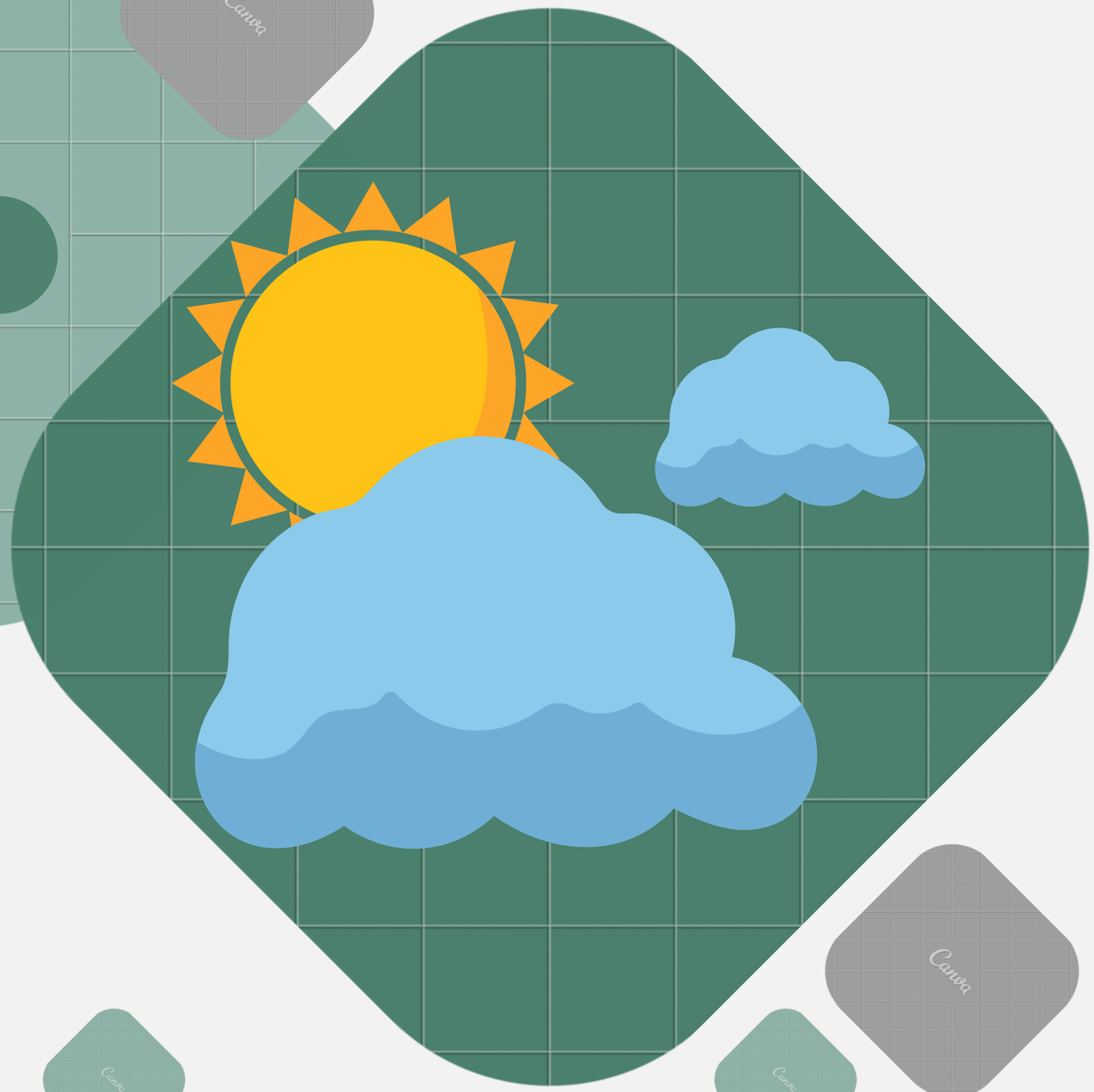


WEATHERSENSE PROJECT

WEATHER SENSE

Presentation, April 2024

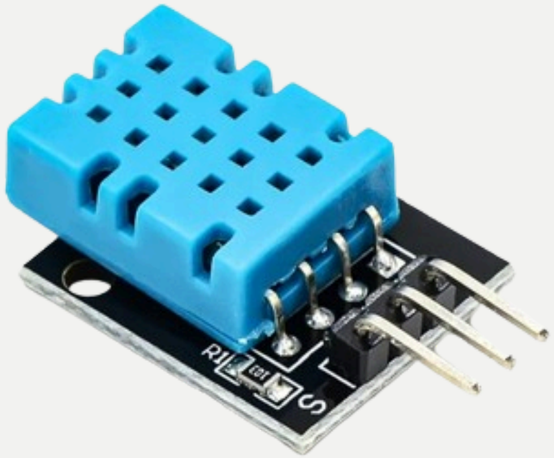
PRESENTATION

About Project

WeatherSense is a weather tracking website that updates weather data and predictions every 10 minutes. Leveraging our trained random forest model, WeatherSense provides accurate forecasts and historical weather information.

HOW IT WORK?

THE SYSTEM COLLECTS DATA USING THE KY-015 SENSOR AND INTEGRATES WITH THE OPENWEATHERMAP API TO ENSURE RELIABILITY. NODE-RED ASSISTS IN DATA CORRECTION.



