Machine Health App

Thankyou for this opportunity, which allowed me in learning new technologies and frameworks.

This app calculates the machine health for a given machine. It also provides the overall factory health as well using the scores of all the machines and parts.

Backend - Test scenarios

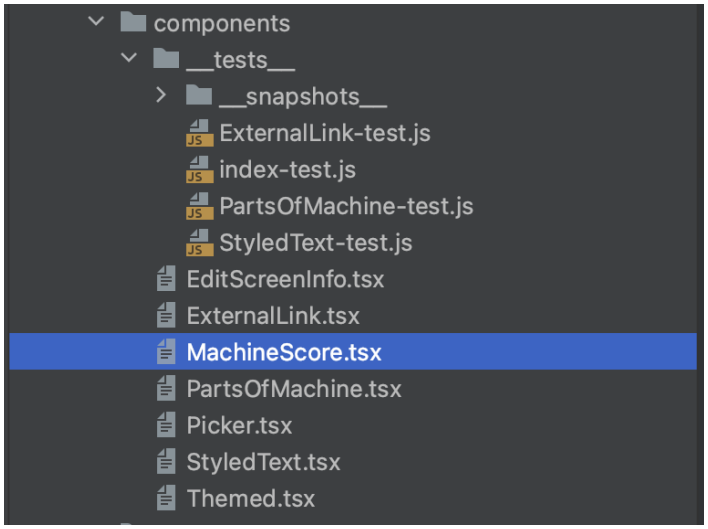
I created a test file called “calculate-machineHealth-test.ts”, in which I captured the following scenarios,

- Whether the machine-health API call is successful or not - By making an axios request call.
- I created a Map of request and the corresponding response values(hard coded the values for now)
- Whenever an API call is made with the requestBody from the Map, the response retrieved is compared with the expected response values from the map.

To improvise in this code(TBD),

- Instead of hard coding the response values, i would like to improvise on calculating the machine health with the expected logic(which ideally has to be from the requirement document).
- Generate random values for all the machine parts based on the normal and optimal Range provided in “data/machineData.json” file.
- Capture more invalid scenarios with the different range of values referring to “data/machineData.json” file

Native-app unit testing,

- I created snapshots for ExternalLink.js,PartsOfMachine.js and index.js
- 

I have created a new branch called “MachineHealthUpdated”.

Pushed the updated code to git. To run the tests, navigate to the __tests__ folder under native-app and npm run test.

More detailed Testing for the UI(TBD),

- Field Validations for Machine Name and Part Name
  - When the user doesnt select any options and try to Save appropriate error message should be displayed.
  - Only values from “native-app/data/types.ts” should be displayed in the UI.
- Validation for Part value
  - When the user enters string/char/long/byte/any other data types other than float/int appropriate error message should be displayed.
-

We have to make sure that all the elements are visible and values are populated in the dropdown. I would craete a behaviour testing framework using pupeeteer to capture the below scenarios,

- Opens the web page
- Verify the web elements(Calculate Health, Reset Machine Data, Machine State, Log Part and the URL to LogPart page) are visible and clickable.
- Click on Log Part URL/Tab - should open tab/two
- The Machine Names drop down should display all the values from “types.ts(`enum MachineType`
-)” and PartNames should be displayed from partInfo.
- When a valid value as mentioned in the range is entered and saved, the data should get saved successfully.
- Click on Machine State -> Calculate Machine health ->Should send request and make an API call and the response should be displayed.

Integration Testing

- For the **Integration Testing** of this project the ideal flow has to be,
  - The native-app has to call the backend files.
  - Open the browser and select the Machine names and Partnames→Save and Calculate Health of the machine
    - In this, Calculate Health of the machine, is where we have to verify the API request/response. So, with the current implementation, When i run native-app/components/__tests__/calculateHealthPage.js —----->Should call the backend calculation logic —---------->backend/__tests__/calculate-machinehealth-valid-testcase.ts—---------> Verify this data with the value displayed in the UI.
    - Backend logic to be calculated using the different range of values given in the file “native-app/data/machineData.json”.

- POSTMAN allows the user to upload files with feild values to run multiple requests. I used it to help me understand the request/ response behaviour.

