



TerraByte

Testing Policy

A project for  by  GeekGurusUnion
Demo 4

30 September 2024

Table of Contents

Testing Automation.....	3
Testing Procedure.....	4
Testing our Quality Requirements.....	4
Test Cases.....	4
Testing Reports.....	5
Code Coverage Report.....	7

Testing Automation

To achieve the goal of having automated tests, we decided to go with the GitHub Action tools. Since our project is present on GitHub, integrating GitHub Actions eased the need for external tools such as Travis CI.

Justification of testing tools

1. Presentation Layer Testing (Frontend Testing)
 - a. For testing the frontend we have decided to go with Vitest. Since we can specify the testing environment in Vitest to Nuxt, which is the framework that we used for our application, it eases the need to learn a new testing framework for the project. It is also very suitable for writing unit tests for our applications.
2. Logic Layer Testing (Backend Testing):
 - a. Majority of the backend for our application is written in Python which makes writing our unit tests in Pytest easier due to the easy mocking capabilities as well as familiar syntax and an abundance of libraries that can aid in the production of good unit tests.
3. Data Layer Testing:
 - a. To test how the interactions between the database and application will perform under different circumstances we have used Postman and Vitest. Since our API is a big part of our system testing it is crucial. Postman allowed us to not only test our API thoroughly by allowing us to simulate plenty of API calls, but it also allowed us to test under different environments such as development, testing, and deployment environments.
 - b. Vitest was used to test how the data reacts with parts of our application and test to see if the correct parts of our application responds when certain data is passed in.
4. End-to-End Testing:
 - a. Cypress was our framework of choice for end-to-end testing due its fast and reliable performance. Its straightforward setup and configuration streamline test writing and execution. Key features like real-time reloading and in-browser debugging enhance the developer experience. Additionally, its headless execution capabilities ensure smooth CI/CD integration, improving development efficiency and delivering higher quality outcomes.

Testing Procedure

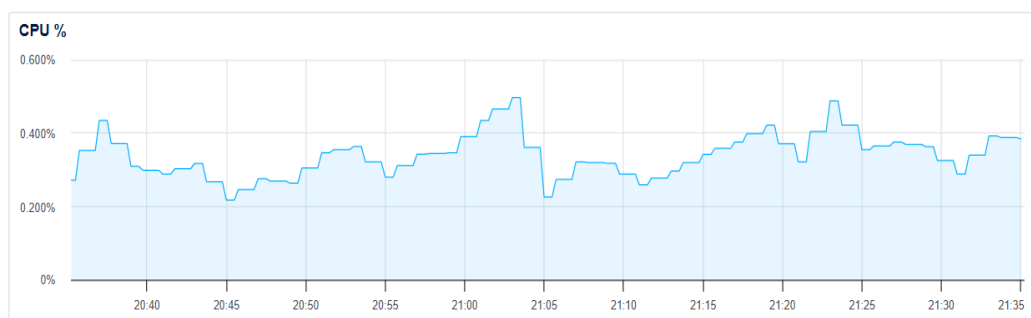
For testing we decided to go with an agile testing methodology. This helps speed up our application development by periodically writing tests on all the components and procedures of our application to ensure that we are staying on track and so that no surprises are encountered when developing at a later stage, as such surprises could slow down our development at a later stage.

Testing our Quality Requirements

- Testability:
 - Using the Process and tools mentioned above we ensure that our application remains well tested to have a clear distinction on areas that need major improvement.
- Efficiency:
 - To test the efficiency of our application and software we have used the graphs provided by our Digital Ocean which provides us.

Test Cases

- Efficiency



- As shown in the graph above, we have a specified percentage of CPU usage that gives us a good indication of how efficient the processes are with them having an average of 38% CPU usage.
- Usability
 - We have given the application to some of our colleagues and family members to give feedback on how usable the application is.
- Testing for Frontend:

- GitHub Repository:
<https://github.com/COS301-SE-2024/Crop-Prediction-System/tree/main/frontend/tests>
- End-to-End Testing:
 - GitHub Repository:
<https://github.com/COS301-SE-2024/Crop-Prediction-System/tree/main/frontend/cypress>
- Unit Testing for Backend:
 - GitHub Repository:
<https://github.com/COS301-SE-2024/Crop-Prediction-System/tree/main/backend/test>

Testing Reports

- Backend Unit Tests

----- coverage: platform linux, python 3.10.12-final-0 -----

Name	Stmts	Miss	Branch	BrPart	Cover
IoT/getData.py	15	0	0	0	100%
app.py	113	14	8	2	83%
database/supabaseFunctions.py	320	98	76	19	67%
database/supabaseInstance.py	14	0	2	0	100%
logic/calculateHectare.py	20	0	2	0	100%
logic/gemini.py	15	3	0	0	80%
logic/weather.py	101	63	20	1	32%
model/FusionModel.py	64	5	18	4	89%
model/ML.py	33	3	0	0	91%
model/MultiScaleModel.py	119	2	74	4	97%
model/StageModel.py	60	2	20	2	95%
model/YieldOnlyModel.py	35	1	0	0	97%
model/pipeline.py	77	17	22	7	74%
sensor/SensorModel.py	57	46	0	0	19%
TOTAL	1043	254	242	39	74%

- Frontend Unit Tests

File	% Stmts	% Branch	% Funcs	% Lines
All files	66.33	61.32	31.49	66.33
components	71.96	69.29	40	71.96
AccountRange.vue	83.67	65.65	0	83.67
CustomRange.vue	100	100	0	100
FieldCard.vue	58.19	41.37	81.81	58.19
FieldStatsPanel.vue	100	75	100	100
GaugeMap.vue	34.51	40	11.11	34.51
GaugeMapField.vue	52.89	77.77	11.11	52.89
LogTwicker.vue	100	100	100	100
MarketDataCard.vue	100	100	100	100
MarketHectareCard.vue	100	100	100	100
MessageTwicker.vue	87.3	87.14	100	87.3
Navbar.vue	79.5	60	0	79.5
NavigationMenu.vue	87.15	0	0	87.15
PieChart.vue	100	100	100	100
Sidebar.vue	100	100	50	100
StatPanel.vue	100	66.66	100	100
StatCard.vue	100	100	100	100
composables	3.83	0	0	3.83
useFieldStatsCard.ts	0	0	0	0
useFieldStats.ts	0	100	0	0
useMarketData.ts	0	0	0	0
middleware	100	100	100	100
auth.ts	100	100	100	100
pages	100	100	100	100
Index.vue	100	100	100	100
pages/Login	100	100	100	100
Index.vue	100	100	100	100
pages/Signup	100	100	100	100
Index.vue	100	100	100	100
pages/accept	100	100	100	100
Index.vue	100	100	100	100
pages/confire	100	100	100	100
Index.vue	100	100	100	100
pages/help	100	100	100	100
Index.vue	100	100	100	100
pages/inputs/add-field	100	100	100	100
Index.vue	100	100	100	100
pages/inputs/manage-fields	100	100	100	100
Index.vue	100	100	100	100

- Integration Tests (Postman)

PUT	updateEntry	5	0
POST	createField	4	0
POST	deleteField	5	0
PUT	updateField	4	0
GET	getTeamFields	5	0
GET	getFieldInfo	5	0
GET	getFieldData	5	0
GET	getTeamFieldData	6	0
GET	getPastYieldAvg	3	0
POST	addToTeam	5	0
PUT	removeFromTeam	4	0
POST	updateRoles	5	0

- End-to-End Tests