OpenWeather

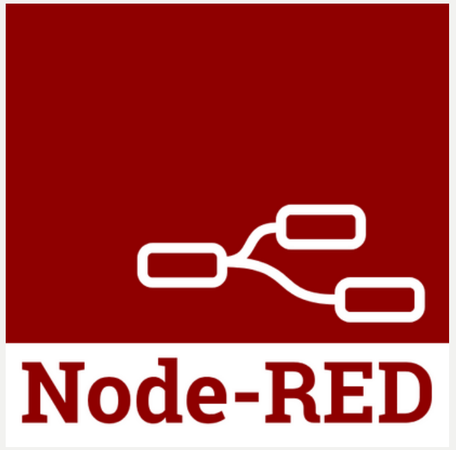
FRONTEND

WE DEVELOP FRONTEND USING SVELTEKIT FRAME WORK AND TAILWINDSCSS

tailwindcss

BACKEND

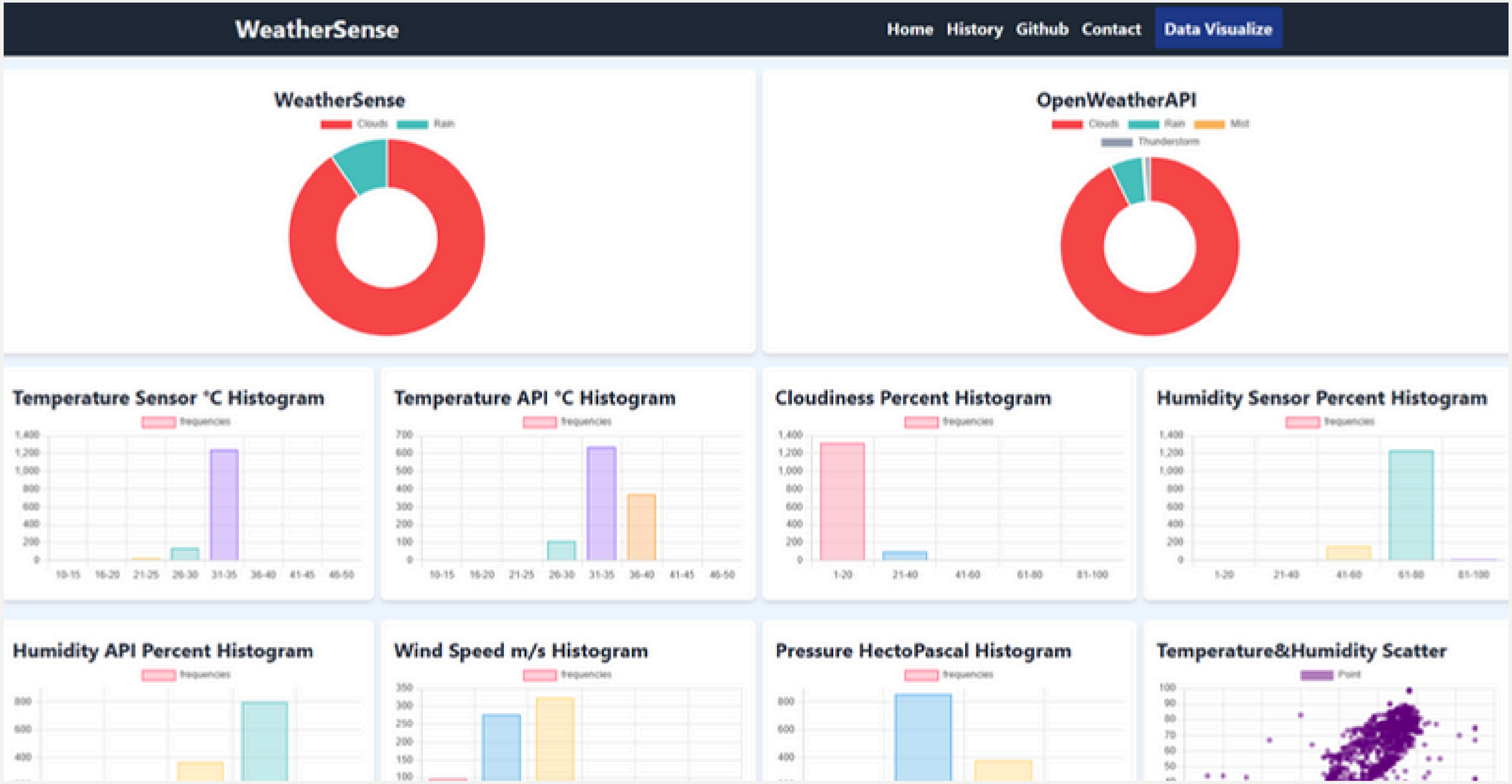
DEVELOP BACKEND BY NODEJS + EXPRESS , NODERED, KU MUSQL SERVER



