Development and Testing of EcoWeather Web App

AfzoodNaaz Surati - GH1028962

B215R Software Testing – Reassessment

1/8/26

GitHub Repository: https://github.com/afzoodnaaz/EcoWeather-WebApp.git

Video Demos: https://gismauniversity-my.sharepoint.com/:f:/g/personal/afzood_surati_gisma-student_com/IgAQJEMnbUAVQ6FtKURS83h-AQSRGRNszwj7xO-neEkaOhI?e=w1HqOC

Table of Contents

Introduction 2
Project objectives 2
Application scope 3
Application Overview 4
User stories and Acceptance criteria 6
Strategic test plan 7
  Test objectives 7
  Test scope 7
  Test techniques 8
  Testing types 9
  Tools and framework 9
Automated testing 9
  Testing library selection 9
  Automated test case 9
Static testing 11
  Static analysis tools used 11
  Static testing findings and resolutions 12
Bug report - Defect log 14
Test coverage 15
  Coverage results 16
  Covered analysis 16
References 17

Introduction

In today's world, applications that are web-based highly influence the way in which users' access and interact with all types of software systems. Web-based applications are often expected to be user-friendly, easily accessible, and quick to recover from errors. Therefore, successful software development practices focus on implementation, planning, and quality assurance, with emphasis on testing. The EcoWeather Web Application project manages these expectations by portraying efficiency through its design, development, and testing.

The EcoWeather App is a single page web application which is highlighted as a project in my online academic portfolio as an addition to my previous work. This project was originally designed to be a desktop-based Java Swing application that operate as a conservational tool for the management of reptiles, more specifically crocodiles by using third party OpenMeteo APIs. EcoWeather has now evolved to be a web-based application intended to provide real-time weather data for user-specified cities and crocodile habitats across the world.

The main web application, my online Academic portfolio, comprises of 8 navigable sections: About, Experience, Education, Skills, Interests, Resume, Projects (highlighting the EcoWeather Web App), and Contact. The EcoWeather App comprises of two sections within the single scrollable webpage, The City weather section requires the user to manually input any city name to receive its corresponding weather data, and the Habitat weather section provides the users with a drop-down list of researched and predefined crocodile habitats around the world to choose from in order to receive its corresponding weather information. This dual functionality separates the EcoWeather App from a conventional weather app and supports a domain specific case focused on environmental habitats as well.

The key motivation for this project was to utilise the core functionalities and error-handling logic from a Java-based desktop application and implement it into a modern web-based application. This transition is meant to showcase software evolution by highlighting the migration of applications from desktop environments to browser platforms with intentions to improve deployment flexibility.

This report outlines the design and implementation aspects of the EcoWeather App project with emphasis on the software testing practices applied throughout the developmental stages of this project. A test plan for this project was defined to consistently ensure the project objectives corresponding to multiple user stories were successfully met. The main automated testing tool used to perform software testing on this project is Cypress. Cypress was preferred as the right tool for this project as it enables an almost realistic simulation of any user interactions in a web application environment, especially regarding the front end, and due to its ability to test form submissions, drop-down list selection, and user interface updates.

The development and testing of this project put forth an application that is doesn't only serve as a conservational tool, but also demonstrates recognised software engineering principles.

Project objectives

The EcoWeather Web Application is developed with the objective of showcasing practical software engineering practices with the limitations of a fairly small-scale web application. This task doesn't solely focus on design and implementation of the project but also integrates structured testing processes commonly used in software development.

The primary goals of this project are listed below:

- To design and implement a web application while utilising modern web technologies such as HTML, CSS, and JavaScript, and emphasizing user-side knowledge and API integration.
- To implement at least 10 user stories, making sure the functional requirements are clearly defined and met by a user-centred standard and can be validated through software testing.
- To successfully perform API integration for real-world and real-time data by handling network requests, JSON parsing, and exception handling of errors and invalid data.
- To apply software testing and quality assurance practices into the core developmental stage, including automated testing using Cypress and manual testing.
- To produce a comprehensive and cohesive report including project specifications, project life cycle, and technical testing documentation of design strategies, testing strategies, and implementation.

Application scope

The scope of the project EcoWeather Web Application is defined to meet all the technical requirements of a small-scale web-based application and highlighting its functionality, usability, and complexity.

Category	In-scope	Out of scope
Application type	User-side web application	Desktop application.
Architecture	Single page application with redirection to other webpages	Multi-tier back-end architecture
Weather data	Current weather data includes temperature, humidity, and wind speed	Weather forecasting beyond given parameters
Data source	public third-party weather and geolocation API	Custom API usage
City weather search	User-entered city weather search	List or batch city search and data storage
Habitat weather search	Predefined drop-down crocodilian habitat weather search.	User designed or geolocation defined habitat creation
Input validation	Empty input detection and invalid input handling	Advanced validation framework
Error handling	API connection failure handling and error message	Automated error recovery
Data storage	None	Database or user-based storage integration
User accounts	None	User profile management
Testing focus	Frontend, end to end, and integration	Load and performance testing
Accessibility	Basic usability	Full compliance testing
Deployment	Static hosting (GitHub, desktop)	Cloud-based deployment

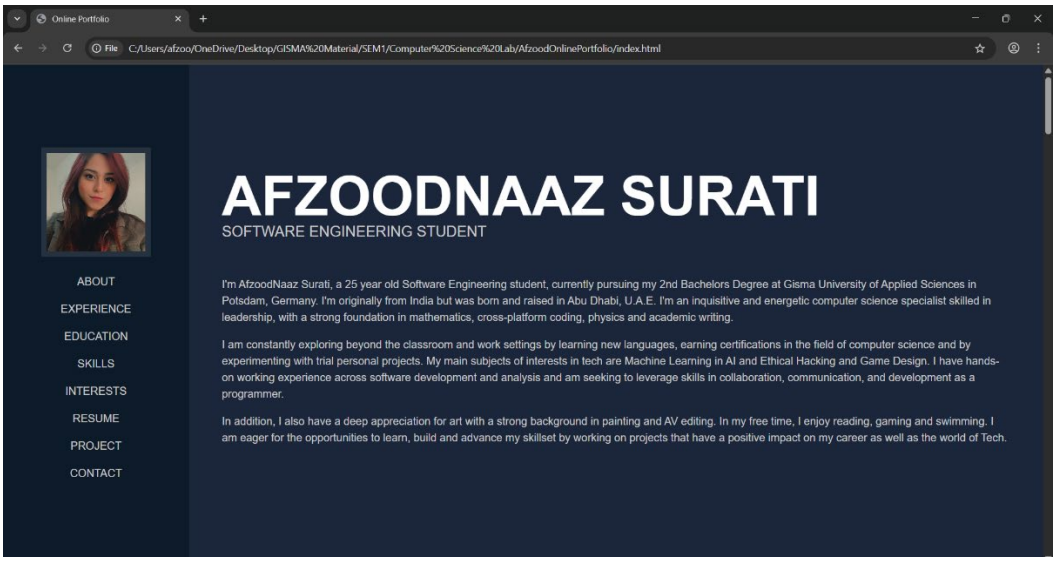
Application Overview

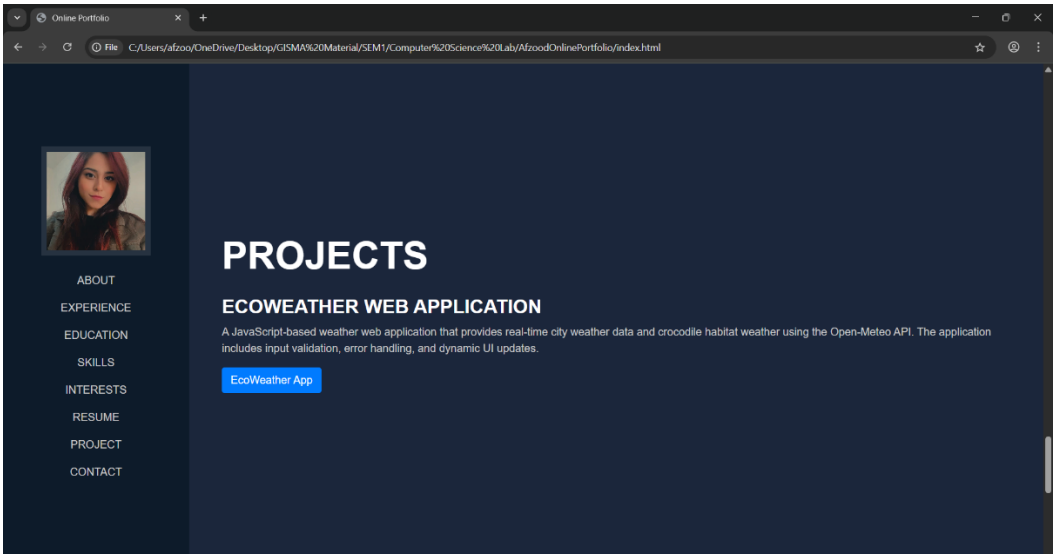
The EcoWeather Web App is a single page web-based application tended to provide real time weather data through a responsive web user interface. The EcoWeather web page is highlighted in the project section of my online academic portfolio. The application has 2 main functionality sections: a City based weather search and a predefined drop-down list Crocodile Habitat based weather search. Both these sections are accessible on the same single page and allow users to simultaneously perform multiple searches while these sections display retrieved weather data independent from each other without the application needing to be reloaded.