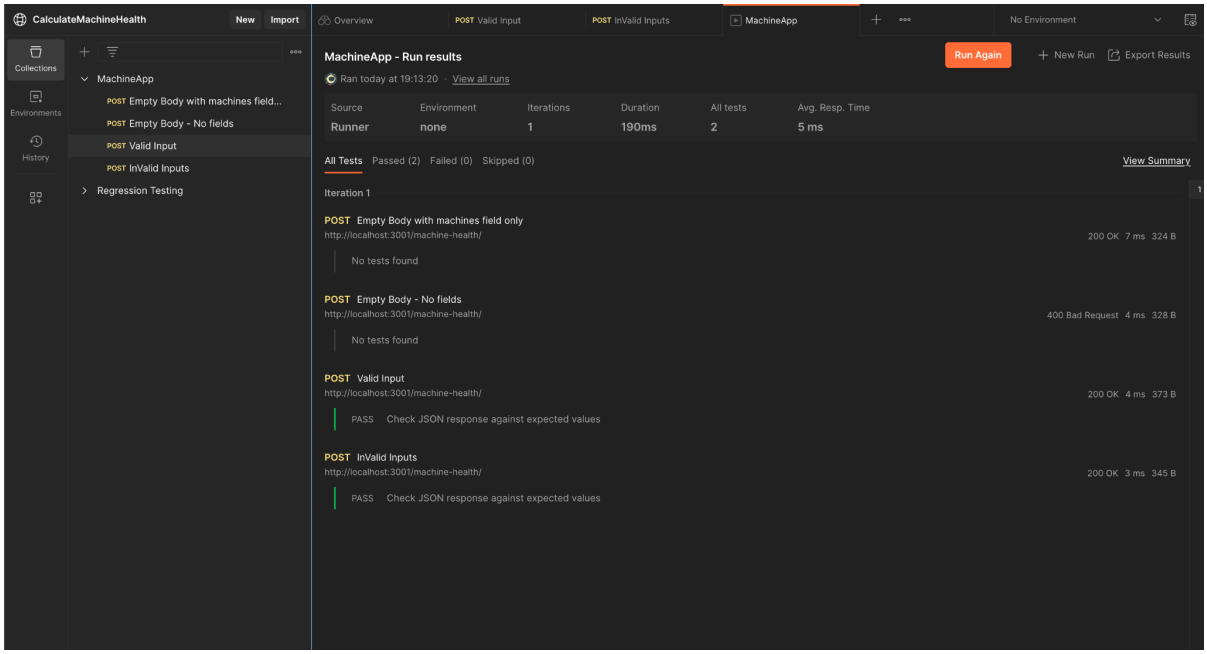
Regression Testing:

- This can be achieved by grouping and prioritizing testcases.
  - Verify the web page opens in all the browsers.
  - Calculate the factory values
  - Calculate the machine health.
  - Verify the API request/response is working.

Challenges/TBD:

- I have not worked with Jest framework and puppeteer before. I have worked with Selenium Webdriver and Cucumber(BDD) and have used Java as the programming language in implementing various automation scripts. For the API testing i have used POSTMAN. I have created freelance projects using JavaScript. So, that helped and motivated to take up this challenge.
- After I set up the project “QAEngineeringChallenge” in my local machine, i had to resolve the CORS policy issue that was getting thrown as the server side application was not allowing the host “http://localhost:8081”.
  - I resolved it by including a CORS middle ware code in backend/node_nodules/app.ts
  -
```
//Included  CORS Middleware to  allow origin (http://localhost:8081)
const cors = require('cors');// Import the "cors" middleware
app.use(cors());
```

- I learnt about snapshot testing and how to create snapshots. I created 3 snapshots successfully.
- For the backend logic, I used POSTMAN to create a collection of requests to test the APIs with valid and invalid sets of data. I have attached the collection with the project folder “MachineApp.postman_collection” for reference. To run the collection install postman and import this collection file —-->select Run collection. And the result will be displayed as below,


CalculateMachineHealth
New
Import
Collections
Environments
History
MachineApp
POST Empty Body with machines field...
POST Empty Body - No fields
POST Valid Input
POST InValid Inputs
Regression Testing
Overview
POST Valid Input
POST InValid Inputs
MachineApp
No Environment
MachineApp - Run results
Run Again
New Run
Export Results
Ran today at 19:13:20 · View all runs
Source
Runner
Environment
none
Iterations
1
Duration
190ms
All tests
2
Avg. Resp. Time
5 ms
All Tests
Passed (2)
Failed (0)
Skipped (0)
View Summary
Iteration 1
POST Empty Body with machines field only
http://localhost:3001/machine-health/
200 OK 7 ms 324 B
No tests found
POST Empty Body - No fields
http://localhost:3001/machine-health/
400 Bad Request 4 ms 328 B
No tests found
POST Valid Input
http://localhost:3001/machine-health/
200 OK 4 ms 373 B
PASS Check JSON response against expected values
POST InValid Inputs
http://localhost:3001/machine-health/
200 OK 3 ms 345 B
PASS Check JSON response against expected values