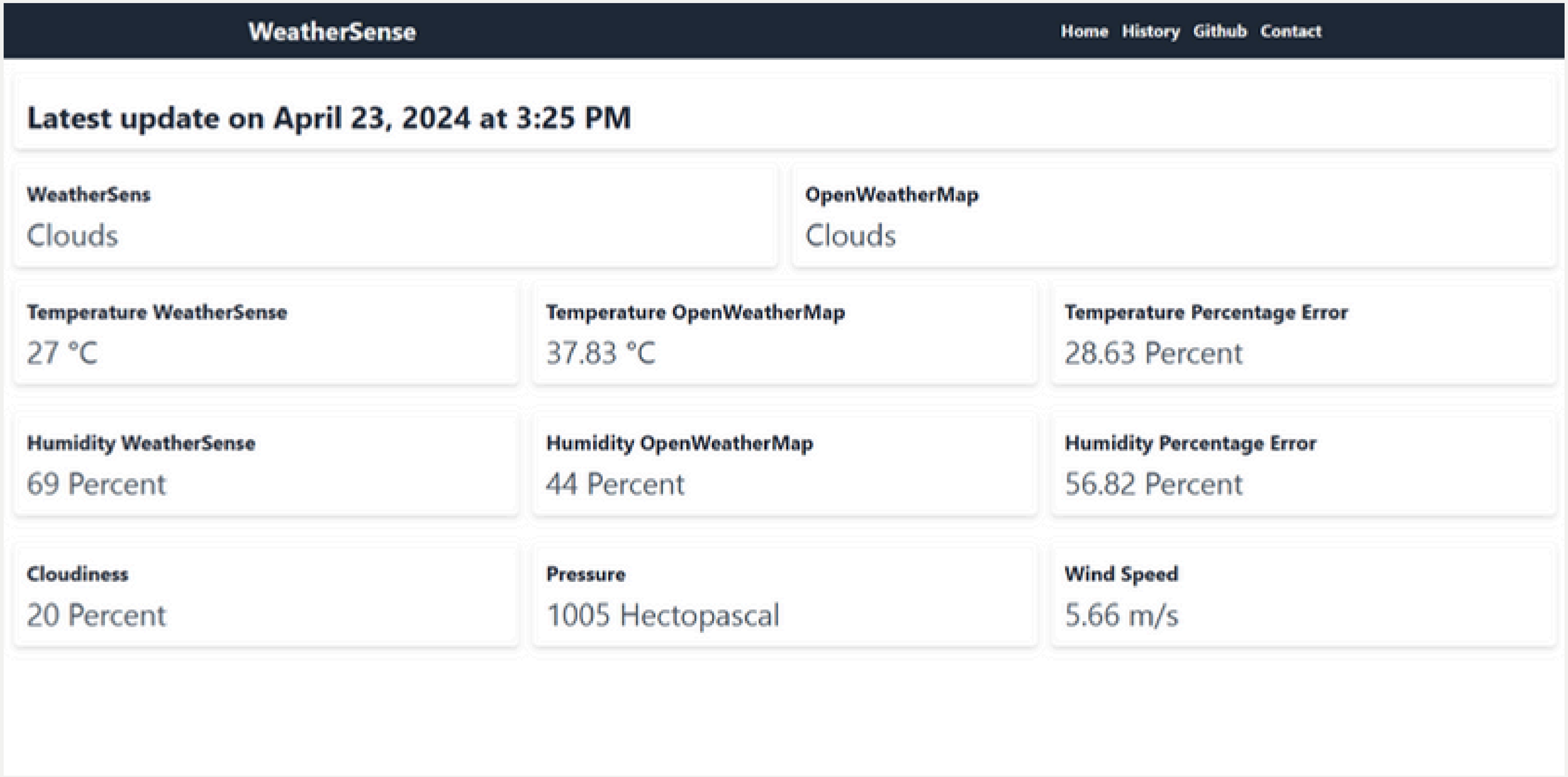
MODEL SERVER

DEVELOPMENT BY FLASK

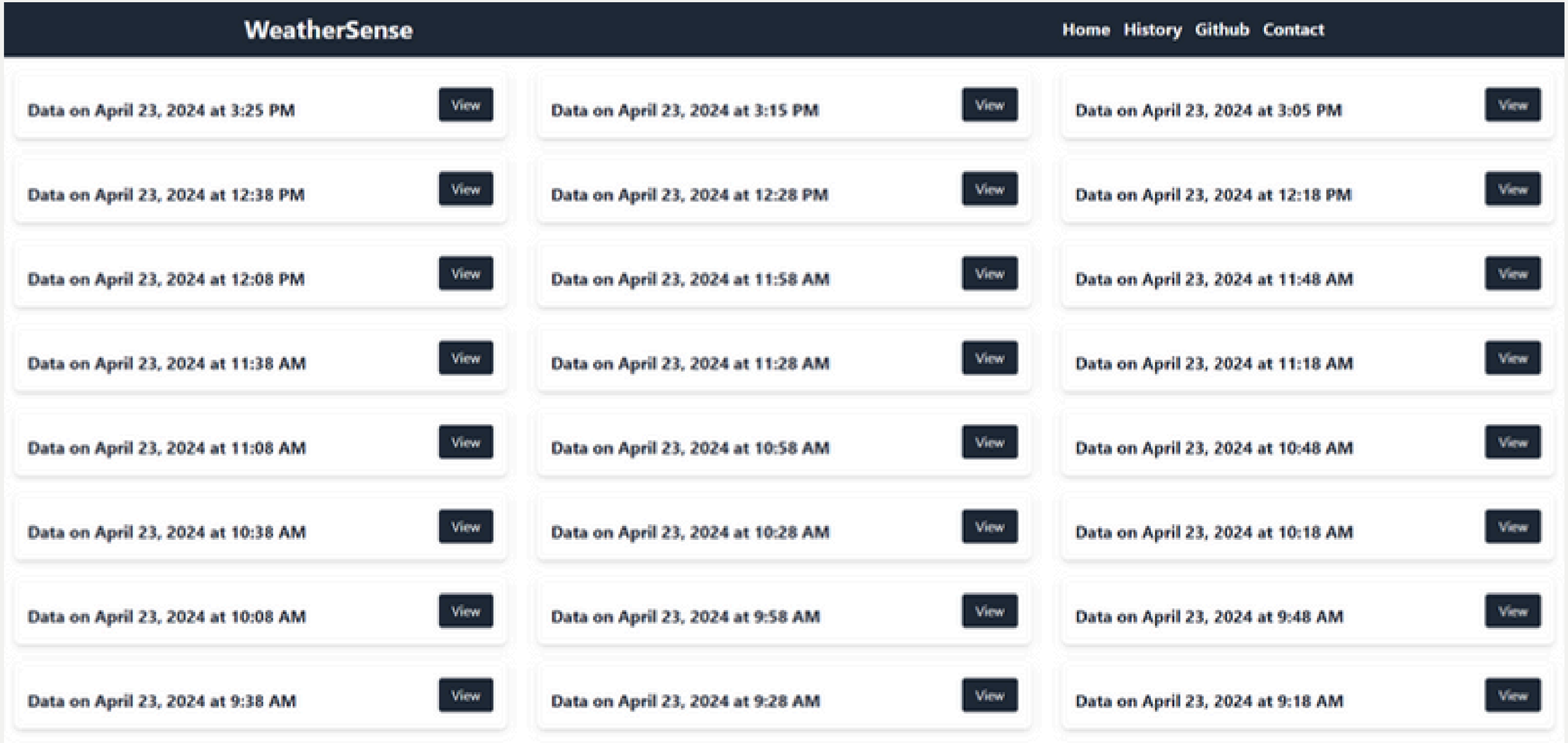
REAL WEBSITE SHOWCASE



REAL WEBSITE SHOWCASE



REAL WEBSITE SHOWCASE



TEST PLAN

TEST PLAN

Objective: As our project comprises three major components, we want to make sure they function independently and integrated together.

