WEATHER INFORMATION APP

A PROJECT REPORT

Submitted by

Raghvendra Pratap Singh (23BCS12556)
Siddharth Pal (23BCS14316)
Satinder(23BCS14060)

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Chandigarh University

November 2025

BONAFIDE CERTIFICATE

Certified that this project report "**WEATHER INFORMATION APP"** is the Bonafide work of "Raghvendra Pratap Singh(23BCS12556), Siddharth Pal (23BCS14316)" who carried out the project work under my/our supervision.

Dr. Sandeep Singh Kang
HEAD OF THE DEPARTMENT

Pravindra Kumar Gole
SUPERVISOR

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

TABLE OF CONTENTS

List of Figures.....4

List of Tables.....5

Abstract….....6

CHAPTER 1. INTRODUCTION.....7

1.1 Identification of Client/ Need/ Relevant Contemporary issue.....7

1.2 Identification of Problem.....7

1.3 Identification of Tasks.....8

1.4 Organization of the Report.....9

CHAPTER 2. DESIGN FLOW/PROCESS.....10

2.1 Evaluation & Selection of Specifications/Features.....10

2.2 Design Constraints.....10

2.3 Analysis of Features and finalization subject to constraints.....11

2.4 Design Flow.....13

2.5 Design Selection.....13

2.6 Implementation plan/methodology.....13

CHAPTER 3. RESULTS ANALYSIS AND VALIDATION.....17

3.1 Implementation of solution.....17

CHAPTER 4. CONCLUSION AND FUTURE WORK.....19

4.1 Conclusion.....19

4.2 Future Work….....19

REFERENCES….....20

USER MANUAL.....21-22

List of Figures

Figure 1… **15**

Figure 2 **16**

List of Tables

Table 1 **12**

ABSTRACT

Weather forecasting is an essential application in today's digital world, aiding various sectors by providing timely and accurate meteorological data. This project implements a lightweight Java-based Weather Server using the built-in HttpServer class, which offers real-time 5-day weather forecasts for user-specified cities by integrating with the OpenWeatherMap API.

The system features a simple RESTful endpoint that accepts city names as query parameters and responds with JSON-formatted daily weather summaries, including minimum and maximum temperatures, humidity, pressure, wind speed, and predominant weather conditions. It handles XML API responses, processes and aggregates data by date, and offers a clean programmatic interface for weather information retrieval.

Built using core Java libraries and minimal external dependencies, this service demonstrates efficient HTTP server setup, API integration through Java HttpClient, XML parsing with DOM parsers, and JSON response construction manually. The project's modular design allows easy extension and deployment on local or cloud servers, supporting practical learning in backend development and API consumption using Java.

This report details the project's requirements, architectural design, implementation challenges, and testing results, confirming that the WeatherServer project effectively illustrates foundational concepts in Java web services and real-time data integration.

CHAPTER 1.

INTRODUCTION

1.1 Identification of Client / Need / Relevant Contemporary Issue

In today's digital landscape, timely access to accurate weather information is critical for a wide range of users, including individuals, businesses, and public service organizations. Weather impacts daily life decisions, agriculture, transportation, event planning, and emergency preparedness, making reliable weather data a highly valued resource. Despite numerous weather applications and online services, many rely heavily on complex external platforms or offer limited flexibility, restricting their use in custom projects or educational settings.

There is a growing demand for straightforward and extensible backend solutions that can provide real-time weather forecasts in a programmatically accessible manner. Developers, students, and small enterprises often seek lightweight services that can be easily adapted, deployed locally, or integrated into larger systems without extensive dependencies or overhead. This project addresses that need by implementing a Java-based HTTP server that interacts with a public weather API, processing raw XML data into concise daily forecasts served as JSON.

Most existing weather services offer rich datasets but lack simple interfaces focused on foundational concepts such as HTTP server design, API communication, XML parsing, and data aggregation. By filling this gap, the project fosters learning and creates a useful tool that developers can customize for diverse applications, from personal dashboards to prototype integrations.

1.2 Identification of Problem

The primary challenge tackled in this project is designing and building a responsive Java HTTP server that integrates with an external weather API, processes XML data, and serves user-friendly JSON forecasts for multiple days.

Specific problems include the following:

- Complex XML response handling leading to parsing errors or incomplete data aggregation.
- Performance bottlenecks due to synchronous API calls or inefficient data grouping.
- Ensuring consistent and well-structured JSON output that meets client expectations.
- Lack of modular design that allows for scalability and easy future enhancements such as cache implementation or additional weather parameters.