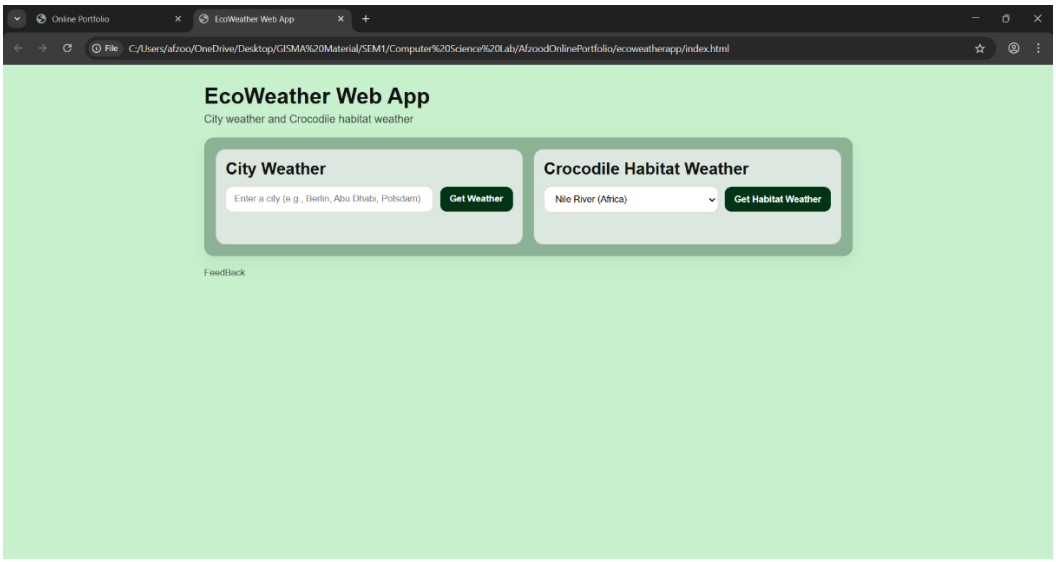
In the city's weather section, users are intended to input the name of a city into an allocated text box input field. Weather data is retrieved through geographic coordinates using a geocode service API & pulling a weather API once the button labeled “Get Weather” is interacted with. The weather data include the temperature, humidity, wind speed, and geolocation coordinates, and are displayed in an athletically pleasing and clean manner.

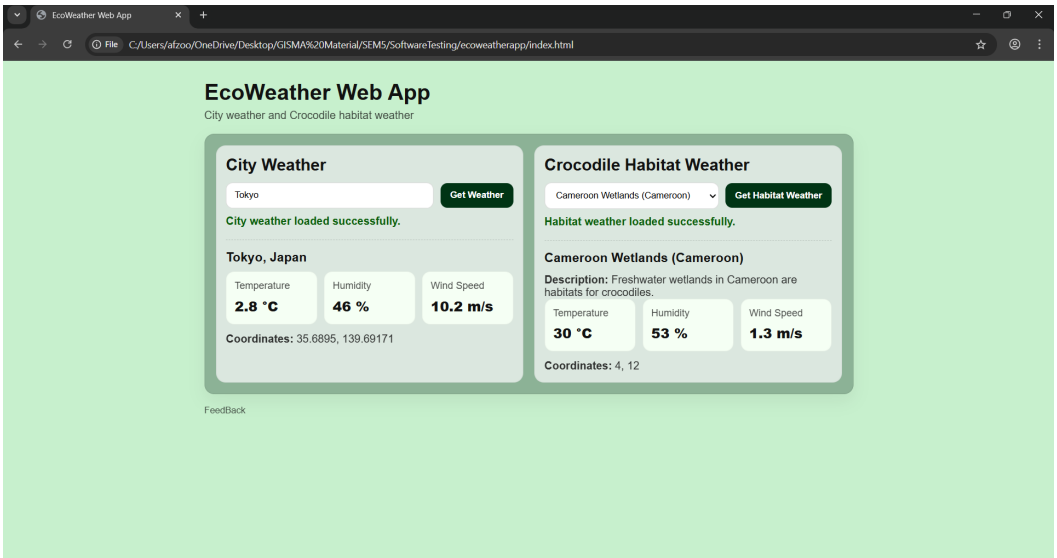
The crocodile habitat weather section provides users with a drop-down list of predefined crocodile habitats from all over the world. Once the selection is made interaction with a button labeled “Get Weather” fetches its corresponding weather data along with a short description of the user-selected habitat. This feature is inspired by the habitat-based logic implemented in my original Java Swing desktop application.

The academic portfolio and the EcoWeather Web App highlighted was developed using llightweight programming languages. The web page is designed using HTML For its core structure as HTML provides A semantic markup and improves maintainability of the project. The HTML elements were mainly used to define sections form inputs and layout boxes for result containers within the application. The styling and grid layouts were handled using CSS to create a responsive design with visual clarity and then aesthetic theme. CSS helped create a clean layout with consistent spacing and adaptive grid structures that ensured usability across any screen sizes. The use of JavaScript in this project was necessary to handle user input functionalities, validating form submissions managing API requests and improving the code readability and effective operations.




Online Portfolio
File C:/Users/afzoo/OneDrive/Desktop/GISMA%20Material/SEM1/Computer%20Science%20Lab/AfzoodOnlinePortfolio/index.html
ABOUT
EXPERIENCE
EDUCATION
SKILLS
INTERESTS
RESUME
PROJECT
CONTACT
PROJECTS
ECOWEATHER WEB APPLICATION
A JavaScript-based weather web application that provides real-time city weather data and crocodile habitat weather using the Open-Meteo API. The application includes input validation, error handling, and dynamic UI updates.
EcoWeather App


Online Portfolio
EcoWeather Web App
File C:/Users/afzoo/OneDrive/Desktop/GISMA%20Material/SEM1/Computer%20Science%20Lab/AfzoodOnlinePortfolio/ecoweatherapp/index.html
EcoWeather Web App
City weather and Crocodile habitat weather
City Weather
Enter a city (e.g., Berlin, Abu Dhabi, Potsdam)
Get Weather
Crocodile Habitat Weather
Nile River (Africa)
Get Habitat Weather
FeedBack


EcoWeather Web App
File C:/Users/afzoo/OneDrive/Desktop/GISMA%20Material/SEM5/SoftwareTesting/ecoweatherapp/index.html
EcoWeather Web App
City weather and Crocodile habitat weather
City Weather
Tokyo
Get Weather
City weather loaded successfully.
Tokyo, Japan
Temperature
2.8 °C
Humidity
46 %
Wind Speed
10.2 m/s
Coordinates: 35.6895, 139.69171
Crocodile Habitat Weather
Cameroon Wetlands (Cameroon)
Get Habitat Weather
Habitat weather loaded successfully.
Cameroon Wetlands (Cameroon)
Description: Freshwater wetlands in Cameroon are habitats for crocodiles.
Temperature
30 °C
Humidity
53 %
Wind Speed
1.3 m/s
Coordinates: 4, 12
FeedBack

User stories and Acceptance criteria

For the echo weather app, 10 user stories were adopted to define the application's functional requirements. Each user story has its corresponding acceptance criteria for this project to reach its primary objectives.

i. User story 1: As a user, I want to access the academic portfolio of a software engineering student for future job prospectives.
Acceptance criteria: The online portfolio application needs to load successfully in a web browser. Once the EcoWeather app button is pressed, the web page must redirect and open to the echo weather app web page without any errors.

ii. User story 2: As a user, I want to access the eco weather web application so I can check whether information with ease.
Acceptance criteria: The eagle weather application web page must load successfully in any web browser without any errors.