Strategy: We will use both manual and automated tools to test our project.

Strategy:
We will use both manual and automated tools to test our project.
Activities:

1. Choose one of the three major components.
2. Choose the part that we want to test on its functionality.
3. Determine the objective of our test on the part.
4. Set a clear starting and ending point for each part that we want to test.

FRONTEND TEST

WE DECIDED TO TEST WITH VITEST, PLAYWRIGHT, POSTMAN

Frontend Unit testing

Unit testing on the frontend is conducted using Vitest and jestdom, which focuses on testing Svelte components to ensure each component functions as expected. Since our website is not complex and has minimal functionality, functional testing is deemed unnecessary.

PRESENTATION

TEST SCENARIO AND SOME TEST CASE

IN FRONTEND UNIT TESTING SHOW CASE

TEST SCENARIO

Test Scenario: Rendering Svelte Components with Props
Description: This test scenario verifies that all Svelte components render correctly with their respective props.
Precondition: 1. The Svelte component is available. 2. Props are correctly defined and accessible within the component.
Step: 1. Instantiate the Svelte component. 2. Verify that each prop is correctly rendered within the component.
Expected Result: 1. The Svelte component renders without error. 2. Each prop is displayed or utilized as expected within the component.

COMPONENT TESTING TESTCASE

CARD COMPONENT

Test Case title: Card should render with title and value
Preconditions: 1. The Card.svelte component is available. 2. Props are correctly defined and accessible within the component.
Test step: 1. Render Card with defined props. 2. Verify Card component.
Expected result: 1. The Card component renders correctly without errors. 2. Both title and value props are displayed correctly.
Test environment: JestDOM
Actual result: 1. The Card component renders correctly without errors. 2. Props are displayed correctly.
Test Scenario: Rendering Svelte Components with Props
Status: Pass

COMPONENT TESTING CODE

CARD COMPONENT

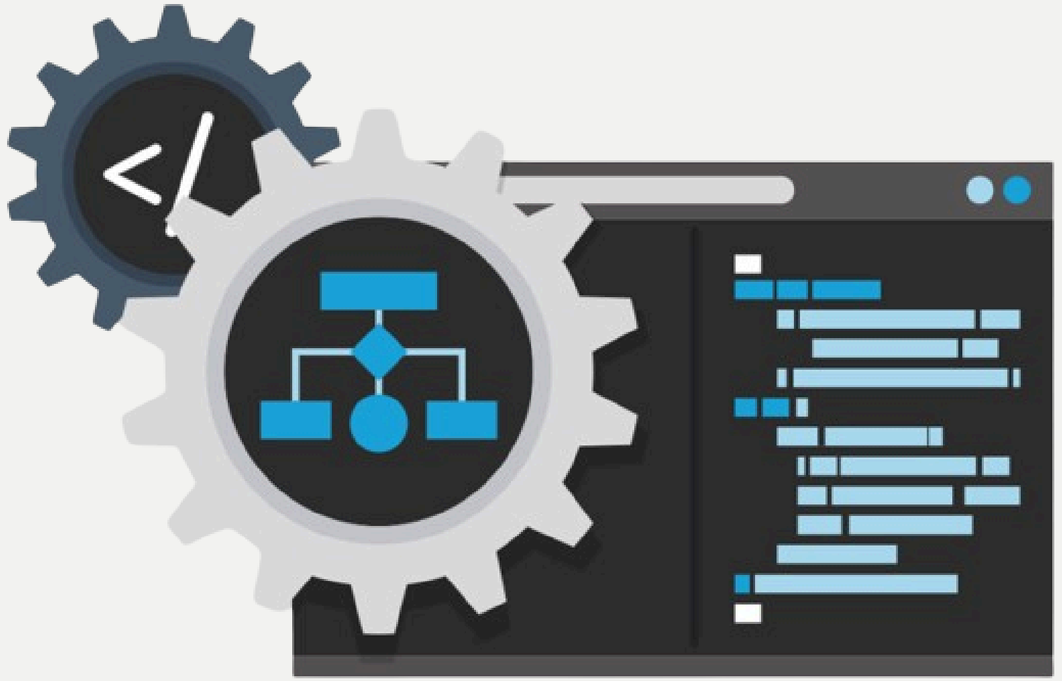
```
describe("Card component", () => {

    beforeEach(() => {
        render(Card, { props: { title: "Test Title", value: "Test Value" } })
    })

    it('should render with title and value', () => {
        expect(screen.getByText("Test Title")).toBeInTheDocument();
        expect(screen.getByText("Test Value")).toBeInTheDocument();
    });
});
```

CHECKING CARD COMPONENT REDENDER SUCESSFULL
ALL PROPS ARE IN THE COMPONENT

Frontend Integration Testing



In integration testing, we utilize Vitest similarly to unit testing, and we also employ Postman for testing. Our focus during testing is on requests to every existing webpage route and checking the responses.

TEST SCENARIO AND SOME TEST CASE

IN FRONTEND INTEGRATION TESTING SHOW CASE

TEST SCENARIO

Test Scenario:
HTTP Request to every web page route

Description:
This test scenario verifies that all Frontend routes can request and get responses correctly as expected.

Precondition: The front-end server must be running.

Step:
1. Request to expect route of webpage.
2. Verify HTTP response.

