Test Strategy Document

Demand forecasting of energy consumption of air conditioning in a commercial complex

Team number - 12
Akshat Goyal, Kanish Anand, Nikunj Nawal, and Sridhar M

Scope

This document will be reviewed by Mr.Malliswar who is our client form the company INDRIYN DATA ANALYTICS PRIVATE LIMITED. He will take into account the basic functionality of the webapp and see if all the predictions regarding energy consumptions made are close to actual values.

Test Approach

Process of testing:

- Isolate the development environment from the test environment
- Write test cases that are independent of each other
- Aim at covering all paths through the unit
- We will do selective testing and will cover major cases
- Beforehand we will be doing function coverage
- Also, check the APIs using tools like mocha, postman
- Follow the Whitebox testing while development
- Do the control-flow test
- Do data flow test
- For the integration testing, considering the scale of the project we decided to do *critical first* integration technique
- Also to have a user perspective, do Greybox testing

Testing Levels:

- Do feature acceptance testing
- To finalize the code do the unit testing
- Check the APIs using tools like mocha, postman.
- To check the working of designs do the integration testing
- Regarding the requirements do the system testing
- Setting up the test for a particular piece of an application (called the system under test)

- Observing the resulting behavior and checking whether expectations were met or not.
- Do the acceptance testing again to verify the demands of the client
- So far we have done initial acceptance testing, system testing, and integration testing for backend and frontend parts.

Test Environment

- We have done unit testing and integration testing for backend and frontend parts using jupyter notebook and UI was tested on the localhost having node, react and express installed.
- The data for the training model was given in CSV file data was loaded to train the model using a python script.
- Running internet service is also required to fetch weather and temperature data.
- The code is uploaded on GitLab for efficient version control.
- We have created a copy of the initial raw data provided by the client.

Testing Tools

- Automation and Test management tools needed for test execution
- Figure out the number of open-source as well as commercial tools required, and determine how many users are supported on it and plan accordingly testing Tool
- Mocha is used to automatically test rest APIs in node.js
- Postman is used for testing APIs
- Energy Prediction values are tested through test set accuracy on our model using model.score() function.

Use Cases

1. View predicted values for energy consumption on Jupyter notebook
2. View energy consumption graph of data provided on Jupyter notebook
3. View predicted value for energy consumption
4. View actual data of energy consumption
5. View graphical analysis of energy consumption
6. Download the generated energy consumption prediction graphs
7. Download the given energy consumption data in form of graphs
8. Download the generated budget prediction graphs

9. View budget Prediction
10. Login and Register for different roles for the system
11. View anomaly cases as a list
12. View notification for an anomaly
13. View current energy consumption
14. Check the total consumption of energy
15. View weather forecast
16. Select API for weather forecast
17. Update profile of the logged-in user
18. Update the email for receiving the anomalous data updates
19. Get help regarding the web-app
20. Give user feedback via the web-app

Test Cases

Write the test cases derived from the use cases. The written test cases should be arranged by the use cases.

1. Plot graph of predicted values for the entire year
2. The output should correspond to the given data
3. Plot graph of predicted values for the entire year
4. Total consumption in given data should match the output checked via google spreadsheet API call using code
5. Verify output via graph generator API call using code
6. Verify using the web technology already available in browsers
7. Verify using the web technology already available in browsers
8. Verify using the web technology already available in browsers
9. Run code to calculate the budget of actual data and verify it by passing the data in our system
10. Try registering multiple times using the same user name. Login with different combinations of inputs
11. Pass data that will create anomaly and check if it makes to the list
12. Manually check the notification for the forged anomalous data
13. Cross verify the shown value with the data obtained
14. Total consumption in given data should match the data obtained
15. Run the test API and also verify with open-weather site
16. Selecting API should give value corresponding to their site
17. Updating profile should update values in the backend data
18. Updating email should update the value in the backend data

19. User should be able to contact the maintainer/developer
20. Feedback form should save the input and mail to the help person