Dharmsinh Desai University, Nadiad
Faculty of Technology
Department of Computer Engineering
B. Tech. CE Semester – V

Subject: (CE – 520) Advanced Technologies
Project Title: Climate Smart Agriculture PlatForm

Prepared by: Name: Surati Ira P. ID: 22CEUSG067 Roll No: CE046 Computer Engineering, Dharmsinh Desai University	Prepared by: Name: Timbadiya Disha N ID: 22CEUOG056 Roll No: CE050 Computer Engineering, Dharmsinh Desai University

Guided by:
Prof Prashant Jadav

Dharmsinh Desai University

Faculty of Technology, College Road, Nadiad – 387001, Gujarat

CERTIFICATE

This is to certify that the term work carried out in the subject of (CE520) Advanced Technologies and submitted is the bonafide work of **Ira Surati & Disha Timbadiya** Roll No.: **CE046, CE050** Identity **No.: 22CEUSG067, 22CEUOG056** of B.Tech. **Semester V** in the branch of Computer Engineering during the academic year **2024-2025**.

Prof. P. M. Jadav	Dr. C. K. Bhensdadia
Associate Professor	Head, CE Department

Contents

1.Abstract

The Climate Smart Agriculture Platform is designed to provide farmers with precise crop predictions, weather insights, and management tools to optimize their farming processes. The platform integrates various technologies such as MongoDB, React, Node.js, and external APIs (OpenWeatherMap and OpenCage) to ensure that climate data and predictions are seamlessly delivered to users. This project aims to aid farmers in making informed decisions about planting schedules, crop selection, price prediction and soil management by leveraging climate data and intelligent algorithms.

2.Introduction

The Climate Smart Agriculture Platform seeks to integrate smart farming practices with modern technologies to provide a digital solution for sustainable agriculture. It combines crop prediction algorithms, Price prediction, real-time weather data, and detailed farm management features to help farmers improve yield and reduce the risks posed by climate variability.

Technology/Platform/Tools Used

- Frontend: React, HTML5, CSS3
- Backend: Node.js, Express.js, MongoDB
- APIs: OpenCage (Geocoding), OpenWeatherMap (Weather Data), Custom Crop Prediction Model, Price prediction
- Version Control: Git, GitHub
- Database: MongoDB for storing user, weather, and farm data