Expected Result: The response should be 200

INTEGRATION TESTING TESTCASE

/WEATHER/ID (VALID)

Test Case title: GET request to "/weather/id" with valid id
Preconditions: 1.The front-end server is running. 2.Back-end server is running. 3.Define valid id to test
Test step: Go to route "localhost:xxxx/weather/id" Verify response status code
Expected result: status 200
Test Scenario: Request to every web page route
Test environment: HTTP
Actual result: status 200
Status: **Pass**

INTEGRATION TESTING CODE

/WEATHER/ID (VALID)

```
1  test("should allow users go weather page (valid ID)", async () => {
2      const id = 1;
3      const response = await axios.get(`${backendRoutes.getWeatherById}${id}`);
4      expect(response.status).toBe(200);
5  });
```

WHEN USER REQUEST TO THIS ROUTE
RESPONSE SHOULD BE STATUS 200

INTEGRATION TESTING TESTCASE

/WEATHER/ID (INVALID)

Test Case title: GET request to "/weather/id" with invalid id
Preconditions: 1.The front-end server is running. 2.Back-end server is running. 3.Define valid id to test
Test step: Go to route "localhost:xxxx/weather/id" Verify response status code
Expected result: status 404
Test Scenario: Request to every web page route
Test environment: HTTP
Actual result: status 404
Status: Pass

INTEGRATION TESTING EXAMPLE

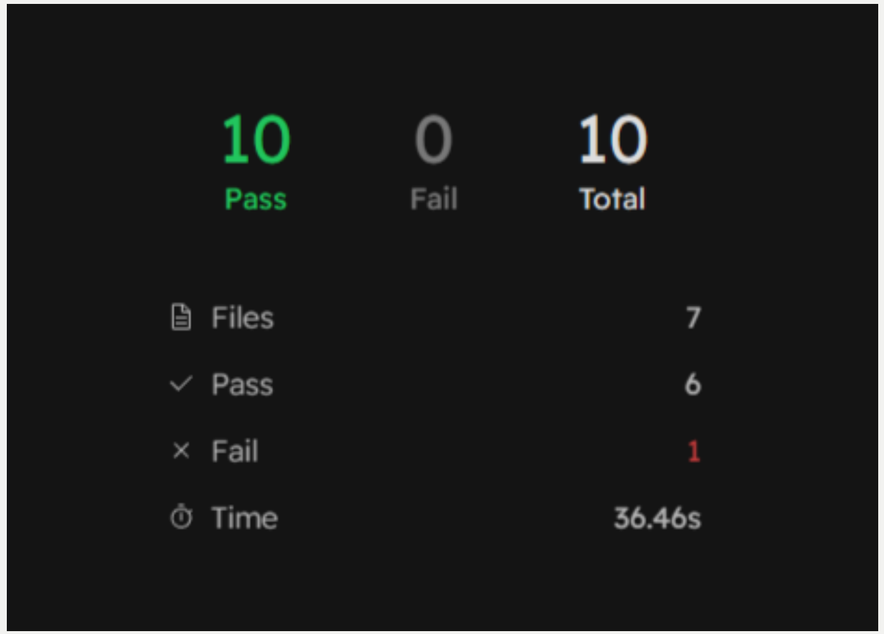
/WEATHER/ID (INVALID)

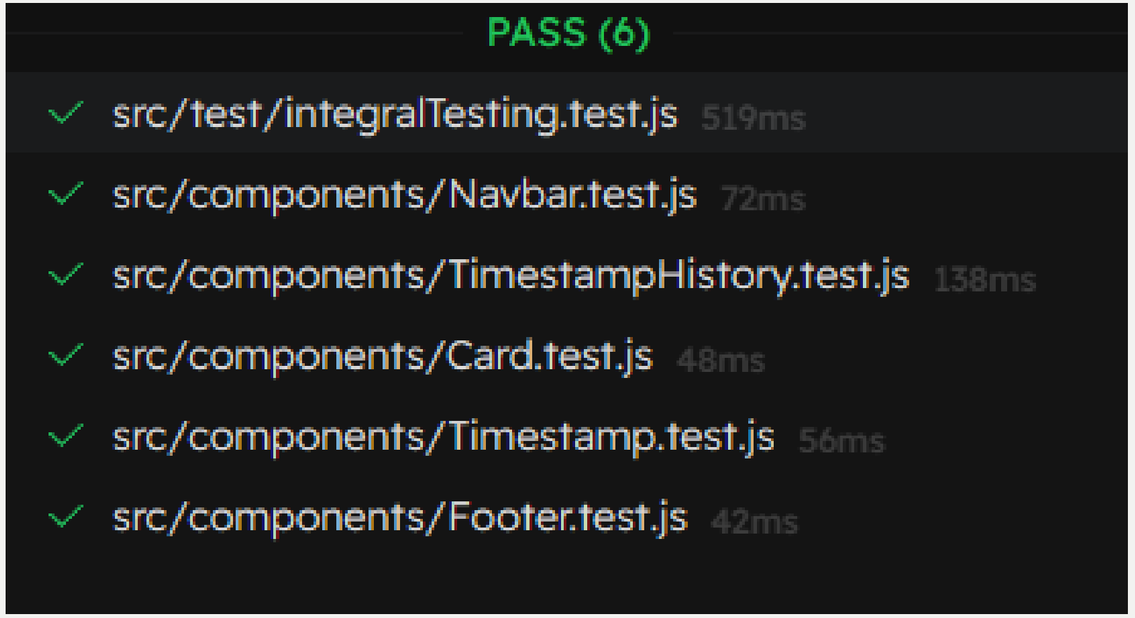
```
1  test("should not allow users to go to history page with invalid ID", async () => {
2      const id = -99;
3      try {
4
5          const response = await axios.get(`${backendRoutes.getWeatherById}${id}`);
6          expect(response.status).not.toBe(200);
7
8      } catch (error) {
9          expect(error.response.status).toBe(404);
10     }
11 });
12
```

WHEN USER REQUEST INVALID ID TO ROUTE
RESPONSE SHOULD BE STATUS 404

VITEST TEST RESULT

npm run test:ui





1 TEST IS FAILED BECAUSE VITEST TRY TO RUN TEST OF PLAYWRIGHT SO IT HAS CONFLICT BUT IT NOT EFFECT TO OTHER

POSTMAN TEST RESULT

GET MainPage

http://localhost:5173/

PASS Response status code is 200

GET HistoryPage

http://localhost:5173/history

PASS Response status code is 200

GET WheaterById

http://localhost:5173/weather/1

PASS Response status code is 200

Frontend E2E Testing

In end-to-end testing, we utilize Playwright to simulate real-world scenarios to ensure that users can interact with our system effectively and that it functions correctly.