iii. User story 3: As a user, I want to successfully enter a city name into the textbox to retrieve current weather data for a specific city.
Acceptance criteria: when a valid city name is entered and submitted current weather is retrieved through the API connections and correctly displayed on the page.

iv. User story 4: As a user, I want to view the current weather conditions including the temperature, humidity, and wind speed of a specific city.
Acceptance criteria: The displayed weather information must include temperature, humidity, and wind speed.

v. User story 5: As a user, I want to receive a warning message if I submit an empty search to avoid invalid requests.
Acceptance criteria: When the city's input field is empty a warning message must be displayed to the user and no API network call is made.

vi. User story 6: As a user, I want to see an error message for an invalid city name input if a city is not found so I may correct my input.
Acceptance criteria: When an invalid city name is entered by the user and error message indicating this information must be displayed.

vii. User story 7: As a user, I want API connection handling or network errors so that the web page does not crash.
Acceptance criteria: When an API network request fails the application must display a user-friendly error message indicating so.

viii. User story 8: As a user, I want to select a crocodile habitat from the predefined drop-down list so that I may receive its corresponding weather data.
Acceptance criteria: The crocodile habitat dropped down list must consist of all the predefined habitat options.

ix. User story 9: As a user I want to view weather information for the selected habitat so I can understand its environmental conditions.
Acceptance criteria: When a habitat is selected by the user whether information for that have that is successfully displayed.

x. User story 10: As a user, I want to view a description alongside weather information of a selected crocodile habitat so I can understand its ecological context.
Acceptance criteria: When a habitat is selected a description corresponding to that habitat must be displayed along with the current weather data.

xi. User story 11: As a user, I want to perform multiple searches on both the city weather and crocodile habitat weather simultaneously without having to refresh the page.
Acceptance criteria: Users must be able to perform multiple searches on both sections simultaneously with the correct retrieval and display of current weather data corresponding to their inputs without reloading the page.

Strategic test plan

This section outlines the structured testing approach used for the EcoWeather Web application. The test plan comprises of the testing objectives, the scope of all the features tested, the testing techniques, the types of testing performed, and the frameworks used for testing. The goal of this structured plan is to ensure that the project goals are met through systematic testing.

Test objectives

The primary testing objective was to verify is its correct behavior of the eco weather web application under normal circumstances and with exposure to error conditions. The software testing of this particular application was aimed at confirming that valid user input consistently results in correct weather data being displayed, while invalid inputs and connection failures were handled through clear error messaging, and confirm timely API connections are being carried out.

Another objective was to ensure that the web application continues to remain stable even during repeated interactions with the user without the need to reload the page, and the user interface updates in real time in response to the user's actions.

Test scope

The scope of the software testing includes verification of all core functionalities of the eco weather application. The core functionalities the testing focuses on are the city-based weather searches, the crocodile habitat-based weather searches, user input validation, API network handling, exception and error handling, user interface updates, and application management. The testing also covers any negative scenarios such as invalid user inputs and empty input fields, and any API network failures.

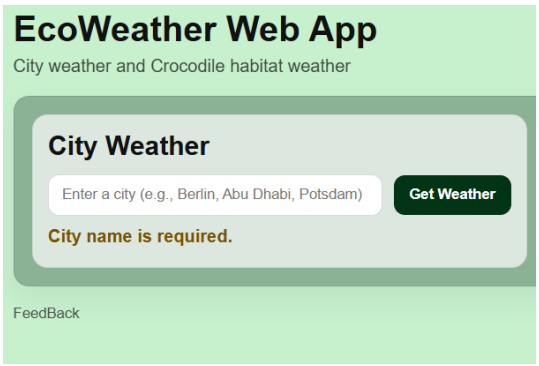
Any third-party API internal behavior and performance testing, as well as cross browser compatibility testing and modern desktop browsers are out of scope of the current test plan.

Test techniques

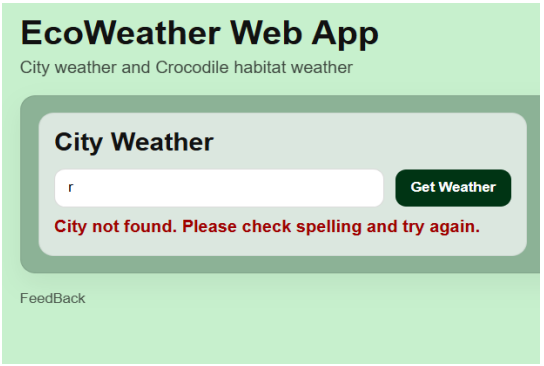
The test techniques applied during testing are listed below:

- Manual testing: Manual testing was used consistently throughout the early and late developmental stages of the EcoWeather Web App. The testing Involved manually predicting user interactions with the application through a browser to verify its user interface behavior, input management, network navigation, and visual appeal. Manual tests were performed for negative scenarios such as entering invalid city names in the city weather search, submitting Mt text boxes, different crocodile habitats from drop down lists with no rest time, and verifying whether information being retrieved and outputted.
- Black-box testing: Black box text testing was used in the software testing of this project tool validate all the systems feature functionalities. Specific test cases were designed based on all 11 user stories and their acceptance criteria while focusing um their inputs and outputs. This approach ensured that the EcoWeather web application met all of its functional requirements and project goals that aligned with the user's expectations.
- Boundary value analysis: this test technique was used to test any edge cases where failures would frequently occur. This includes testing through negative scenarios such as entering empty input fields, attempting to retrieve weather information without selecting habitat from the predefined list, and managing the API returns.

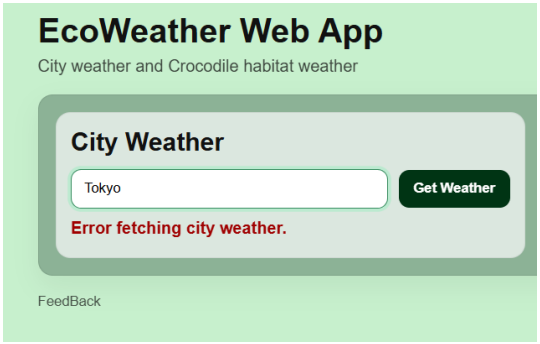
i.e.



(empty input)



(invalid entry)



(failed API connection)

Testing types

The types of testing used for this project are listed below:

- Ene-to-end testing (E2E) is the primary testing technique used throughout the development of EcoWeather App, and was used to verify and validate the complete core feature functionalities as well as the front-end flow using the testing tool Cypress.
- Static testing was also performed to ensure quality assurance practices by focusing on source code examination and file configuration without running the program application. This technique helped manage issues related to code quality and maintainability.

Tools and framework

Several frameworks were used to execute and test the EcoWeather Web application.

- Visual Studio Code: Web app development.
- Windows Shell: Testing tool installation and static testing.
- Prettier: Static testing for code formatting.
- Google Chrome: Web application host.
- Cypress: Automated testing purposes.
- ESLint: Static testing for source code issues.
- GitHub: Repository management and web app host.

Automated testing

Automated testing of the EcoWeather app was done using the Cypress testing tool by focusing on validating feature functionalities and acceptance criteria.

Testing library selection

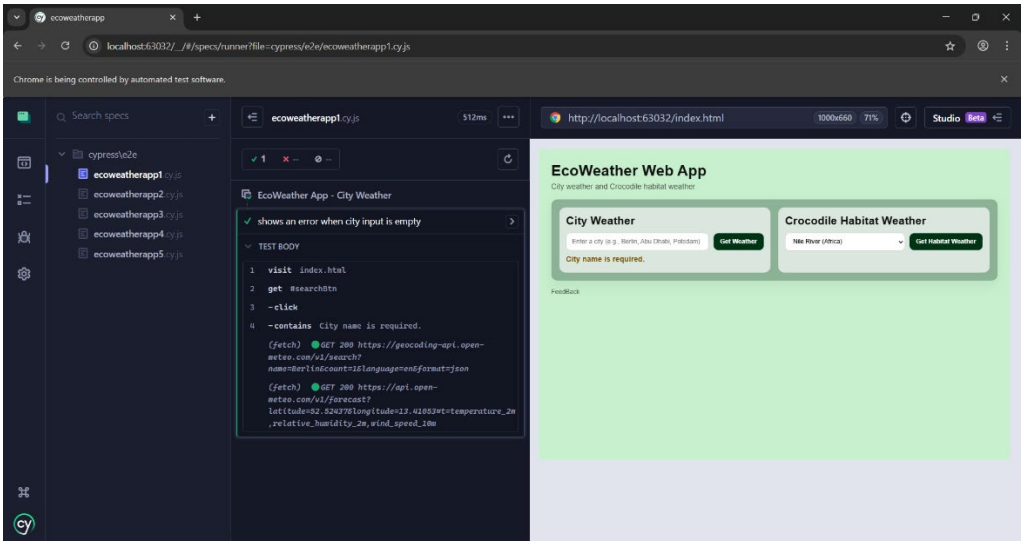
Cypress was selected as the testing library used for automated testing of EcoWeather due to its vast suitability options for front end web applications and its ability to test through real user interactions in a real time browser environment. Cyprus is an easy testing language to use, and it supports waiting, readable assertions, and easy querying making it the ideal testing tool for form-based applications.