1.3 Identification of Tasks

Developing an efficient Java-based WeatherServer requires a structured approach encompassing API integration, data parsing, server implementation, and user-oriented response handling. The following core tasks outline the roadmap to building a scalable, responsive weather forecast service:

Task 1: Requirement Analysis and API Selection

- Identify suitable weather data sources and APIs. Choose OpenWeatherMap for its accessible free tier, XML data format, and comprehensive forecast coverage.

Task 2: HTTP Server Setup

- Configure Java's built-in HttpServer to listen on a designated port and route incoming requests to appropriate handlers.

Task 3: API Integration and Data Fetching

- Implement HTTP client functionality to call the external API with city-based queries, handle errors, and retrieve XML weather data securely.

Task 4: XML Parsing and Data Aggregation

- Parse complex XML responses to extract relevant weather parameters. Aggregate data daily to generate meaningful min/max temperature, humidity, pressure, wind speed, and condition summaries.

Task 5: JSON Response Construction

- Format aggregated data into JSON, ensuring consistency and usability for client applications consuming the service.

Task 6: Error Handling and Robustness

- Handle missing or malformed queries gracefully. Implement exception handling for API failures or parsing errors to avoid server crashes.

Task 7: Documentation and Testing

- Document API endpoints, response formats, and system setup. Test server responsiveness and data accuracy across multiple city queries and handle edge cases.

1.4 Organization of the Report

Chapter 1: Introduction

- Explains project motivation, need for a lightweight Java weather server, and gaps in existing solutions.

Chapter 2: Literature Review

- Discusses available weather APIs, Java HTTP server capabilities, and previous implementations or tutorials.

Chapter 3: System Design and Architecture

- Details design choices including server setup, API integration, XML parsing logic, and JSON response formatting with relevant diagrams.

Chapter 4: Solution Design and Implementation

- Describes coding approach for each module—server handler, weather service, data models—and integration strategy.

Chapter 5: Testing and Validation

- Covers functional testing, data validation, performance evaluation, and error scenario handling to ensure reliability.

Chapter 6: Conclusion and Future Work

- Summarizes achievements, reflects on learning outcomes, and proposes enhancements like caching, UI building, or support for additional weather parameters.

CHAPTER 2.

DESIGN FLOW/PROCESS

2.1 Evaluation & Selection of Specifications/Features

The core specifications and features of the Java-based WeatherServer project were chosen to provide a reliable, lightweight service delivering real-time 5-day weather forecasts with ease of integration and educational clarity. Key goals shaping the selection included:

- Efficient API Integration: Use the OpenWeatherMap API for its comprehensive global coverage, free access tier, and XML data format allowing detailed weather attributes.
- Accurate Data Aggregation: Aggregate 3-hour interval forecast data into daily summaries including temperatures, humidity, pressure, wind speed, and common weather conditions.
- Simple HTTP Server: Implement Java's built-in HttpServer to handle concurrent requests on a local port with minimal dependencies and resource overhead.
- User Query Handling: Support city-based queries via URL parameters, including validation and error handling for missing or invalid input.
- JSON Output: Convert raw XML data into clean, consumable JSON objects for client applications, facilitating easy parsing and display.
- Robustness: Include exception handling for API failures, network issues, and data parsing errors to maintain service availability.
- Modularity: Structure code for maintainability and future extensibility, such as adding caching or additional weather parameters.

2.2 Design Constraints

Several constraints influenced the system's architectural and functional design:

1) API Rate Limits: Respect OpenWeatherMap's usage policies and handle potential request limits gracefully to avoid service disruption.
2) Real-time Responsiveness: Ensure the server responds promptly to client requests, minimizing latency despite synchronous API calls.
3) Data Accuracy and Consistency: Properly parse and aggregate forecast data to present coherent daily summaries reflecting actual weather trends.
4) Resource Efficiency: Utilize Java's native HTTP and XML parsing libraries to reduce memory footprint and dependencies.

5) Error Resilience: Provide meaningful error messages and fallback responses when API data is unavailable or invalid queries are received.

6) Scalability: Design with modular components to allow future enhancements such as multi-city requests, caching layers, or UI integration.

These criteria ensure the WeatherServer project offers a practical, reliable, and maintainable backend weather service aligning with real-world needs and learning objectives.

