DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

SANTOSH, TANGAIL-1902, Dhaka, Bangladesh

A Project Report

On

Web Project Using Java Spring Boot: Weather API & User Authentication

For The Course

"Project-II"

Course Code:ICT-2200

By

Sahriar Ahmad Tarek- IT-22047

Annay Das – IT-22048

Supervised by

Dr. Abir Hossain

Declaration

We hereby declare that the project titled **"Weather Forecasting Web Application using Spring Boot"** has been carried out by us as part of our academic curriculum under the supervision of **Dr. Abir Hossain**, at the **Department of Information and Communication Technology, Mawlana Bhashani Science and Technology University (MBSTU), Tangail, Bangladesh**.

We confirm that this project is original and has not been submitted elsewhere for any degree, diploma, or academic certification. Any references or data taken from other works—whether published or unpublished—have been duly acknowledged throughout the report, and a comprehensive list of references is provided at the end.

The development and documentation of this project reflect our understanding and application of web technologies and software engineering principles learned during our academic course.

Signature of Supervisor

Dr. Abir Hoissain

Associate Professor
Dept of ICT
MBSTU

Acknowledgement

All praise is due to Almighty Allah, whose boundless mercy and blessings have enabled us to complete this project successfully.

We would like to express our deepest gratitude to **Dr. Abir Hossain**, our respected supervisor and a faculty member of the **Department of Information and Communication Technology at Mawlana Bhashani Science and Technology University (MBSTU), Tangail, Bangladesh**. His continuous guidance, insightful suggestions, and persistent encouragement were invaluable throughout the development of this project titled **"Weather Forecasting Web Application using Spring Boot"**.

We sincerely appreciate his support, which not only helped us overcome the technical challenges but also motivated us to maintain the highest standards in our work. His keen observations, constructive feedback, and patient mentorship enriched our learning experience immensely.

Our heartfelt thanks also go to our respected teachers and the department for providing us with the infrastructure and academic environment necessary for completing this project.

Lastly, we would like to express our profound appreciation to our families. Their unwavering love, support, and encouragement have been the backbone of our journey, and we remain forever grateful for their presence in our lives.

Weather Forecast Web Application – Project Report

Table of Contents

1. Introduction
2. Objective of the Project
3. Project Overview
4. System Architecture
5. Technologies Used
6. Functional Requirements
7. Non-Functional Requirements
8. Database Design
9. RESTful API Integration
10. User Registration & Login Module
11. Weather Data Fetching Module
12. Front-End Design
13. Controller Layer
14. Service Layer
15. Repository Layer
16. Security and Best Practices
17. Testing and Validation

18. Challenges and Solutions
19. Future Enhancements
20. Team Contribution
21. Supervisor’s Role
22. Tools Used
23. Screenshots
24. Summary
25. References

1. Introduction

Weather forecasting is a significant and frequent requirement in our daily life. This project is a simple yet effective weather forecast web application built using Spring Boot, Thymeleaf, and OpenWeatherMap API. It allows users to register, log in, and view real-time weather conditions of any city worldwide. The system combines backend data processing with intuitive front-end presentation, making it suitable for practical use and educational purposes. As climate awareness grows and weather-related risks increase, a user-friendly application like this one helps the common people access weather updates in a timely manner.

This report provides a complete documentation of the system design, implementation strategy, code explanation, and user interface aspects. Our goal is to provide a working weather system that uses third-party APIs, integrates database persistence, and maintains modular and readable code. We also focus on adopting security practices, usability in the front-end, and extensibility in the backend.

The application supports fundamental software development practices like MVC (Model View Controller) pattern, RESTful API consumption, form validation, and database interaction using JPA. This report discusses each component of the project in detail, from the architecture level to the source code implementation.

2. Objective of the Project

The main objectives of this weather forecasting web application project are:

- To develop a full-stack Java web application with Spring Boot and Thymeleaf.
- To integrate a public RESTful API (OpenWeatherMap) for real-time weather information.

- To design user authentication functionalities including registration and login.
- To provide a responsive and intuitive interface for user interactions.
- To demonstrate the use of MVC architecture with clear separation of concerns.
- To enable CRUD operation using JPA and MySQL for user data storage.
- To apply password security using hashing algorithms.
- To document the design, architecture, implementation, and testing of the application.