TEST SCENARIO AND SOME TEST CASE

IN FRONTEND E2E TESTING SHOW CASE

TEST SCENARIO

Test Scenario:
Verifies that the website functions correctly and as expected from the user's perspective.

Description:
This test scenario verifies that the user uses the web site to work correctly as expected.

Precondition:

1. The frontend server is running.
2. The backend server is running.
3. The ModelAPI server is running.

Step:

1. Open the website in a web browser.
2. Navigate through different pages and functionalities of the website
3. Verify that all interactions and functionalities work as expected.
4. Test the website on different devices and browsers (if necessary).

Expected Result: All pages and features of the website load without errors.

E2ETESTING TESTCASE

USER GO TO MAIN PAGE

Test Case title: In the home page all data Card components are rendered.
Preconditions: 1. All website services are running.
Test step: 1. Go to route “localhost:xxxx/”. 2. Verify the page.
Expected result: All data displayed
Test Scenario: Verifies that the website functions correctly and as expected from the user's perspective.
Test environment: Chromium
Actual result: All data displayed
Status: **Pass**

E2ETESTING EXAMPLE

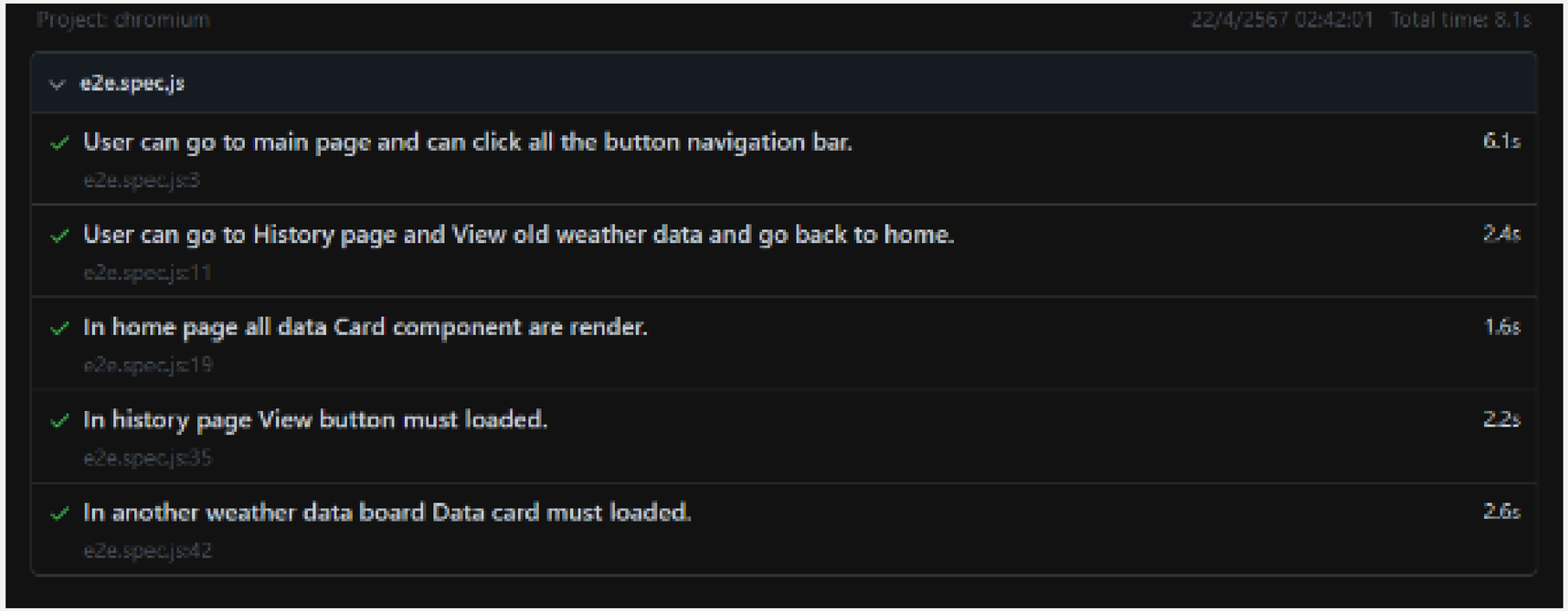
USER GO TO MAIN PAGE

```
test("In home page all data Card component are render.", async ({ page }) => {
  await page.goto('http://localhost:5175/');
  await page.getByRole('navigation').click();
  await page.getByRole('heading', { name: 'WeatherSens', exact: true }).click();
  await page.getByRole('heading', { name: 'OpenWeatherMap', exact: true }).click();
  await page.getByRole('heading', { name: 'Temperature WeatherSense' }).click();
  await page.getByRole('heading', { name: 'Temperature OpenWeatherMap' }).click();
  await page.getByRole('heading', { name: 'Temperature Percentage Error' }).click();
  await page.getByRole('heading', { name: 'Humidity WeatherSense' }).click();
  await page.getByRole('heading', { name: 'Humidity OpenWeatherMap' }).click();
  await page.getByRole('heading', { name: 'Humidity Percentage Error' }).click();
  await page.getByRole('heading', { name: 'Cloudiness' }).click();
  await page.getByRole('heading', { name: 'Pressure' }).click();
  await page.getByRole('heading', { name: 'Wind Speed' }).click();
});
```

ALL COMPONENT MUST BE LOADED

PLAYWRIGHT TEST RESULT

npx playwright test --project=chromium



ALL TEST RESULT PASS

BACKEND SERVER TEST

WE DECIDED TO TEST WITH POSTMAN

POSTMAN

Node Express API testing

API testing on the backend (Node.js Express) server focuses on testing requests to API endpoints and verifying that the responses are correct as expected. This ensures that users who want to utilize our API endpoints receive accurate response data, as communicated to the frontend earlier.