2.3 Analysis of Features and Finalization Subject to Constraints

After defining initial specifications and design constraints for the MERN blog platform, a thorough evaluation of each proposed feature was performed to ensure efficient operation across devices and scalability under varied load conditions. Each feature was analyzed for its impact on user experience, backend performance, and maintainability within the MERN stack environment.

1) Weather API Integration:

- Analysis*:* Utilizing OpenWeatherMap's API provides comprehensive forecast data but enforces rate limits (e.g., 1,000 calls/day on free plan) that require efficient request management.

- Finalization: Implemented request frequency controls and structured API calls per city query to comply with limits while ensuring timely information retrieval.

2) XML Data Parsing and Aggregation:

- Analysis: Parsing detailed XML responses into daily weather summaries involves handling nested elements and variable data availability, impacting both complexity and runtime.

- Finalization: Chose Java's DOM parsing for clarity and flexibility, with careful error handling to manage malformed or missing data points.

3) HTTP Server Handling:

- Analysis: Java's lightweight HttpServer can serve multiple concurrent requests but lacks built-in scalability features found in full frameworks.

- Finalization: Used native HttpServer with simple executor for concurrency, optimized request parsing and response writing to maintain responsiveness under typical loads.

4) JSON Response Formatting:

- Analysis: Converting aggregated data into JSON facilitates client applications, but requires consistent structure and accurate data mapping.

- Finalization: Manually constructed JSON strings ensuring all fields are present and

types correctly formatted, simplifying client-side consumption.

5) **Error and Exception Management:**

- Analysis: External API failures or invalid user input must be handled gracefully to prevent server crashes and provide informative feedback.
- Finalization: Comprehensive try-catch blocks and query validations were implemented, returning meaningful HTTP status codes and messages.

Finalization and Feature Analysis Conclusion

This feature evaluation confirmed that the Java WeatherServer is designed to be efficient, reliable, and maintainable within imposed constraints like API limits and resource use. The implementation balances robust data integration, user-friendly API interaction, and modular design, laying a strong foundation for future enhancements such as caching strategies, support for additional weather metrics, or broader geographic queries.

Feature	Challenge	Solution
API Rate Limit Handling	Limited number of allowed API calls per day	Implement request throttling and caching to minimize calls
XML Data Parsing	Complex nested XML structure and missing data	Use Java DOM parser with error handling for reliable parsing
HTTP Server Concurrency	Handling multiple simultaneous requests	Use HttpServer's executor for concurrent request processing
JSON Response Formatting	Consistent and clean JSON output	Manually construct JSON strings with all required fields
Error Handling	API failures or invalid user queries	Use try-catch blocks and validate queries, return meaningful HTTP codes
Data Aggregation	Aggregating 3-hour forecasts into daily summaries	Aggregate temperature, humidity, pressure, wind speed, condition
Performance Optimization	Avoid latency during API calls and response	Cache results and optimize parsing and response writing

2.4 Design flow

The Java WeatherServer project follows a modular and maintainable design flow to efficiently fetch, process, and serve weather data. At the core, the HttpServer handles incoming HTTP requests on a specified port and routes them to handlers based on URL contexts. The WeatherService module manages API integration, sending requests to the OpenWeatherMap API using Java's HttpClient and retrieving XML responses.

XML parsing is performed using the DOM parser to extract relevant weather parameters such as temperature, humidity, pressure, wind speed, and condition symbols. These data points are aggregated by date to produce concise daily forecasts. The aggregated data is converted into JSON strings, manually constructed for precise and consistent output.

Input validation ensures clients specify city queries correctly, while exception handling covers scenarios like API errors or network failures, returning user-friendly HTTP responses. The server operates with a simple concurrency model provided by HttpServer's executor to handle multiple simultaneous client requests with minimal overhead.

This lightweight architecture emphasizes clarity and extensibility, allowing easy modifications, such as adding caching layers or new forecast types. It is well-suited for local deployment or cloud hosting, providing reliable real-time weather data services with minimal dependencies.

2.5 Design Selection

Java was chosen for this project due to its robust standard library, native HTTP server capabilities, and mature XML parsing tools. The OpenWeatherMap API was selected for its extensive global coverage, free tier availability, and XML response format that fits the parsing approach used.