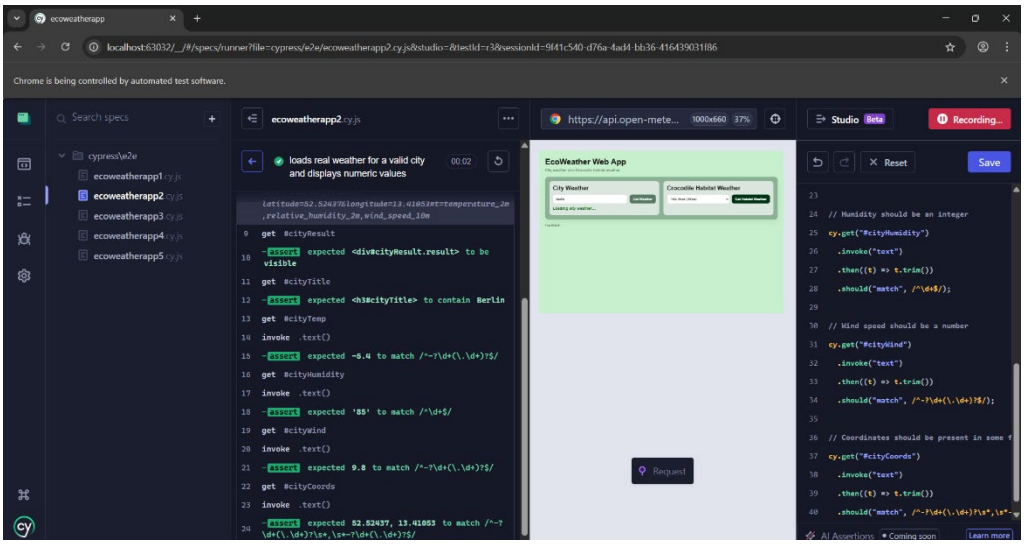
Automated test case

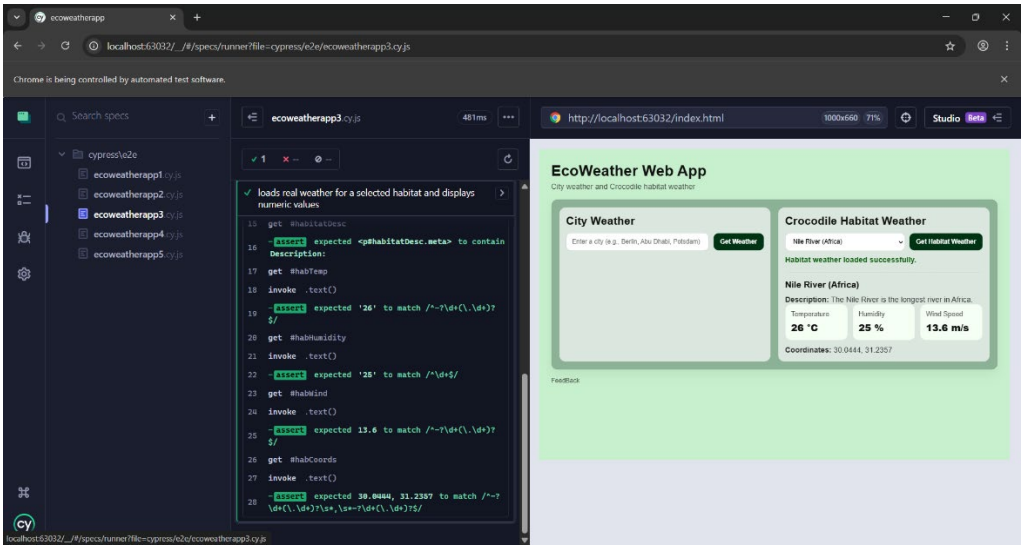
The automated test cases were put in place to validate all the key user story scenarios including the following:

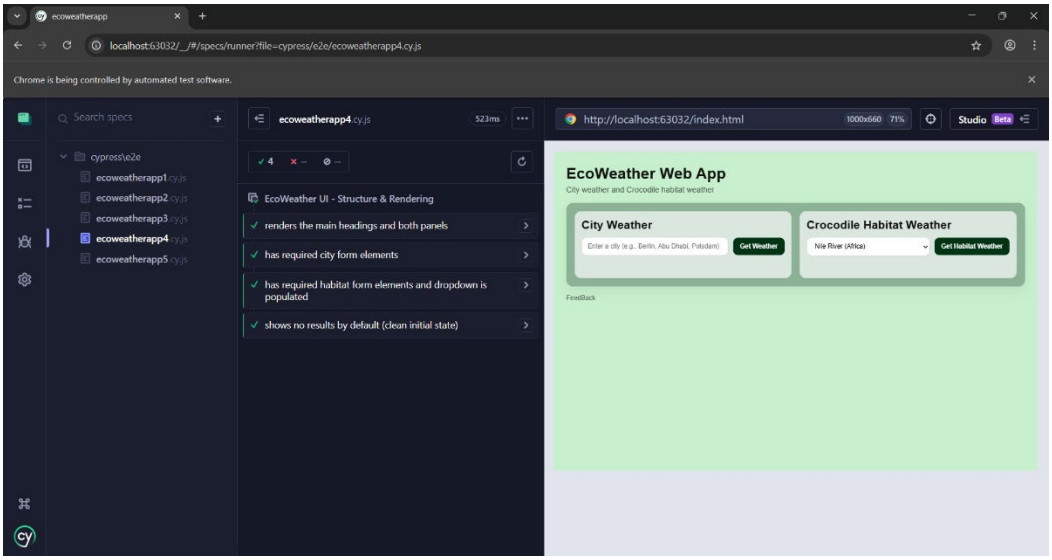
- Successful entry by user in city weather search text box,
- Handling of valid and invalid city inputs by user,
- Successful search retrieval and output of weather data upon search,
- Successful interaction with habitat weather drop-down list,
- Correct management of exceptions and error messages,
- Correct rendering of UI after API request and responses.

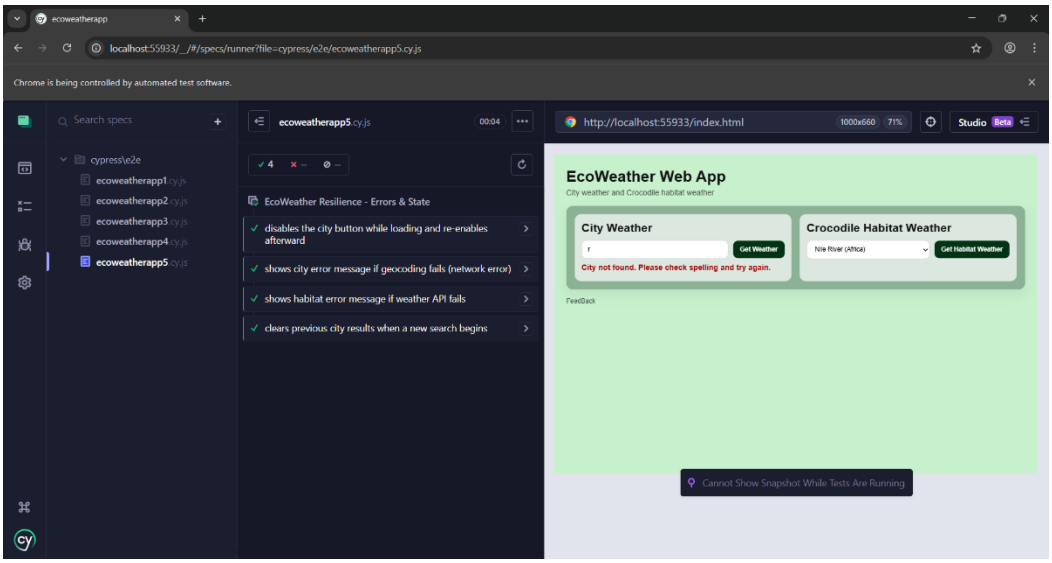
The testing method used here was end-to-end and each test specifies the purpose of the test and its acceptance criteria.


