe)

To communicate within the application, the following interfaces are defined: simulation setup, simulation parsing, and algorithm notification. The simulation setup interface interprets information from the frontend and creates a simulation. The simulation parsing interface converts JSON into objects to display on screen. And finally, the notification system alerts the frontend on the status of simulations.

Testing user selection of simulation setup with backend:

To test the simulation setup interface:

- Sample JSON files given to the interface and results will be compared to other known results of the algorithm.
- Invalid arguments will be given to the interface to ensure the interface rejects invalid data.

To test simulation parsing interface:

- Partial simulation results in the form of JSON objects will be passed to the interface to display, verify that each drones, locations, and phenomena display individually.
- Complete JSON files will also be passed to the interface to verify that all objects are able to display at once.

To test the notification interface:

- Sample notifications will be created and passed to the interface which the frontend must display.

None of the success of each the setup, parsing, and notification interfacing test shall depend on each other, and thus each may be developed and verified in parallel. This also supports the Agile methodology by further compartmentalizing functionality.

Tools: Jest, Mockito, JUnit

5.3 Integration Testing (Jacob Houts)

There are a couple critical integration paths that need to be tested. The first path is between springboot and the database, and the second path is between springboot and the React frontend. These paths are critical because we pass algorithm information, simulation information and user information between these paths. The data is sent along these paths by using the REST API that we are implementing using springboot. These API Calls need to be tested to ensure correctness and efficiency. Our primary way of testing these API calls will be to send specific data to a known location, then ensuring that this data arrives at said location unmodified. These tests will be conducted with input data on a smaller scale then production data.

Examples:

- Test that we can send a user inputted algorithm and store it on the database.
- Test that simulation information gets processed to the frontend correctly.
- Test that the notification system on the front end works when the notification is triggered in springboot.

Tools: Mockito, Postman

Many of these tests will be conducted in parallel to in order to reduce the amount of time and resources this testing will take. This also is in accordance with Agile methodology, which we are using for our development strategy.

5.4 System Testing (Jacob Houts)

For system testing we will use the functional requirements as criteria. Refer to section 2.1.1 of this document. Key requirements that need system testing.

- The software shall provide a visualization of the drone flight.
    - The software shall have a 2d grid layout of the geographical area.
- The software shall save previously run simulations for input combinations that will be accessible for other users.
- The software shall allow the user to login to a personal account profile.
- etc.

Each of these requirements require the entire system to be operational in order to test. These tests will be conducted using a combination of unit, interface, and integration tests. In order for these tests to be deemed successful, they need to finish with the correct result, within a timely manner, without using too many resources.

The tools used throughout this process may include (but are not limited to):
Postman, Mockito, Jtest, Junit.

5.5 Regression Testing (Jaden Forde)

How are you ensuring that any new additions do not break the old functionality? What implemented critical features do you need to ensure they do not break? Is it driven by requirements? Tools?

Regardless of any changes that are made to the application, it is vital that the web interface remains accessible online. Of course we also need to ensure that the simulation is accessible and functional, and any other functionality should remain unless it is intended to be removed by the update.

We will manage this principally by developing tests for components as part of their development. This is an important part of our agile approach to this project's development, and will allow us to ensure all desired functionality is present before anything reaches deployment. For example, if I was creating a component to import new drone algorithms, I would also write unit and integration tests to demonstrate proper operation before any code is reviewed. Similarly, if I was updating a component, I would need to write new tests showing the new functionality. For these tests to be meaningful, they must of course completely demonstrate the behavior our application should exhibit.

These tests will remain, and each test set will verify proper operation of their respective components. Therefore before any new changes and tests are deployed, all existing unit, integration, and system tests must be run to ensure that existing functionality is retained.

These tests will run via our CI/CD pipeline once code is pushed to our remote version control system.

These tests should also be run when non code changes are made, such as the addition of new algorithms and phenomena.

5.6 Acceptance Testing

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?