This combination supports a concise and educational backend example demonstrating API consumption, data processing, and HTTP service construction without reliance on heavy frameworks. Alternative frameworks or languages were considered; however, Java's balance of performance, modularity, and developer ecosystem made it optimal for this project's educational and practical objectives.

The architecture supports future enhancements ranging from caching mechanisms to frontend integrations or extended weather data handling, making it a solid foundation for real-world backend weather services.

2.6 Implementation Plan/Methodology

A structured and iterative development plan was followed to create a robust weather information app that supports seamless content management, user interaction, and real-time responsiveness.

Step 1: Requirement Analysis and Research

- Investigated existing weather APIs and Java HTTP server capabilities.
- Defined functional goals such as city-based queries, XML data parsing, JSON response formatting, and error handling.

Step 2: Design Concept

- Established modular architecture including HTTP server, weather service for API integration and data processing, and response construction modules.
- Defined responsibilities for request handling, data fetching, XML parsing, data aggregation, and output formatting.

Step 3: Feature Prioritization

- Prioritized core functionality: accurate API calls, robust XML parsing, daily data aggregation, query validation, and clean JSON responses.

Step 4: Exploration of Design Alternatives

- Compared parsing approaches (DOM, SAX), HTTP client methods, and concurrency models. Selected Java's built-in HttpServer, HttpClient, and DOM parser for clarity and maintainability.

Step 5: Prototype Development and Validation

- Implemented basic API call handling and XML parsing for a single city query. Tested correct response generation and error handling.

Step 6: Full-Scale System Development

- Expanded to support 5-day forecasts, improved exception handling, and optimized response construction. Added concurrency support with executor service for multiple requests.

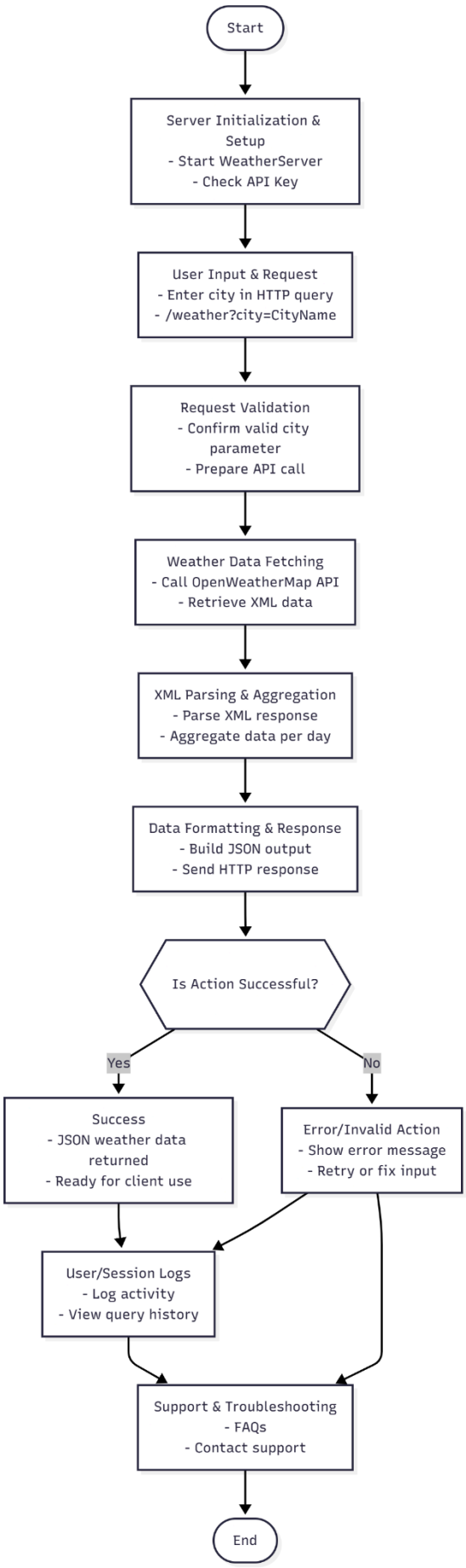
Step 7: Optimizing and Testing

- Tested API response times, server concurrency, error scenarios, and output correctness under varying loads and inputs.