ecoweatherapp
localhost:63032/__/#/specs/runner?file=cypress/e2e/ecoweatherapp1.cy.js
Chrome is being controlled by automated test software.
Search specs
cypress\e2e
ecoweatherapp1.cy.js
ecoweatherapp2.cy.js
ecoweatherapp3.cy.js
ecoweatherapp4.cy.js
ecoweatherapp5.cy.js
ecoweatherapp1.cy.js
512ms
http://localhost:63032/index.html
1000x660
71%
Studio
Beta
EcoWeather App - City Weather
shows an error when city input is empty
TEST BODY
1 visit index.html
2 get #searchBtn
3 -click
4 -contains City name is required.
(fetch) GET 200 https://geocoding-api.open-meteo.com/v1/search?name=Berlin&count=1&language=en&format=json
(fetch) GET 200 https://api.open-meteo.com/v1/forecast?latitude=52.52437&longitude=13.41053&t=temperature_2m,relative_humidity_2m,wind_speed_10m
EcoWeather Web App
City weather and Crocodile habitat weather
City Weather
Enter a city (e.g., Berlin, Abu Dhabi, Potsdam)
Get Weather
City name is required.
Crocodile Habitat Weather
Nile River (Africa)
Get Habitat Weather
Feedback


ecoweatherapp
localhost:63032/__/#/specs/runner?file=cypress/e2e/ecoweatherapp2.cy.js&studio=&testId=r3&sessionId=9f41c540-d76a-4ad4-bb36-41643903186
Chrome is being controlled by automated test software.
Search specs
cypress\e2e
ecoweatherapp1.cy.js
ecoweatherapp2.cy.js
ecoweatherapp3.cy.js
ecoweatherapp4.cy.js
ecoweatherapp5.cy.js
ecoweatherapp2.cy.js
loads real weather for a valid city and displays numeric values
00.02
https://api.open-mete...
1000x660
37%
Studio
Beta
Recording...
Reset
Save
EcoWeather Web App
City Weather
Crocodile Habitat Weather
Request
AI Assertions
Coming soon
Learn more


ecoweatherapp
localhost:63032/__/#/specs/runner?file=cypress/e2e/ecoweatherapp3.cy.js
Chrome is being controlled by automated test software.
Search specs
cypress\e2e
ecoweatherapp1.cy.js
ecoweatherapp2.cy.js
ecoweatherapp3.cy.js
ecoweatherapp4.cy.js
ecoweatherapp5.cy.js
ecoweatherapp3.cy.js
481ms
http://localhost:63032/index.html
1000x660
71%
Studio
Beta
loads real weather for a selected habitat and displays numeric values
EcoWeather Web App
City weather and Crocodile habitat weather
City Weather
Enter a city (e.g., Berlin, Abu Dhabi, Potsdam)
Get Weather
Crocodile Habitat Weather
Nile River (Africa)
Get Habitat Weather
Habitat weather loaded successfully.
Nile River (Africa)
Description: The Nile River is the longest river in Africa.
Temperature
26 °C
Humidity
25 %
Wind Speed
13.6 m/s
Coordinates: 30.0444, 31.2357
Feedback





Static testing

Static testing of the EcoWeather app was implemented to identify code issues and enhance code quality without debugging and executing the program.

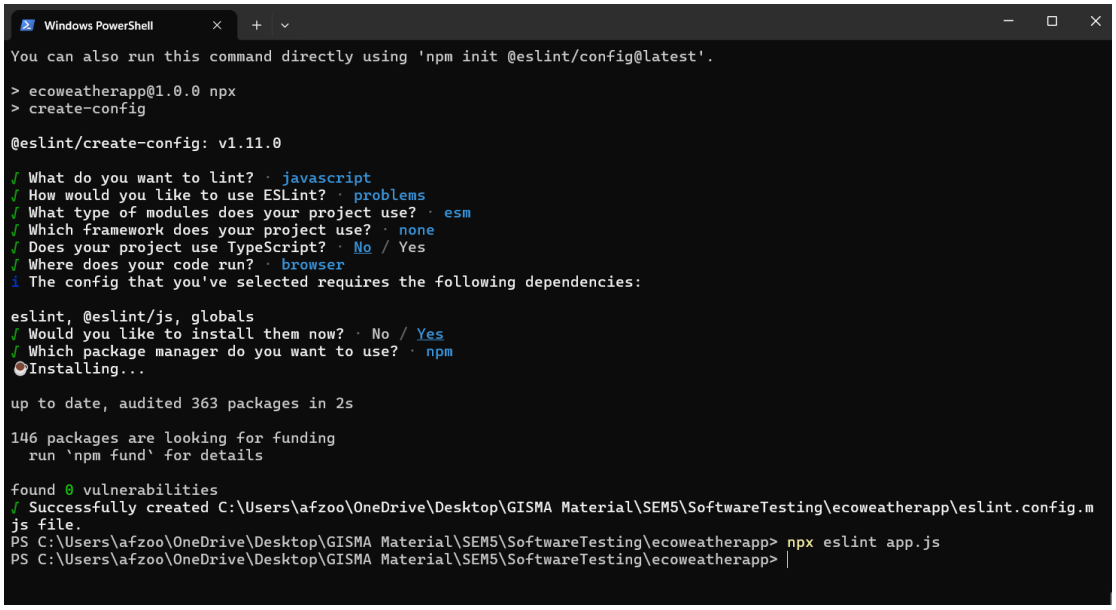
Static analysis tools used

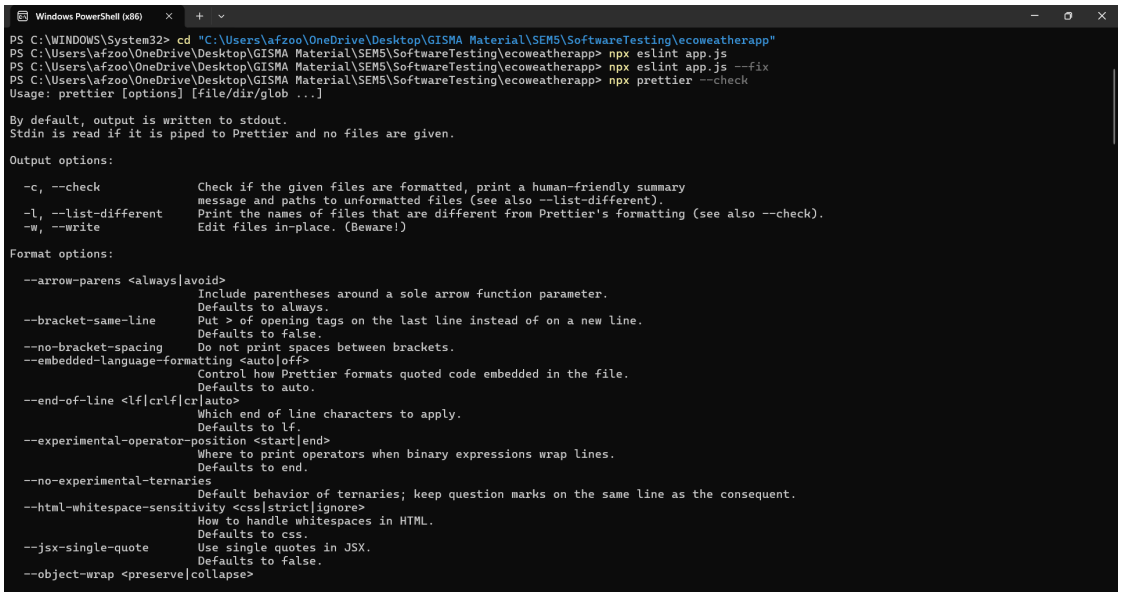
ESLint and Prettier were used to perform static analysis on the EcoWeather app to identify code issues and better code quality by fixing any inconsistent code patterns and removing any unused variables. Prettier was specifically used to implement consistent formatting across the HTML, CSS, and JavaScript files named index.html, style.css, and app.js. This form of static testing and analysis improved flexibility, maintainability, and reliability.