PRESENTATION

TEST SCENARIO

Test Scenario: HTTP Request to Back-end Server
Description: This test scenario verifies that all Back-end API endpoint work and return json correctly
Precondition: 1. Back-end Server is running
Step: 1. HTTP request to expect endpoint. 2. Verify response.
Expected Result: response should return correctly as expected

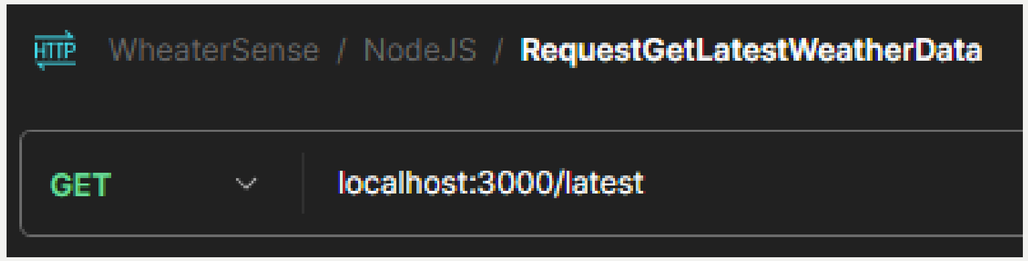
ENDPONT TEST CASE

/LATEST

Test Case title: GET HTTP Request to "/latest"
Preconditions: 1. Back-end Server is running
Test step: 1. HTTP GET request to "localhost:3000/". 2. Verify the response.
Expected result: Response status and json are the latest weather data as expected.
Test Scenario: HTTP Request to Back-end Server
Test environment: HTTP
Actual result: Response status and json are the latest weather data as expected.
Status: Pass

ENDPONT TEST

/LATEST




```
[
    {
        "id": 478,
        "ts": "2024-04-23T08:25:42.000Z",
        "temp_sensor": 27,
        "humidity_sensor": 69,
        "temp_api": 37.83,
        "humidity_api": 44,
        "pressure": 1005,
        "wind_speed": 5.66,
        "cloudiness": 20,
        "weather": "Clouds",
        "weather_pred": "Clouds"
    }
]
```


SHOULD RETURN LASTEST WEATHER DATA

ENDPONT TEST CASE

/HISTORY

Test Case title: GET HTTP Request to "/history"
Preconditions: 1. Back-end Server is running
Test step: 1. HTTP GET request to "localhost:3000/history". 2. Verify the response.
Expected result: Response status and json are all the weather data ordered by DESC.
Test Scenario: HTTP Request to Back-end Server
Test environment: HTTP
Actual result: Response status and json are all the weather data ordered by DESC.
Status: **Pass**

ENDPONT TEST

/HISTORY


```
{
    "id": 478,
    "ts": "2024-04-23T08:25:42.000Z",
    "temp_sensor": 27,
    "humidity_sensor": 69,
    "temp_api": 37.83,
    "humidity_api": 44,
    "pressure": 1005,
    "wind_speed": 5.66,
    "cloudiness": 20,
    "weather": "Clouds",
    "weather_pred": "Clouds"
},
{
    "id": 477,
    "ts": "2024-04-23T08:15:44.000Z",
```


SHOULD RETURN ALL ORDER DESCEND

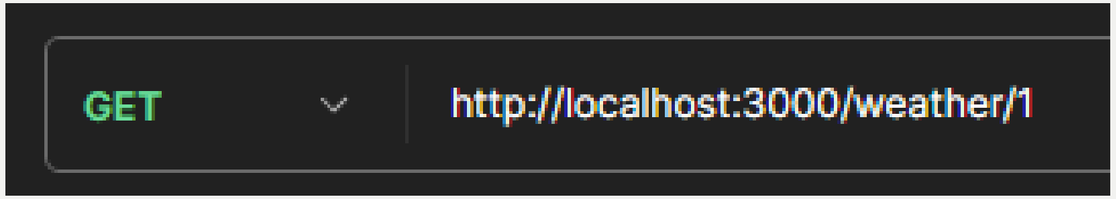
ENDPONT TEST CASE

/WEATHER/ID

Test Case title: HTTP GET Request to "/weather/id"
Preconditions: 1. Back-end Server is running 2. defined id
Test step: 3. HTTP GET request to "localhost:3000/weather/id". 4. Verify the response.
Expected result: Response status and json are the weather data specific id.
Test Scenario: HTTP Request to Back-end Server
Test environment: HTTP
Actual result: Response status and json are the weather data specific id.
Status: **Pass**

ENDPONT TEST

/WEATHER/ID




```
{
    "id": 1,
    "ts": "2024-04-18T10:04:21.000Z",
    "temp_sensor": 20,
    "humidity_sensor": 50,
    "temp_api": 36.88,
    "humidity_api": 57,
    "pressure": 1004,
    "wind_speed": 5.66,
    "cloudiness": 20,
    "weather": "Clouds",
    "weather_pred": "Clouds"
}
```


SHOULD RETURN WEATHER DATA GIVEN ID

POSTMAN APITEST RESULT

GET RequestGetLatestWeatherData
localhost:3000/latest

- PASS Response status code is 200
- PASS Verify that the 'id' is a non-negative integer
- PASS Verify that 'ts' is a non-empty string
- PASS Verify that the 'temp_sensor' is a non-negative number

GET RequestWeatherDataByID
http://localhost:3000/weather/1

- PASS Response status code is 200
- PASS Validate that the 'id' is a non-negative integer
- PASS Validate that the temperature sensor reading is a non-negative number
- PASS Validate that the humidity sensor reading is a non-negative number

GET History
localhost:3000/history

- PASS Response status code is 200
- PASS Response is an array with at least one element
- PASS Verify that the 'id' field is a non-negative integer

ALL TEST AND TEST CASE RESULT PASS

ModelAPI Server (Flask)

API testing in this context ensures that users who wish to utilize our model can send a POST request to our API, and our backend receives the correct response containing the predicted data.

