4 Testing

With this project being purely a software development project, testing will occur on exclusively the software side. That being said, it is recognized that testing will be extremely important for ensuring the validity of the code that each developer will produce, and subsequently ensuring that requirements are appropriately met. The tools that will be used will vary depending on the area of the development (frontend, backend, etc), but will be along the lines of software testing tools of the various languages and frameworks that are being utilized for the development of the project.

The testing strategy that will be utilized was agreed upon by the team, and uses modern practices of software testing in a development environment in accordance with existing software testing standards. These various testing components will be performed alongside the agile development methodology chosen for the development process of this project.

4.1 Unit Testing

We are going to utilize two testing frameworks for our separate codebases. First, for the client-side application, we will be using the built-in testing framework “react testing library” with “jest.” This gives us the ability to test individual React components, custom hooks, functions, and mock up items between them as necessary for as much granular testing as possible. It can simulate individual unit tests as well as simulate user input and firing of events to give us some UI/UX testing as well, although not as much as would be preferred by QA engineers, for example.

The server side application will use the builtin go testing framework. This will give us a lot of control over what is tested server side, so as not to go so far into testing where we are just testing the library. Individual functions, components, and modules will all need to be thoroughly tested and have those tests passed in order to be accepted into the main branch,

The unit tests will be a baked in component of the overall CI/CD pipeline as well. Should any tests fail or act not according to the testing criteria in the automated tests, it will not be accepted into the main branch. While developers are given liberties as to the specifics of their testing, one popular example the team is encouraged to follow is the ZOMBIE testing methodology.

4.2 Interface Testing

In this software system, we have two primary interfaces: the User Interface (UI), and the Application Programming Interface (API). The UI is the web-based interface that users can interact with to perform tasks as described in the requirements. It can take in commands and display the results to the users. The API communicates between the front-end and the back-end, sending users requests generated by the UI to the algorithm, and sending completed results back to the UI to be rendered and displayed.

To test the API, automated tests such as the Unit Tests described in section 4.1 and integration testing as described in section 4.3 can be performed periodically, after each code push. This testing will ensure that the API is compliant with the requirements and prevent regression defects as development continues.

Similar to the API, the User Interface will also implement automated tests that are executed periodically. In addition, we will use manual testing to ensure that the user experience requirements from our clients are met as well. Each component's design will include a section that describes a UI test case, in the form of a step-by-step checklist. During testing, the validator will interact with the UI as described in the test case, and verify that the UI's behavior is exactly as expected. This combination of automated and manual testing will ensure that the interface meets the requirements.

4.3 Integration Testing

There are a few integration paths for our design. One is between the front end and back end, which we have designed to be via an API layer. There are also integration paths with how our software will interact with the hosting system, which we may also test, to ensure that the application is functioning correctly and is accessible to the users. For testing the integration between the front and back end, we have a few options on tools.

One such option is using a headless testing browser environment, such as Selenium. This will hook into a headless browser and run a full battery of tests against our entire application. This will ensure that the frontend and backend are producing expected results and are integrating together successfully and that they are producing the correct output to the user.

Another option we have is a test kit that hooks into the frontend, and triggers it to make requests to the backend, just as a real user would. This software may be partially custom written by us to best hook into our application. We will most likely be modifying a unit testing toolkit to make these requests.

All these tests will be run automatically within the confines of the CI/CD pipeline. We will probably be utilizing Docker or a system like it to spin up databases and other system dependencies so that we can test in an environment as close to production as we can. This will also allow us to create new databases and tables as we need, without disrupting the production environment.

4.4 System Testing

System testing is a level of software testing that validates the complete and fully integrated software, and as such will occur after the process of unit, interface, and integration testing.

Considering the plan is for the unit tests to be extensive and required for eventual merging into the main branch program this series of testing should occur with relative ease. Especially when considering that the integration testing will automatically ensure that these tests will properly run and integrate when the code base is sent to CI/CD pipeline on the project's GitLab. Due to this, the process of 4.5 Regression Testing will become a part of the system testing as it will pertain to the connecting of various developers code into the single system, and will thus require testing alongside general system testing for ensuring existing functionality in the system is not broken.

From all of this, the system testing will expect that all existing functionality will not be broken by new integration as well as testing the new functionality that the code being merged into the system remains from its individual testing.

As far as what will be required for this, will be the same as the previous sections (4.1 Unit Testing, 4.2 Interface Testing, and 4.3 Integration Testing). There should not be any need for additional tools for system testing as it will ensure the overall functionality of the system after merging of new functionality. This will mean that additional testing may be required to show full functionality of the system from end-point to end-point. Ideally, to ensure existing developer bias, this series of test creation and testing will be performed by someone other than the developer of what is being merged into the system.

In order to connect this to the requirements, any additional tests will be focused on the functionality of the requirement that the task being merged into the main system is supposed to create.

4.5 Regression Testing

We will be using gitlab’s integrated CI/CD tools to verify that when a new push to a feature branch or master is made, the automated test suite to verify that functionality is intact will run, and should any errors occur, gitlab will either prevent the merge, if the request is made against master, or let the committer know the errors if it is a push to a feature branch. This way we will avoid cases where a new feature merging into the rest of the project potentially breaks the master or production branches, and feature branches will have assurance that all required features are working in the grand scheme of the project rather than just one developer’s computer.