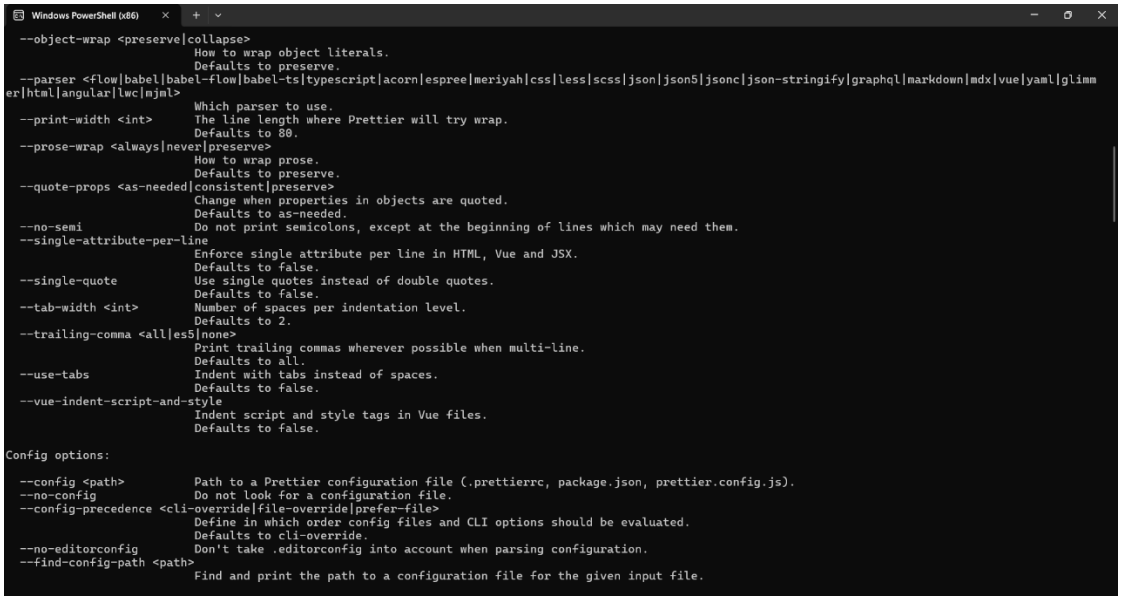
Static testing findings and resolutions

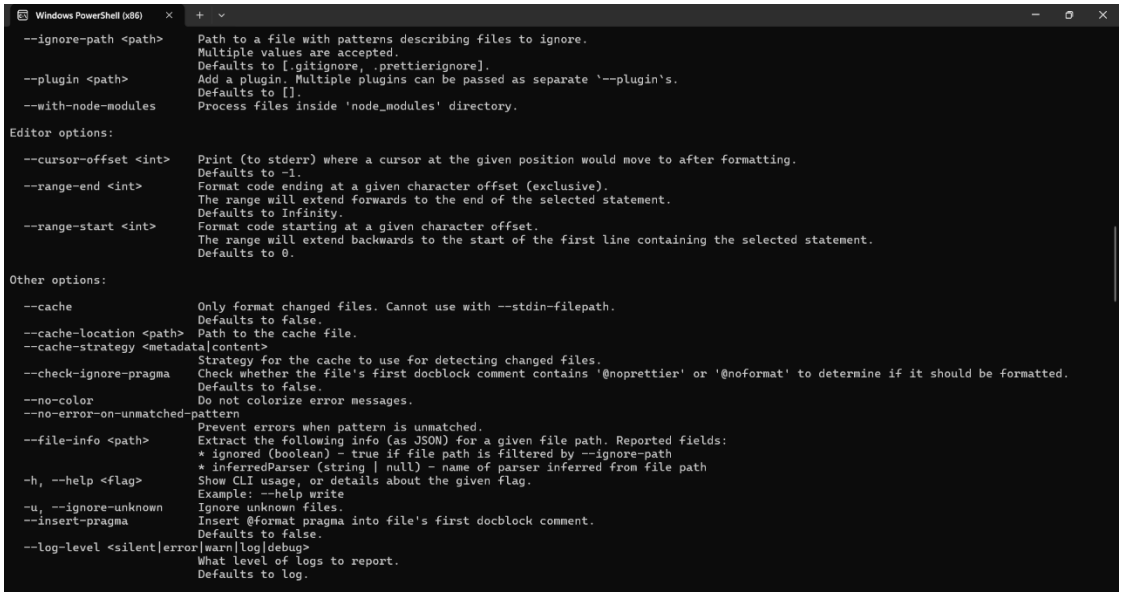
The static testing proved to be quite a success as no critical defects were found. Since no functional issues were identified, the minor formatting inconsistencies that were identified during early formatting checks were resolved by applying premiers formatting functionality which resulted in a consistent and readable source code throughout the application.

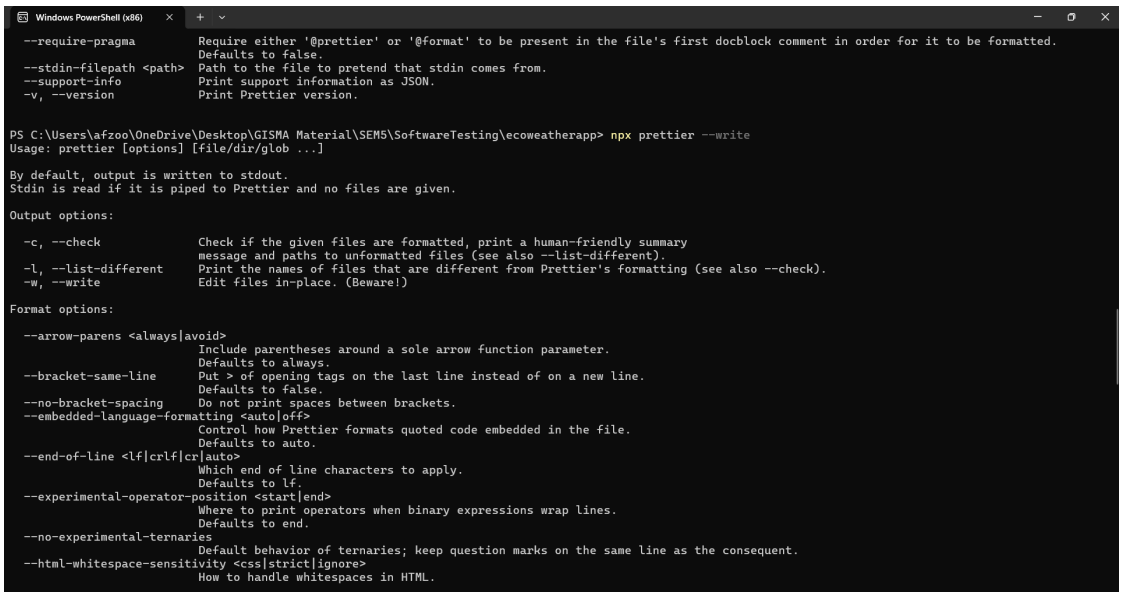
This static testing verified that the eco weather application met all the basic functional requirements and basic code quality expectations upholding its maintainability and reusability goals.