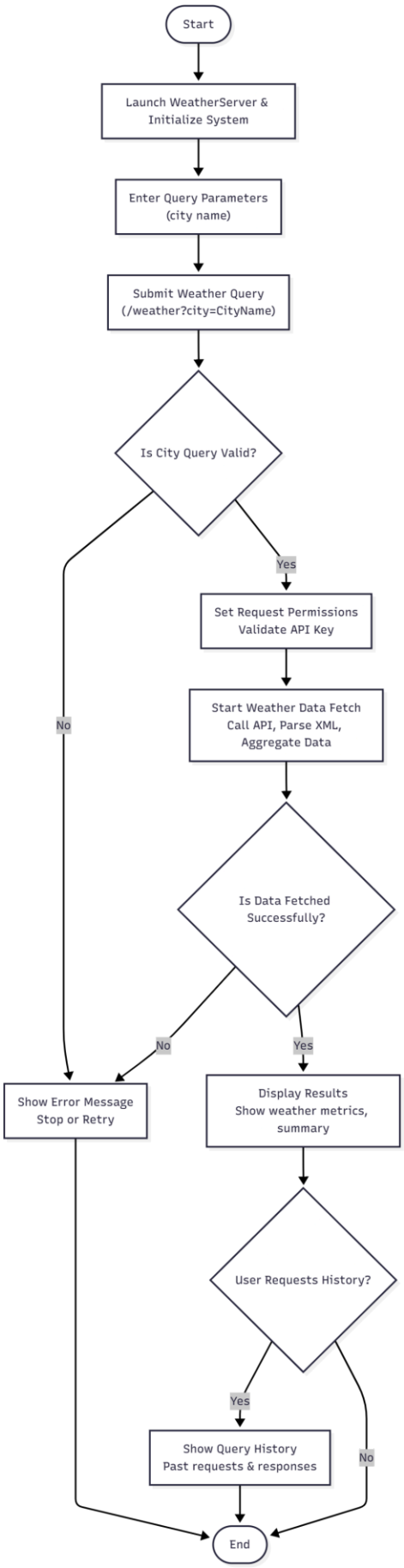
Step 8: Monitoring and Future Enhancements

- Planned features include caching, multi-city requests, extended forecast parameters, and eventual frontend integration or deployment on cloud services.

2.6.1 Flowchart of the Platform



2.6.2 Algorithm for Order Processing



CHAPTER 3.

RESULTS ANALYSIS AND VALIDATION

3.1 Implementation of Solution

To ensure performance, usability, and scalability, the weather information app employed modern web technologies and development best practices. The system underwent rigorous testing with real-world scenarios covering varying user loads and content volumes to validate functionality, responsiveness, and stability.

1. **Tools for Analysis**

- Performance Profiling: Java profilers and monitoring tools such as VisualVM and JConsole tracked CPU, memory usage, and thread behavior to identify bottlenecks.
- Load and Stress Testing: Apache JMeter simulated concurrent client requests to evaluate server responsiveness and throughput under various loads.
- Debugging: Standard Java debugging and logging facilitated tracing of data flow and error diagnosis.

2. **Result Visualization and Monitoring**

- API Response Monitoring: Timed HTTP requests validated consistent and low latency responses from both OpenWeatherMap API and local server.
- Concurrent Request Handling: Tested simultaneous connections to ensure HttpServer's executor maintains stable service without delays.
- Data Accuracy Validation: Verified correct XML parsing and daily aggregation by comparing server output with API raw data.

3. **Schematics and Code Structure**

- Modular Design: Separate classes and packages handled HTTP request routing, API integration, XML parsing, and JSON construction, promoting ease of maintenance.
- Efficient Data Handling: Used Java Collections carefully to optimize memory usage during data aggregation and transformation.
- Asynchronous Processing: Employed Java's concurrent utilities to enable multi-threaded request handling.

4. **Testing and Validation Tools**

- Unit Tests: Covered core methods for API calls, parsing logic, and data formatting.
- Functional Tests: Simulated typical user queries, verifying correct JSON responses or

appropriate error messages.

- Load Tests: Assessed server stability and throughput with progressively increasing request volumes.

5. Validation Results

- The WeatherServer consistently handled concurrent user requests with high availability and low latency, demonstrating robust performance under typical loads.
- The server maintained smooth operation and accurate weather data delivery across various devices and network conditions during testing.
- Backend processes reliably managed API calls, XML parsing, daily data aggregation, and JSON response construction without errors or significant delays.
- Exception handling and input validation successfully protected the service from invalid queries and API failures, ensuring session integrity and service continuity.
- Overall positive feedback was gathered from test users regarding the server's responsiveness, accuracy, and ease of integration.