3. Project Overview

The weather forecast web application provides weather data for any city entered by the user. The user interface allows users to input a city name, and the application fetches and displays the weather information from the OpenWeatherMap API.

The application is divided into several modules:

1. **User Module**: Handles user registration and login.
2. **Weather Module**: Fetches weather data from the API and displays it.
3. **Styling Module**: Applies CSS-based user interface improvements.
4. **Data Layer**: Manages user information in the MySQL database.

Users interact with the system through Thymeleaf-based HTML templates that communicate with backend controllers. On successful login, a user can search for a city's weather and receive dynamic weather data on the page. The result includes the temperature, humidity, wind speed, and weather condition description, along with an icon.

4. System Architecture

This project follows a Model-View-Controller (MVC) architecture. The structure of the application is as follows:

- **Model**: Represents the data layer using JPA Entities (User, WeatherResponse).
- **View**: Composed of HTML files styled with CSS and rendered using Thymeleaf.
- **Controller**: Java classes annotated with @Controller that handle user requests, process form data, call service classes, and return views.
- **Service**: Business logic for user registration, login validation, and API communication.
- **Repository**: Interface extending JPA Repository to perform database operations.

Each component plays a specific role:

- The **UserController** manages login and registration requests.
- The **WeatherController** processes city search requests and returns the weather page.
- The **UserService** interacts with the UserRepository and handles business logic.
- The **WeatherService** (if used) would be responsible for calling the OpenWeatherMap API.

By modularizing our system, we ensure maintainability, reusability, and extensibility.

5. Technologies Used

- **Java 17** – Core programming language.
- **Spring Boot** – Backend web framework for rapid development.
- **Thymeleaf** – Java template engine for rendering HTML.
- **Spring MVC** – To build web components and route user requests.
- **Spring Data JPA** – For database interaction.
- **MySQL** – Relational database to store user data.
- **OpenWeatherMap API** – Third-party API to get real-time weather data.
- **HTML5 & CSS3** – Used for building the front-end interface.
- **IntelliJ IDEA** – IDE for writing, running, and testing the application.
- **Git & GitHub** – For version control and code collaboration.
- **Lombok (optional)** – Used to reduce boilerplate code in entity and DTO classes.

6. Functional Requirements

- Users must be able to register with email, password, and nickname.
- Users must be able to log in using their credentials.
- After logging in, users can enter a city name to get its weather.
- The weather page should display:
    - City name
    - Country
    - Temperature in Celsius

    - Humidity percentage
    - Wind speed in m/s
    - Weather description
    - Weather icon
- Input validation and user feedback must be provided on forms.

7. Non-Functional Requirements

- **Usability**: Interface should be responsive and intuitive.
- **Security**: Passwords should be securely stored.
- **Performance**: Weather results should load quickly after input.
- **Scalability**: Code must support future additions like weather history.
- **Maintainability**: Follow clean code practices.

8. Database Design

The application uses a single table users in the database. Its schema is:

Field	Type	Constraints
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT
email	VARCHAR(255)	UNIQUE, NOT NULL
nickname	VARCHAR(255)	NOT NULL
password	VARCHAR(255)	NOT NULL (hashed)

Spring Data JPA maps this table to the User entity. All user-related queries are handled via UserRepository, which extends JpaRepository.

9. RESTful API Integration

We integrated the OpenWeatherMap REST API using the RestTemplate class provided by Spring. The process is as follows:

- The WeatherController receives the city name from the user input.
- A request URL is built: http://api.openweathermap.org/data/2.5/weather?q={city}&appid={apikey}&units=metric
- RestTemplate sends a GET request to this URL.
- The JSON response is mapped to a Java model (WeatherResponse) using Jackson.
- Data is extracted from the model and passed to the Thymeleaf template.

This demonstrates how our backend consumes a RESTful service and binds it directly into Java classes.