Ideally, there will be no issues with integrating new features into previous main builds of the project, however should a feature break one or more parts of the master branch, GitLab’s CI/CD suite will be able to tell us what exactly is conflicting or breaking what so we don’t need to guess at any point. The most basic regression test for us should see that the algorithm itself is not interfered with in any way, as it is the heart of what will make this project successful. Next, ensuring the backend calls to that algorithm stay functional will be most important as the app has no functionality without being able to get results out of the algorithm itself. Next most important is being able to interact with the backend through the user interface in the frontend, since without a human interface, no one will be able to use our application. These critical

features should always be tested for functionality with new builds and new features before those new features are merged into the rest of the project.

4.6 Acceptance Testing

For our acceptance testing, there are three primary steps. Should any step result in actions to be taken, the process will start again. The first step is the full implementation testing by the team. This is to verify that all the features we wanted to implement are completed and ready to move on to the next step. We will iterate through our requirements document and check that each requirement is being met. Step two would be a very similar process of going through the requirements documentation, but this time with our client Matt Wymore. In doing this, we can get their perspective on if the requirements are being met to the degree that they expect. The final iteration is then meeting with Alliant Energy, our industry reference. We repeat the process of going through the requirements, showing off the implementation of those requirements, and getting a consensus on if the requirements have been met. Additionally, should new requirements be requested, we can evaluate the feasibility of the new requirements, and either go back and implement them, or discuss alternatives.

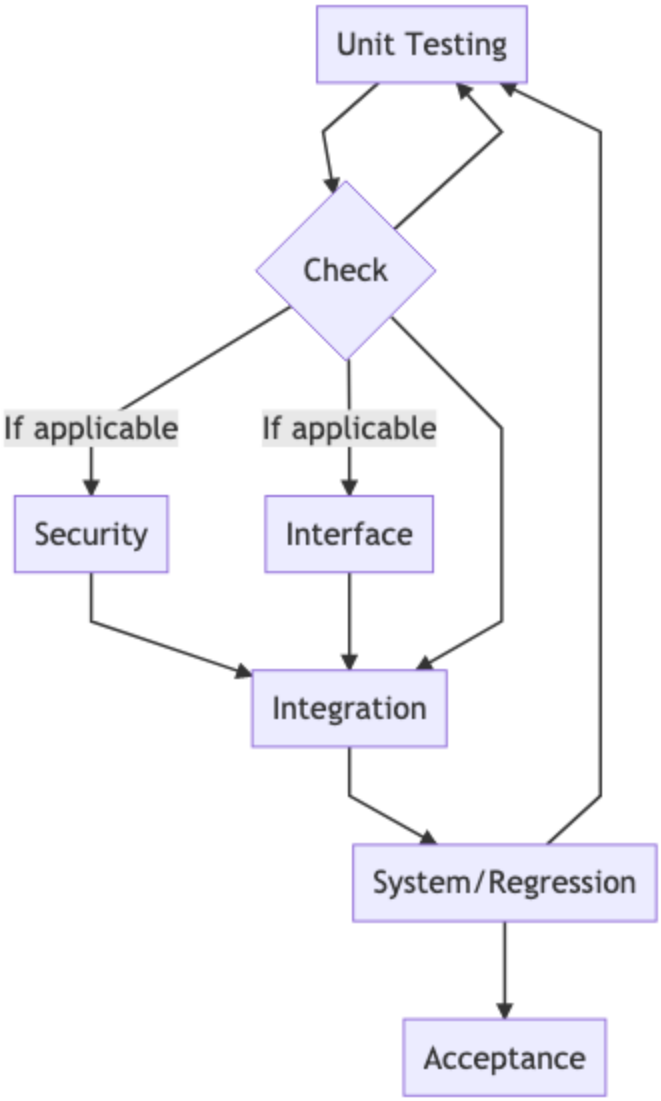
4.7 Security Testing

As we develop the application, along with the server side and client side components, it is important that we take into account the application of modern security procedures. As such, for components like the frontend web application, we will only send information that is explicitly needed. As we configure and set up the server side of the application, we will conduct a penetration test to ensure that access cannot be obtained beyond those who are authorised to have it. Additionally, prior to application publishing, removing any unnecessary accounts that may linger on the server and closing any ports that are unnecessarily open.

4.8 Results

At this time, the only results of existing testing is some basic smoke test results ensuring that the basic framework will correctly compile and launch.

The main method for ensuring compliance with the requirements of this project will be from the overall method of the project development - not just testing. This method is an agile methodology where a developer will take various tasks/issues per sprint that focus on a certain functionality that may be a full requirement, or a part of an overarching requirement. Once reaching the testing phase, compliance will be further ensured by having comprehensive testing that will be required to succeed for integration into the main branch of code. This testing will involve everything from checking validity of code to evaluating if the new code can correctly perform the specific task that it was intended to based on the requirement(s) that it was intended to address.



Shown above, is a visual representation of the project's testing process. It begins with unit testing of the code that is being developed which will repeat as long as there is more functionality to be tested. Subsequently, once all unit tests have been completed and passed by the new code, the developer will check if either security or interface testing will be needed (as not all functionality will). After that has been completed and successfully passed, integration testing will occur. After successfully testing the integration of the new code it will be merged into the main branch where system and regression testing will occur in unison. At this point, if more functionality is to be added the cycle will repeat with the creation of the newer code for the remaining functionality. Otherwise the project will be complete and reach the acceptance testing phase. This will all occur with the agile development methodology meaning that the unit to system/regression testing phases will repeat many times, and at least once for each added functionality.

At this time, there is not a summary narrative concluding the usefulness of this testing design.