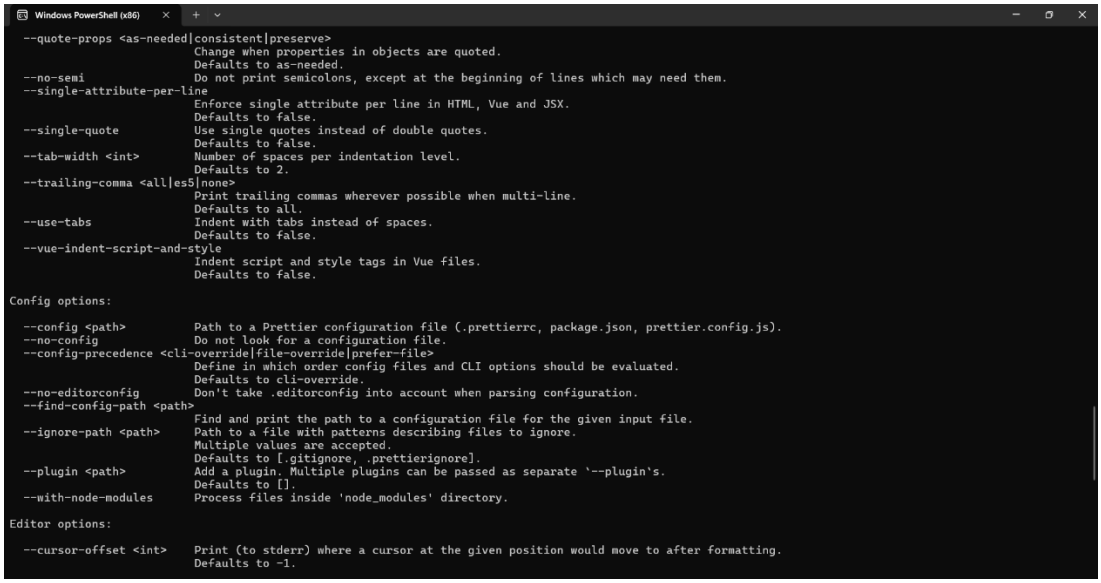


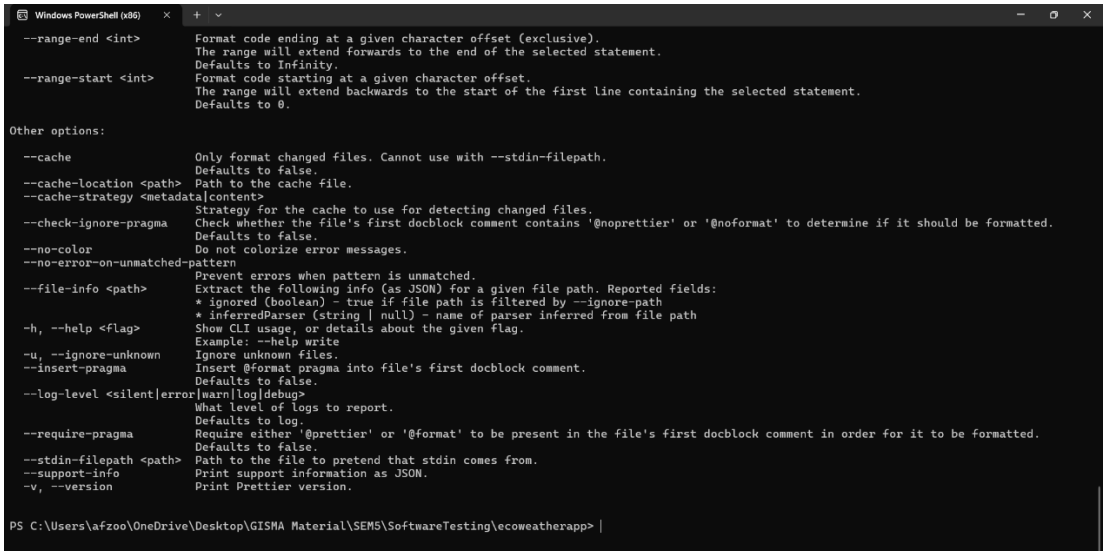



Windows PowerShell (x86)
--object-wrap <preserve|collapse>
How to wrap object literals.
Defaults to preserve.
--parser <flow|babel|babel-flow|babel-ts|typescript|acorn|espree|meriyah|css|less|scss|json|json5|jsonc|json-stringify|graphql|markdown|mdx|vue|yaml|glimmer|html|angular|lwc|mjml>
Which parser to use.
--print-width <int> The line length where Prettier will try wrap.
Defaults to 80.
--prose-wrap <always|never|preserve>
How to wrap prose.
Defaults to preserve.
--quote-props <as-needed|consistent|preserve>
Change when properties in objects are quoted.
Defaults to as-needed.
--no-semi Do not print semicolons, except at the beginning of lines which may need them.
--single-attribute-per-line
Enforce single attribute per line in HTML, Vue and JSX.
Defaults to false.
--single-quote Use single quotes instead of double quotes.
Defaults to false.
--tab-width <int> Number of spaces per indentation level.
Defaults to 2.
--trailing-comma <all|es5|none>
Print trailing commas wherever possible when multi-line.
Defaults to all.
--use-tabs Indent with tabs instead of spaces.
Defaults to false.
--vue-indent-script-and-style
Indent script and style tags in Vue files.
Defaults to false.
Config options:
--config <path> Path to a Prettier configuration file (.prettierrc, package.json, prettier.config.js).
--no-config Do not look for a configuration file.
--config-precedence <cli-override|file-override|prefer-file>
Define in which order config files and CLI options should be evaluated.
Defaults to cli-override.
--no-editorconfig Don't take .editorconfig into account when parsing configuration.
--find-config-path <path>
Find and print the path to a configuration file for the given input file.


Windows PowerShell (x86)
--ignore-path <path> Path to a file with patterns describing files to ignore.
Multiple values are accepted.
Defaults to [.gitignore, .prettierignore].
--plugin <path> Add a plugin. Multiple plugins can be passed as separate `--plugin`s.
Defaults to [].
--with-node-modules Process files inside 'node_modules' directory.
Editor options:
--cursor-offset <int> Print (to stderr) where a cursor at the given position would move to after formatting.
Defaults to -1.
--range-end <int> Format code ending at a given character offset (exclusive).
The range will extend forwards to the end of the selected statement.
Defaults to Infinity.
--range-start <int> Format code starting at a given character offset.
The range will extend backwards to the start of the first line containing the selected statement.
Defaults to 0.
Other options:
--cache Only format changed files. Cannot use with --stdin-filepath.
Defaults to false.
--cache-location <path> Path to the cache file.
--cache-strategy <metadata|content>
Strategy for the cache to use for detecting changed files.
--check-ignore-pragma Check whether the file's first docblock comment contains '@noprettier' or '@noformat' to determine if it should be formatted.
Defaults to false.
--no-color Do not colorize error messages.
--no-error-on-unmatched-pattern
Prevent errors when pattern is unmatched.
--file-info <path> Extract the following info (as JSON) for a given file path. Reported fields:
* ignored (boolean) - true if file path is filtered by --ignore-path
* inferredParser (string | null) - name of parser inferred from file path
-h, --help <flag> Show CLI usage, or details about the given flag.
Example: --help write
-u, --ignore-unknown Ignore unknown files.
--insert-pragma Insert @format pragma into file's first docblock comment.
Defaults to false.
--log-level <silent|error|warn|log|debug>
What level of logs to report.
Defaults to log.


Windows PowerShell (x86)
--require-pragma Require either '@prettier' or '@format' to be present in the file's first docblock comment in order for it to be formatted.
Defaults to false.
--stdin-filepath <path> Path to the file to pretend that stdin comes from.
--support-info Print support information as JSON.
-v, --version Print Prettier version.
PS C:\Users\afzoo\OneDrive\Desktop\GISMA Material\SEM5\SoftwareTesting\ecoweatherapp> npx prettier --write
Usage: prettier [options] [file/dir/glob ...]
By default, output is written to stdout.
Stdin is read if it is piped to Prettier and no files are given.
Output options:
-c, --check Check if the given files are formatted, print a human-friendly summary
message and paths to unformatted files (see also --list-different).
-l, --list-different Print the names of files that are different from Prettier's formatting (see also --check).
-w, --write Edit files in-place. (Beware!)
Format options:
--arrow-parens <always|avoid>
Include parentheses around a sole arrow function parameter.
Defaults to always.
--bracket-same-line Put > of opening tags on the last line instead of on a new line.
Defaults to false.
--no-bracket-spacing Do not print spaces between brackets.
--embedded-language-formatting <auto|off>
Control how Prettier formats quoted code embedded in the file.
Defaults to auto.
--end-of-line <lf|crlf|cr|auto>
Which end of line characters to apply.
Defaults to lf.
--experimental-operator-position <start|end>
Where to print operators when binary expressions wrap lines.
Defaults to end.
--no-experimental-ternaries
Default behavior of ternaries; keep question marks on the same line as the consequent.
--html-whitespace-sensitivity <css|strict|ignore>
How to handle whitespaces in HTML.