10. User Registration & Login Module

This module handles the following:

- Registration form with fields: email, nickname, password.
- Form validation using HTML5 and backend verification.
- User uniqueness check based on email.
- Password hashing using BCrypt before storing.
- Login form that checks the email and hashed password.
- Session creation (or redirection) upon successful login.

Security measures like hashing and validation were implemented to protect user data and prevent misuse. Error messages guide users in case of invalid credentials or registration issues.

11. Weather Data Fetching Module

The weather module is responsible for providing real-time information based on the city name input by the user. It works as follows:

- On the weather page, the user enters a city.
- The controller forms the API URL and makes a REST call.
- The JSON response is parsed and mapped to a Java object.
- Relevant weather data is extracted.
- The information is displayed dynamically in the view.

Data includes temperature, wind speed, humidity, country code, and weather description. An icon is also shown to reflect the weather visually.

12. Front-End Design

We used Thymeleaf for dynamic rendering and CSS for styling. The login and registration forms were centered using Flexbox. The weather page includes temperature, description, and icons.

CSS was kept modular:

- style.css – Used for forms.
- weather.css – Used for weather page design.

A responsive layout ensures usability on desktops and mobile devices.

13. Controller Layer

Controllers manage the routing logic. The two main controllers are:

- **UserController**: Handles /register, /login, and form submissions.
- **WeatherController**: Handles /weather requests and fetches weather data.

Each controller is annotated with @Controller, and methods use @GetMapping or @PostMapping. Model attributes are used to pass data to the Thymeleaf view.

14. Service Layer

Services abstract the business logic. UserService handles:

- Registering new users.
- Validating logins.
- Searching for existing emails.

Services use repositories to interact with the database, maintaining separation between business logic and data access.

15. Repository Layer

UserRepository extends JpaRepository and includes a method findByEmail(String email) to search users.

By using Spring Data JPA, we reduce boilerplate code and simplify database operations. The repository is injected into the service layer using @Autowired.

16. Security and Best Practices

We ensured several best practices:

- **BCrypt** was used to hash passwords.
- Input forms used required fields to reduce backend validation needs.
- Emails were verified for uniqueness.
- Sensitive data like API keys were hidden using application.properties.
- We avoided exposing passwords or internal details in views.

These measures make the app more secure and production-ready.

17. Testing and Validation

We manually tested the following scenarios:

- Registering with valid and invalid inputs.
- Logging in with correct and wrong passwords.
- Entering correct and invalid city names.
- API failure handling (city not found, no internet).

Each scenario resulted in appropriate user feedback.

18. Challenges and Solutions

Challenge	Solution
REST API rate limiting	Used caching if needed (future improvement)
Styling and responsiveness	Used Flexbox and modular CSS
Security for login system	Used BCrypt for password hashing

Input validation	Handled both in front-end and back-end
Mapping JSON to Java model	Used POJO structure and Jackson

19. Future Enhancements

- Add login session management.
- Enable logout functionality.
- Show historical weather data.
- Support multiple languages.
- Add weather forecast (5-day view).
- Make mobile-friendly responsive enhancements.
- Integrate OAuth login (e.g., Google, GitHub).

20. Team Contribution

- **Sahriar Ahmed**: Backend coding, API integration, service logic, testing.
- **Annay**: Frontend design, Thymeleaf view, styling, report writing.

Work was divided and committed on GitHub for version tracking.

21. Supervisor's Role

Abir Hussain supervised the project and provided guidance on architecture, best practices, and final review. Feedback helped refine both functionality and presentation.

22. Tools Used

- IntelliJ IDEA (Java IDE)
- Postman (API testing)
- GitHub (Version Control)
- MySQL Workbench (Database)
- Draw.io (Architecture Diagrams)
- ChatGPT (For help and documentation)

23. Screenshots

(Insert relevant screenshots of login, registration, weather result page, and project structure here)

24. Summary

The weather forecasting web app demonstrates how modern Java web applications can combine REST APIs, user management, and a responsive interface. Using Spring Boot ensures rapid development, while Thymeleaf and CSS provide frontend clarity. This full-stack project reflects both technical competence and good software practices.

25. References

- Spring Boot Documentation
- Thymeleaf
- OpenWeatherMap API
- Spring Data JPA
- Official Spring Guides