PRESENTATION

ENDPONT TEST SCENARIO

Test Scenario: HTTP Request to ModelAPI-Server
Description: This test scenario verifies that ModelAPI-ServerI endpoint work and return json correctly
Precondition: 1. ModelAPI-Server is running. 2. defined a data to send to predict a model.
Step: 1. HTTP request to expect endpoint. 2. Verify response.
Expected Result: response should return correctly as expected

/PREDICT TESTCASE

/PREDICT

Test Case title: HTTP POST Request to "/predict"
Preconditions: 1. ModelAPI-Server is running 2. defined a data to send to predict a model.
Test step: 5. HTTP GET request to "localhost:5000/predict". 6. Verify the response.
Expected result: Response status and json are the same data but have new attribute name "weather_pred" and its value.
Test Scenario: HTTP Request to ModelAPI-Server
Test environment: HTTP
Actual result: Response status and json are the same data but have new attribute name "weather_pred" and its value.
Status: **Pass**

/PREDICT TESTCASE

/PREDICT

BODY

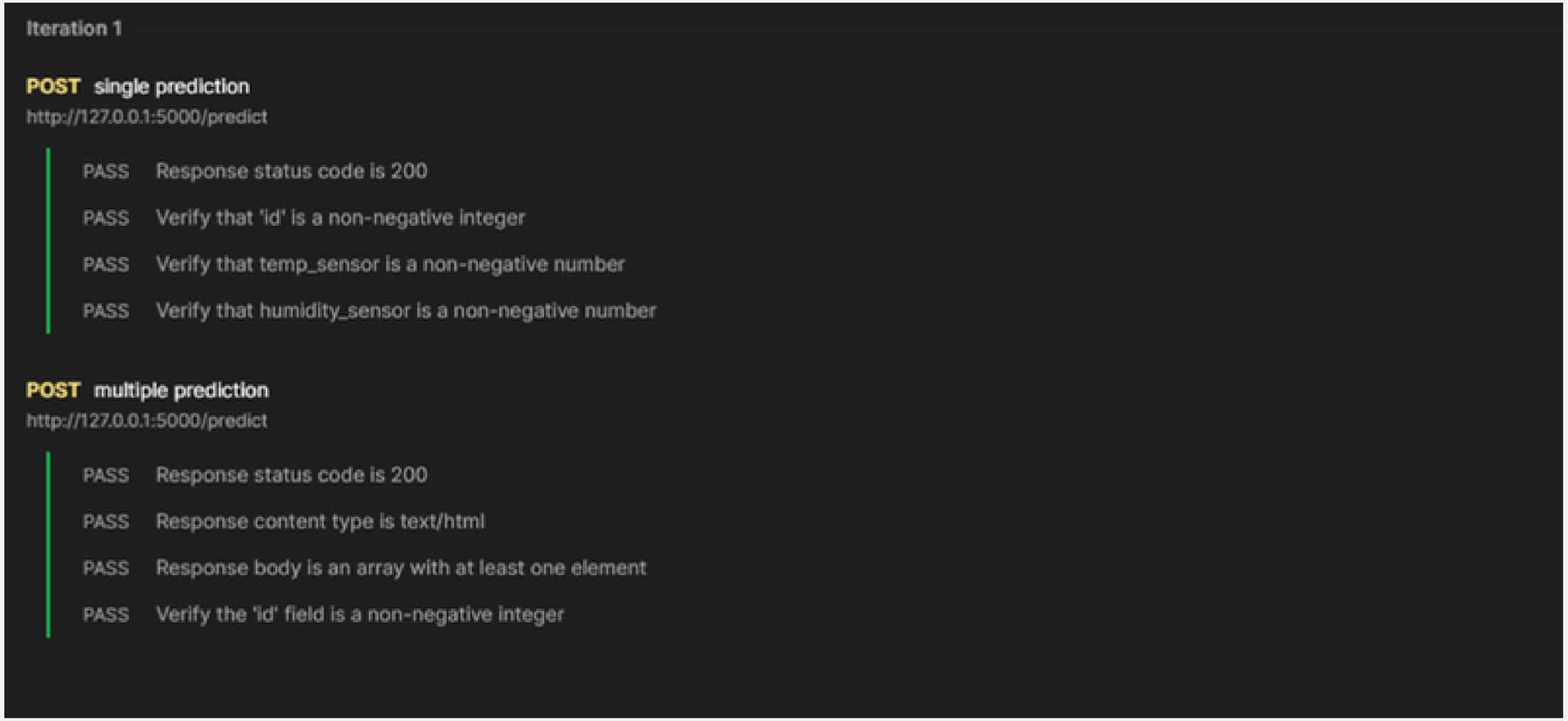
```
[
    {
        "id": 1,
        "ts": "2024-04-14 21:51:13",
        "temp_sensor": 25,
        "humidity_sensor": 50,
        "temp_api": 31.03,
        "humidity_api": 73,
        "pressure": 1010,
        "wind_speed": 5.66,
        "cloundiness": 20,
        "weather": "few clouds"
    }
]
```

RESPONSE

```
[
    {
        "id": 1,
        "ts": "2024-04-14 21:51:13",
        "temp_sensor": 25,
        "humidity_sensor": 50,
        "temp_api": 31.03,
        "humidity_api": 73,
        "pressure": 1010,
        "wind_speed": 5.66,
        "cloundiness": 20,
        "weather": "few clouds",
        "weather_pred": "Clouds"
    }
]
```

RESPONSE SHOULD BE EXPECTED

POSTMAN APITEST RESULT



ALL TEST AND TEST CASE RESULT PASS

GITHUB

YOU CAN SEE MORE

HTTPS://GITHUB.COM/QOSANGLESZ/WEATHERSENSE

PROJECT MEMBER

1. WISSARUT KANASUB

2. SUKPRACHOKE LEELAPISUTH