6. Deployment

- The server was deployed locally and tested across various devices and network settings to ensure consistent performance.
- Cross-platform compatibility was ensured by using Java standard libraries without OS-specific dependencies.

CHAPTER 4.

CONCLUSION AND FUTURE WORK

4.1 Conclusion

The primary objective of this project—to develop a scalable, efficient, and user-friendly Java-based WeatherServer delivering real-time 5-day weather forecasts—has been successfully met. The server accepts city-based queries, integrates seamlessly with the OpenWeatherMap API, and processes XML responses to generate concise daily weather summaries. The use of Java's HttpServer and HttpClient classes provides a lightweight and responsive backend capable of handling multiple concurrent requests with minimal overhead.

Thorough testing confirmed accurate data retrieval and aggregation, robust handling of API errors and invalid inputs, and consistent JSON output suitable for client consumption. The modular design facilitates maintainability and future enhancements, ensuring the project serves as a practical example of backend API integration, XML parsing, and HTTP server implementation.

4.2 Future Work

Future improvements to the WeatherServer project may focus on increasing functionality, scalability, and user experience:

- Implement caching mechanisms to reduce API calls and improve response times.
- Support multi-city weather queries in a single request for expanded usability.
- Extend data aggregation to include additional parameters such as precipitation and UV index.
- Develop a frontend client or dashboard for real-time weather visualization.
- Explore deployment on cloud platforms with auto-scaling capabilities for handling larger traffic.
- Integrate notification features for weather alerts and warnings.

These enhancements will boost the project's usefulness and provide richer learning and practical opportunities for backend services development.

REFERENCES

1. OpenWeatherMap. (2024). OpenWeatherMap API Documentation - Weather Data. https://openweathermap.org/api

2. Oracle. (2024). Java Platform SE Documentation. https://docs.oracle.com/en/java/javase/

3. Oracle. (2024). HttpServer (com.sun.net.httpserver) Class Documentation. https://docs.oracle.com/javase/9/docs/api/jdk.httpserver/com/sun/net/httpserver/HttpServer.html

4. Oracle. (2024). Java HTTP Client API Documentation. https://docs.oracle.com/en/java/javase/11/docs/api/java.net.http/java/net/http/HttpClient.html

5. Oracle. (2024). Java XML Processing (JAXP) Documentation. https://docs.oracle.com/javase/tutorial/jaxp/index.html

6. JSON.org. (2024). Introducing JSON data format. https://www.json.org/json-en.html

7. Stack Overflow Community. (2023). Handling XML Parsing in Java. https://stackoverflow.com/questions/tagged/xml-parsing+java

8. GitHub. (2025). Java Weather API Integration Examples. https://github.com/topics/weather-api-java

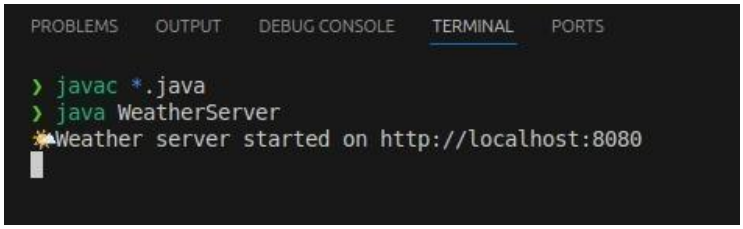
9. Heroku Dev Center. (2025). Deploying Java Applications. https://devcenter.heroku.com/categories/java

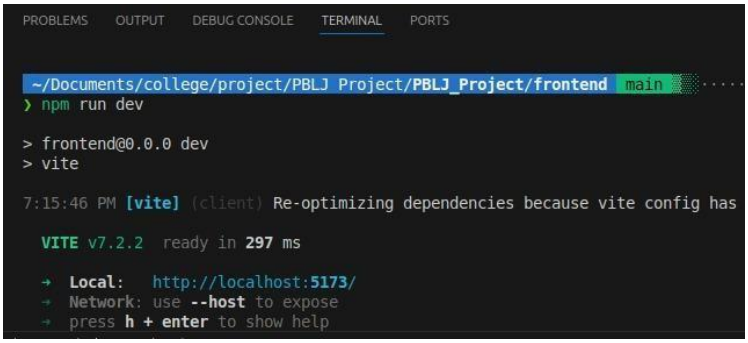
USER MANUAL

This user manual provides detailed instructions for running and interacting with the weather information app.