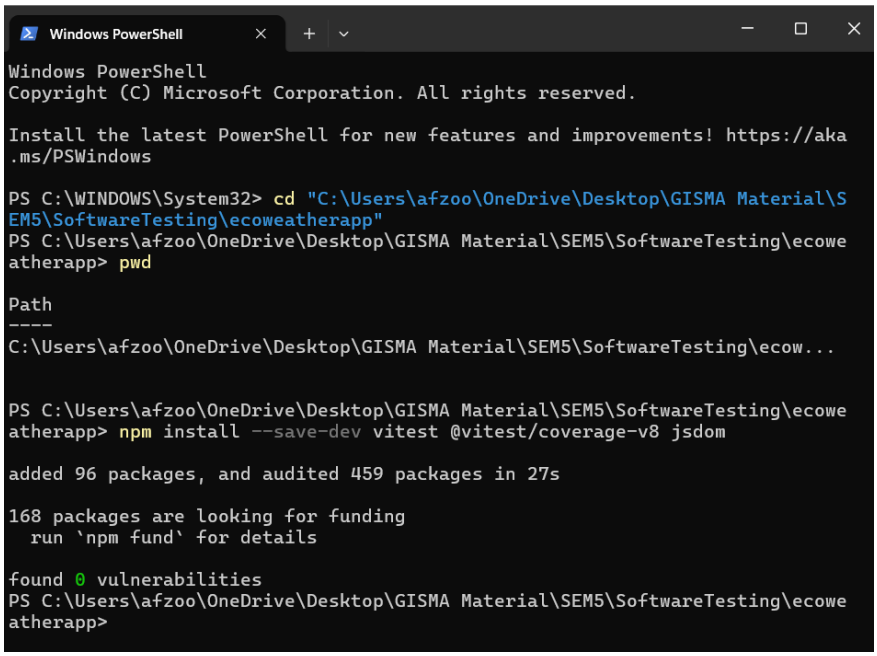


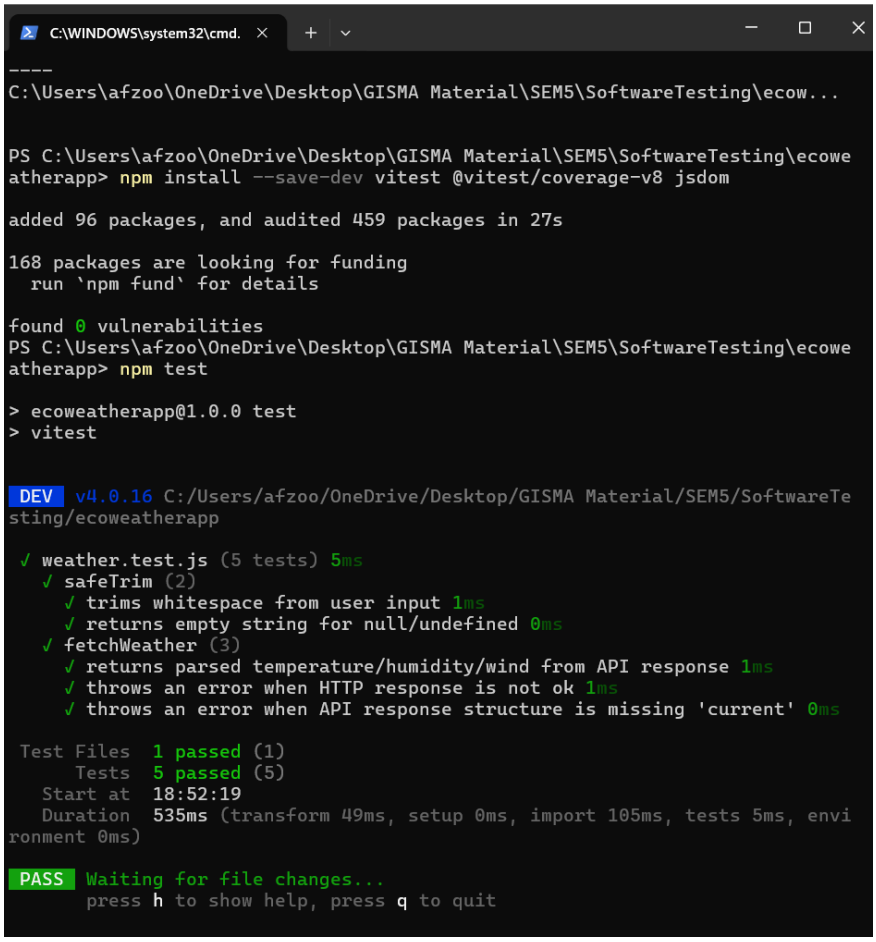
Bug report - Defect log

Bug ID	Summary	Steps to Test	Severity	Implemented Fix
1	City weather data not being displayed	Enter city and click ‘Get Weather”	High	API handling improved
2	Habitat drop down list empty	load page	Medium	Initialization function added
3	Error message not being displayed for invalid entries	submit empty or invalid city name	Medium	Input the validation added

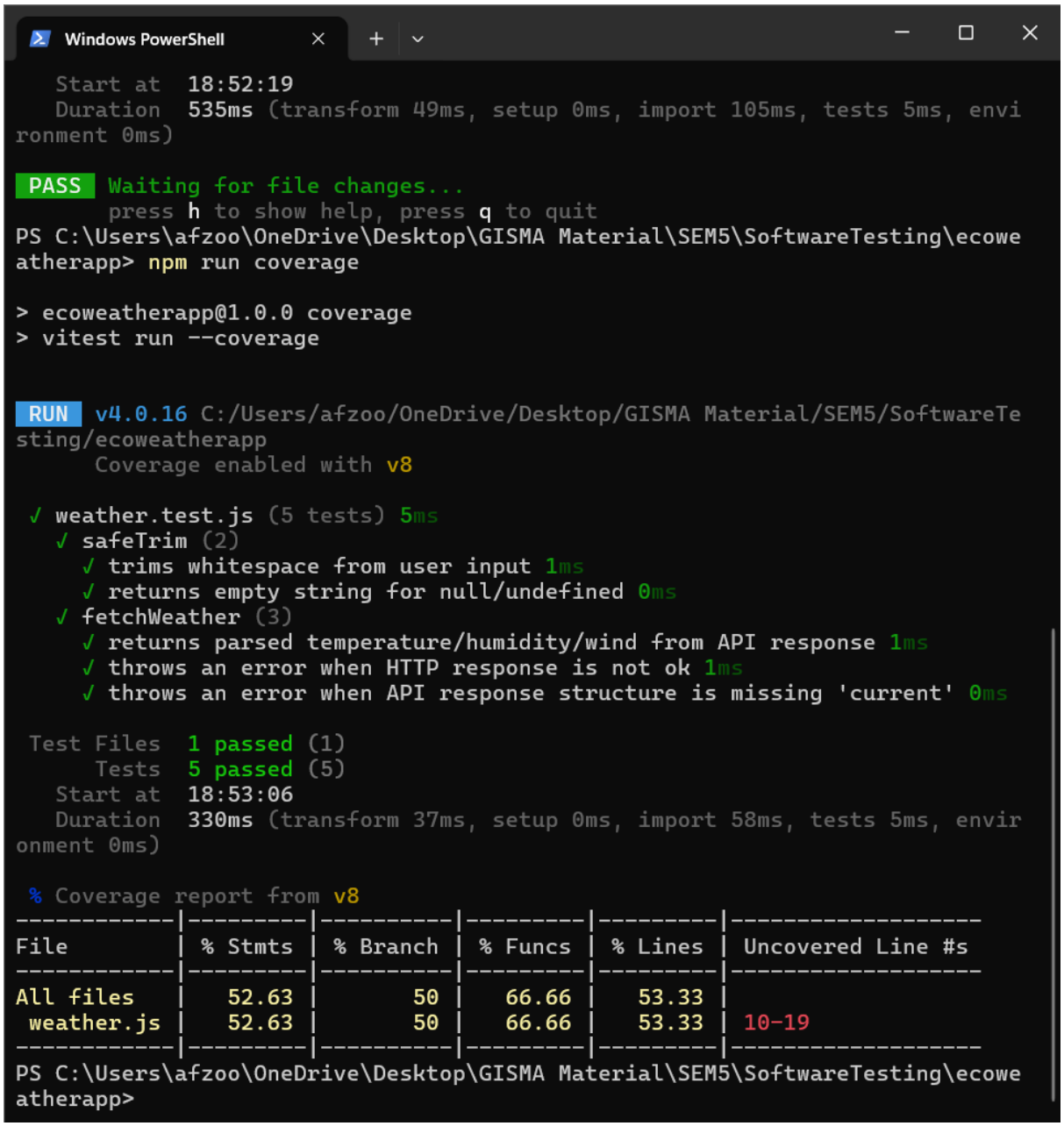
Test coverage

Vitest was used for test coverage reports to evaluate the application by focusing on coverage instead of line or branch metrics.





Coverage results



Covered analysis

The coverage results received by testing using Vitest demonstrate the success of all utility core functionalities while reaching over 52% line and branch coverage. Some uncovered lines seem to be present and are most likely related to paths that are less frequented in the application, which can be handling through future expansion and testing.

References

- Website template for online portfolio: https://themewagon.com/themes/free-bootstrap-4-cv-templatedownload/
- Portfolio website: https://afzoodnaaz.github.io/
- EcoWeather website: https://afzoodnaaz.github.io/ecoweatherapp/index.html
- Portfolio repository: https://github.com/afzoodnaaz/afzoodnaaz.github.io.git
- EcoWeather App repository: https://github.com/afzoodnaaz/EcoWeather-WebApp.git
- OneDrive – Source code, Video demo: https://gismauniversity-my.sharepoint.com/:f:/g/personal/afzood_surati_gisma-student_com/IgAQJEMnbUAVQ6FtKURS83h-AQSRGRNszwj7xO-neEkaOhI?e=w1HqOC