5. Testing

For the different areas of the project, there will be individual tests that ensure each section of the component works and fails when intended, and the system as a whole will be tested. One of the unique challenges with this design is testing the functionality of the AI model rather than testing just the accuracy. Testing the accuracy is straightforward, as the model is run on test data and evaluated.

5.1. Unit Testing

AI Model: We will need to test the AI model for accuracy separate from the other components. We will have a baseline model that provides a reference point for comparing the performance of the model. Looking through the provided errors will show the weakness within allowing us to focus on the weak parts of the model. The data set will be split into two parts where one part can be used for training and the other used for testing. In addition to testing the accuracy of the model, we will test whether the code for the model does what is intended. We will integrate our model testing into a CI/CD pipeline.

Individual Cloud Functions: Our backend cloud functions will each need to be tested individually. This will include validating data sent by the user and data parsing for the response. Our back-end cloud functions will be tested using the PostMan API. For each individual cloud function, there will be a corresponding unit test that sends a request and validates the response, along with unit tests that ensure failures and errors are correctly handled by the cloud functions. In most Python scenarios, we can use the unit test framework to run and validate our tests.

Basic front-end component testing: Our front-end will be made up of components such as buttons, text input fields, etc. We should test the behavior similar to how a user would use the application using libraries such as the React Testing Library (RTL). This library is useful for asserting state or values from front-end components such as text entries, if a button is disabled or enabled, etc. We will also use Jest to run the tests, and confirm whether they failed or succeeded.

5.2. Interface Testing

The front-end interface of our project is a front-end built with React. The React Testing Library will be useful in writing the interface tests by creating a virtual DOM for the tests to run in. We will be running the tests Jest, a JavaScript testing library that is known for its simplicity and isolated tests. Isolating our tests lets each component of the interface be tested without the influence of another.

The backend interface will be built using cloud functions written in Python. Similar to the unit testing stage, the unit test library should provide what we need to test the entire backend suite.

5.3. Integration Testing

One of the major integration paths we will need to test is connecting the front-end to and from our cloud platform. This will involve careful planning from both sides to ensure it works as intended. To ensure that the front-end is receiving calls, we can mimic a response that uses the same path as a real response would. Much like the interface testing, this can be done using Jest.

Our backend cloud functions also have to make calls to both our database and our AI model. We should test these together to ensure we can read/write data to a mock database and send mock calls

to the AI model. This testing framework, mock from unit test, is specifically designed for the mock calls that trigger the cloud functions.

5.4. System Testing

To test the entire system we will need to make sure we have a list of all the requirements our project should meet. This will allow us to create a test plan that outlines the key functionalities. We will continue to use unit tests in our case Jest to make sure they meet their specifications. For integration testing, we will use end-to-end testing to test user interactions. For backend we can use Postman to validate the communication between the front-end and backend. We can also do performance testing that can test response time. This should also be continuously monitored for security, performance, and user experience.

5.5. System Testing

We will build new components and test them with the old components before merging them to main. A CI pipeline will be established to automatically run these tests for each new branch looking to merge. By keeping new components and features in separate branches, we can mitigate risk associated with regressing or damaging finished features. On the ML side, since our model will undergo multiple training iterations on the same data, we aim to prevent overfitting by employing a subset of the data for testing. This allows us to assess whether the model is overfitting or not. Our strategy to prevent overfitting will be a notebook with k-fold cross-validation. This enables automatic assessment of whether the model is improving or succumbing to overfitting for our dataset. Moreover, this method will be useful in the beginning of our project for selecting the ML approach that best suits our data.

5.6. Acceptance Testing

To demonstrate that the design requirements are being met we can establish clear traceability between the design requirements and the testing phases. This can be done by having each requirement aligned with a list of test cases which will ensure that every requirement is tested.

To involve the client in the acceptance testing we will have them test the project to see if it works as they would expect it to. The client can provide feedback during this process which can range from defects, deviation within the requirements, or places where the software doesn't meet expectations. This will also ensure that both functional and non-functional requirements have been met and that the final product aligns with the expectations.

5.7. Security Testing

It is important to ensure the privacy and safety of user data and to maintain the accuracy and reliability of the predictions. In order to do that we will make sure that the dataset used for training and testing the model is anonymized and does not have any personally identifiable information. In order to do this we can implement data encryption mechanisms to protect the sensitive data during transmission and storage. We will also validate input data to prevent malicious input that could lead to incorrect predictions. We will lastly use authentication and authorization mechanisms to control user access to the model.

5.8. Results

We have completed basic tests to confirm that AWS can host an AI model. This uses the SageMaker service with its integrated Jupyter notebooks. Our test consisted of creating a notebook and writing

some Python code to ensure that a basic, algorithmic machine learning model could be created and run on some data hosted in AWS. While our model will use neural networking, the test confirms that the foundational steps exist to create the model.

We have also tested hosting a front-end page utilizing a different AWS service called S3, a simple file storage system. This will help us keep our different components within the same overarching framework of AWS. As shown in figure 1, the link to the HTML page we created is publicly available, meaning that the service can be reached from anywhere once it is fully built out.

Figure 4: Result of basic front-end test using AWS to visually show an HTML page deployed to an AWS URL.