Functional Requirements:

- The software shall give the user the ability to choose which phenomena (e.g. fire, explosion, ...) to apply to the drone flight simulation.
    - **Acceptance criteria: User is able to select a file to be used to describe the event phenomena.**
- The software shall give the user the ability to choose which drone algorithm(s) to apply to the drone flight simulation.
    - **Acceptance criteria: User is able to select a file to be used for the fleet algorithm.**
- The software shall accept input combinations through selected files.
    - **Acceptance criteria: User selected files are used in the running of the simulation.**
- The software shall provide a visualization of the drone flight.
    - **Acceptance criteria: The software shall have a 2d grid layout of the geographical area. There should be identifiable icons for the drones and the events across a timeline of running.**
- The software shall calculate drone statistics and values over time and record them to storage.
    - **Acceptance criteria: Output of running the simulation should have drone statics in it's output file. The output file should be stored in a location to be retrieved at a later data (ie database).**
- The software shall provide a view of statistics around the drones (battery life, location, speed, etc).
    - **Acceptance criteria: Battery life, location, speed and other client specified data points (to be documented) should be in the output file of the simulation.**

- The software shall save previously run simulations for input combinations that will be accessible for other users.
    - **Acceptance criteria: Output file should be stored in a database.**
- If the user selects an input combination that has been simulated before then the software shall return back the already run simulation data.
    - **Acceptance criteria: Repeated input combinations should retrieve previously run simulation output files, instead of re-running the simulation. ie Repeat input should not result in a new simulation output file in the database.**
- If the user selects an input combination that has not been simulated previously then the software shall queue the simulation to be run on the backend.
    - **Acceptance criteria: New input combinations should result in another output simulation file in the database.**
- If a simulation is run from a queued input combination then the software shall notify the requesting user when the simulation is completed (push-based notification).
    - **Acceptance criteria: For new input combinations, the user should receive a notification (via the webpage or possibly email), after the simulation is run.**
- The software shall accept user input (file) which is given in source code which is ready to run (compiled or interpreted, etc).
    - **Acceptance criteria: Python files should be accepted as valid algorithm files.**
- The software shall store the simulation output in a format containing: starting location, starting time, destination location, arrival time, trajectory.
    - **Acceptance criteria: The output simulation file should have starting location, starting time, destination location, arrival time, and trajectory.**
- The software shall have 3 UI components: list of algorithms to pick, list of event phenomena input, visualization screen of simulation output.
    - Possibly a 4th component for notifications.
    - **Acceptance criteria: Self-evident.**
-

Non-Functional Requirements:

- The software should be easy to use and understandable since not all of our users have a technical background
    - **Acceptance criteria: A new user, given an example event phenomena file and algorithm file, should be able to run a simulation and view the results within 30 min of being introduced to the website.**
- UI elements of the software shall be intuitive and clearly labeled or documented.
    - **Acceptance criteria: A new user should be able to correctly identify the purpose of each UI element (buttons, drop-downs, etc), within 30 min of being introduced to the website.**
- The software shall handle errors in the input gracefully.
    - **Acceptance criteria: For a selected file that is not correctly formatted for it's selected use, there should be a red colored warning message about the selected file being invalid, and a short description for the reason behind the error.**
- The software shall combine concurrent access for already executed input combinations and will run push notifications for users with new input simulations.
    - **Acceptance criteria: The user should be able to see and select the result of previously run simulations.**

- The software shall be compatible with Windows, MacOS, and Linux.
    - **Acceptance criteria: A user on all 3 types of machines should be able to access and operate the website.**
- The software shall be developed in a manner that is supportable & maintainable after our team leaves.
    - **Acceptance criteria: Each function, and non-self-evident piece of code, shall have a single/multi-line comment to explain it's purpose. The software should be developed in a manner that allows for modularity and the possibility of adding additional features.**

5.7 Security Testing (if applicable)

Despite security not being implicitly stated in our project description, we feel it is our responsibility to uphold a certain level of scrutiny when it comes to protecting sensitive data. Since users of our app will be giving us usernames, emails, and a password we have a duty to make sure this information is not shared with unintended audiences. We will ensure the safety of our users by using the latest encryption techniques, such as AES,to secure passwords and other sensitive information.

Asking for help from other security oriented students, and our own team, to scan for vulnerabilities in our database or possible flaws in our web application would be beneficial. However, since security is not a primary concern of our project, we believe no detailed plans for security testing would be necessary for the scope of our design.

- The software shall allow the user to login to a personal account profile. (Functional requirement)
    - **Acceptance criteria: User with credentials is able to login.**

5.8 Results

What are the results of your testing? How do they ensure compliance with the requirements? Include figures and tables to explain your testing process better. A summary narrative concluding that your design is as intended is useful.

At this point we do not have a product to test against.