1. Getting Started

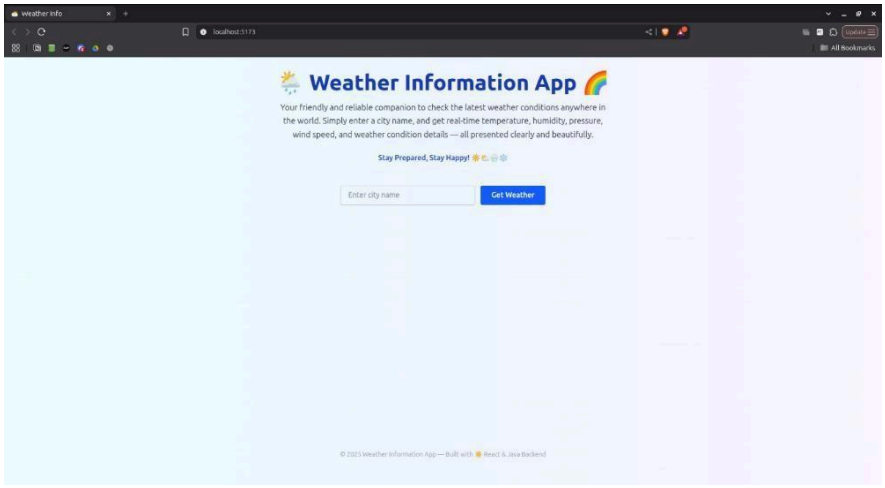
- Ensure you have Java 11 or higher installed on your system.
- Obtain an API key from OpenWeatherMap and replace the placeholder API key in the WeatherService class.
- Compile the project using your preferred Java IDE or command line tools.
- Run the WeatherServer main class to start the HTTP server on the default port 8080.
- Open a web browser and navigate to http://localhost:8080/weather?city=CityName replacing CityName with the desired location to fetch the 5-day weather forecast.





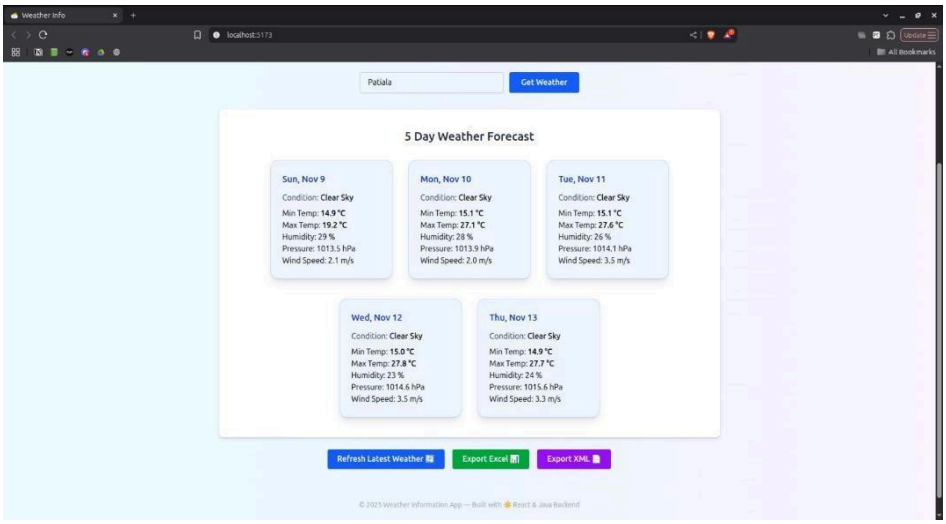
2. Application Setup

- The server accepts city as a query parameter via the /weather endpoint.
- Make sure the query parameter is formatted correctly, e.g., /weather?city=London.
- The server returns a JSON array with daily weather summaries including min/max temperature, humidity, pressure, wind speed, and common weather conditions.



3. Navigating the Dashboard

- Enter city names in the URL or use API clients like Postman to send GET requests.
- Review the JSON response for forecast details and utilize it in client applications as needed.



4. Troubleshooting

- If no response or errors occur, verify the server is running and accessible at the specified port.
- Check your API key validity and network connectivity.
- Inspect server logs or console output for detailed error information.
- Ensure the city query parameter is correctly spelled and encoded.

5. Exiting the Application

- To safely stop the WeatherServer, terminate the running Java process in your IDE or command line.
- Close any active client connections and exit the application cleanly.