3.System Analysis

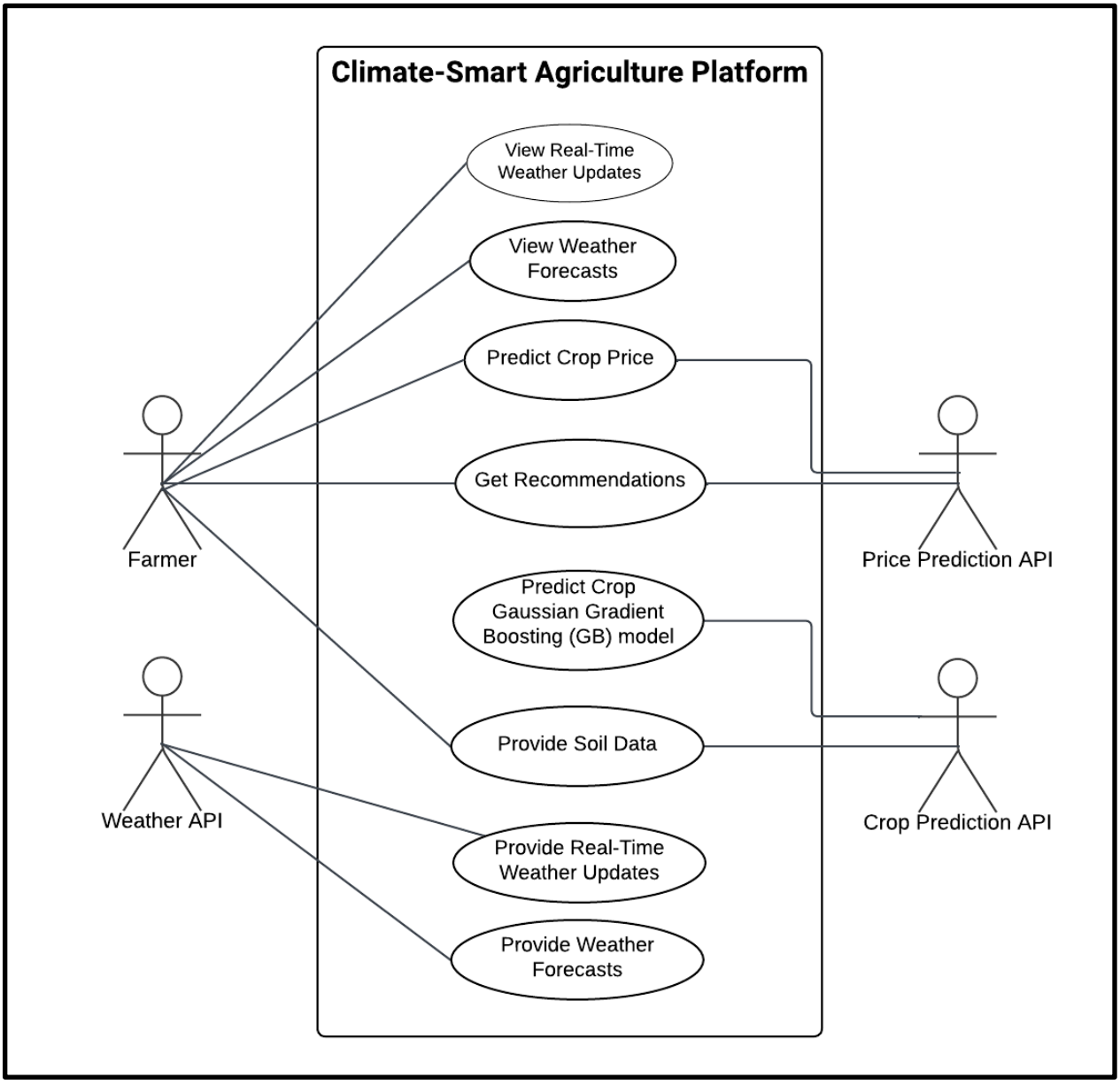

Figure 3.1
[Use Case Diagram]

Sequence Diagrams:

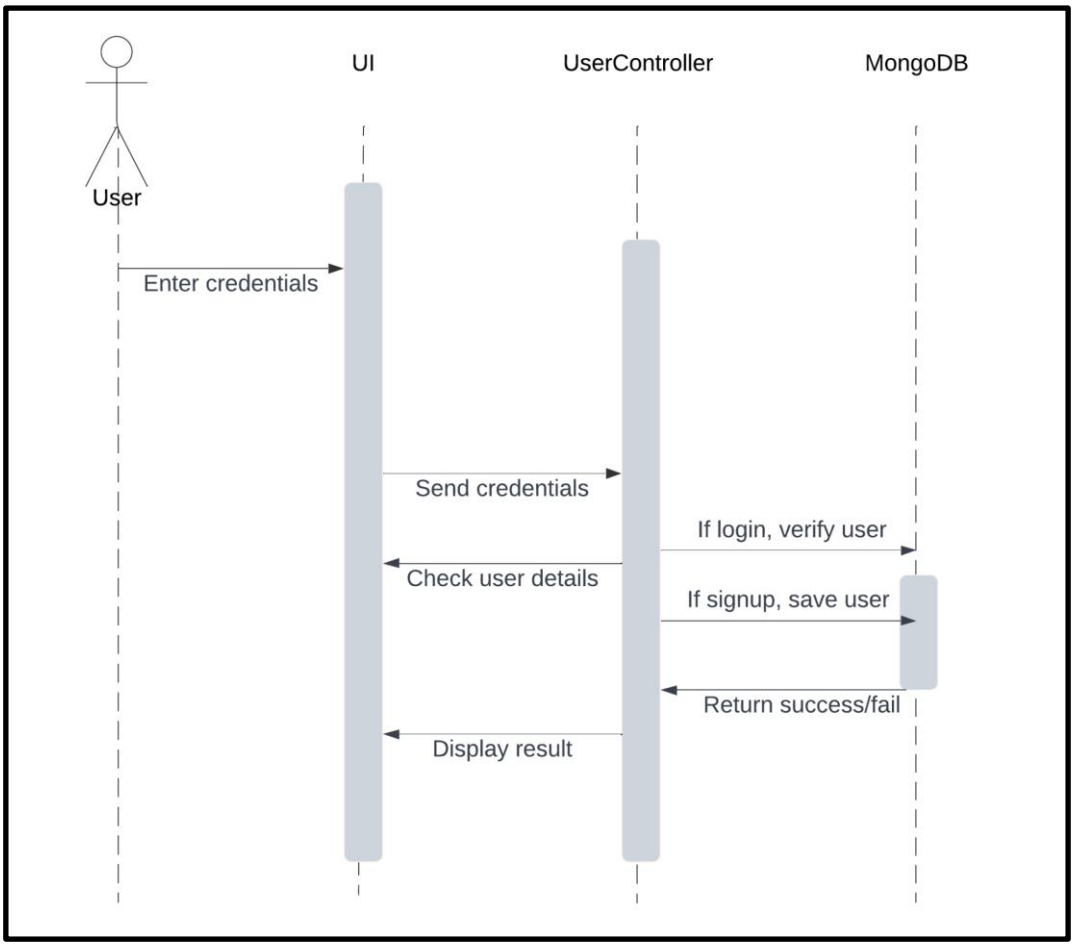

Figure 3.2

[User Management (Signup/Login)]

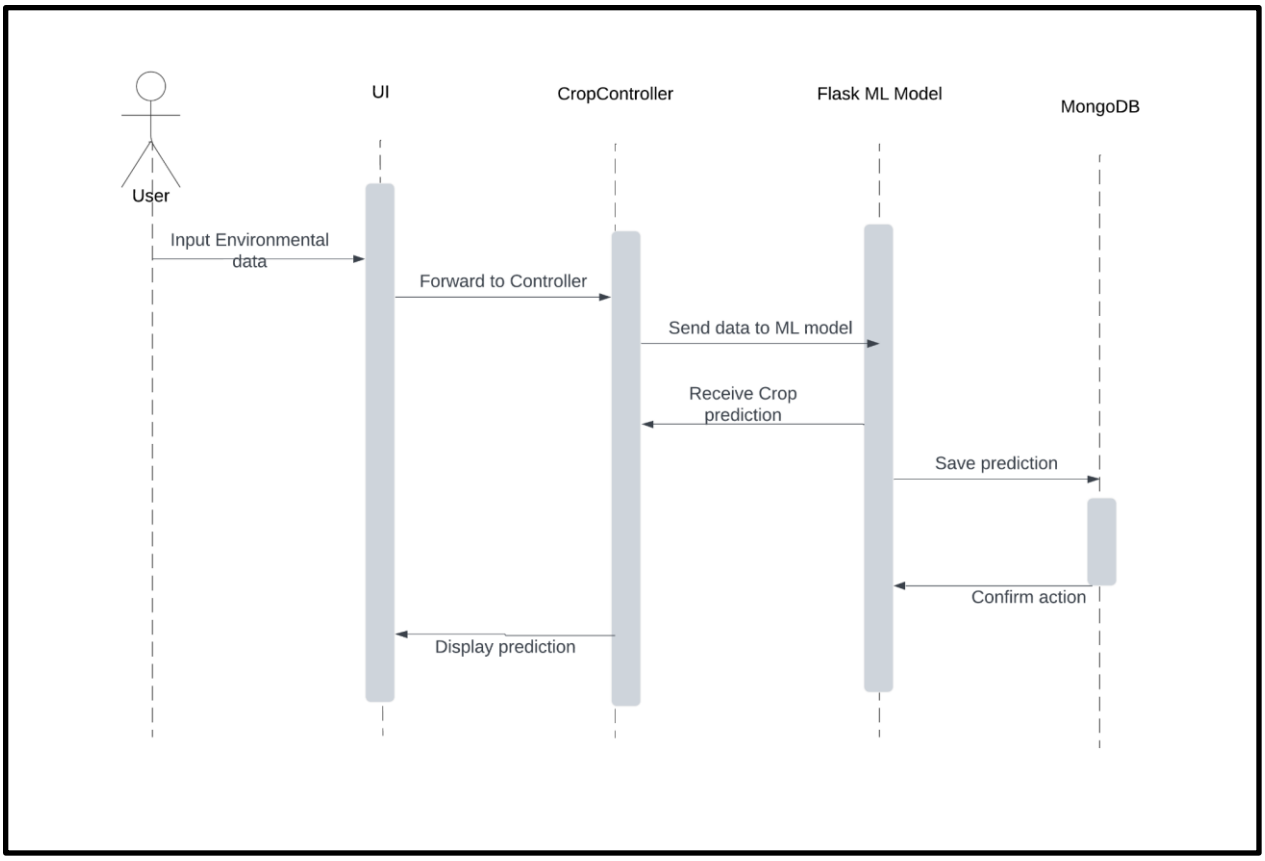

Figure 3.3
[Crop Prediction]

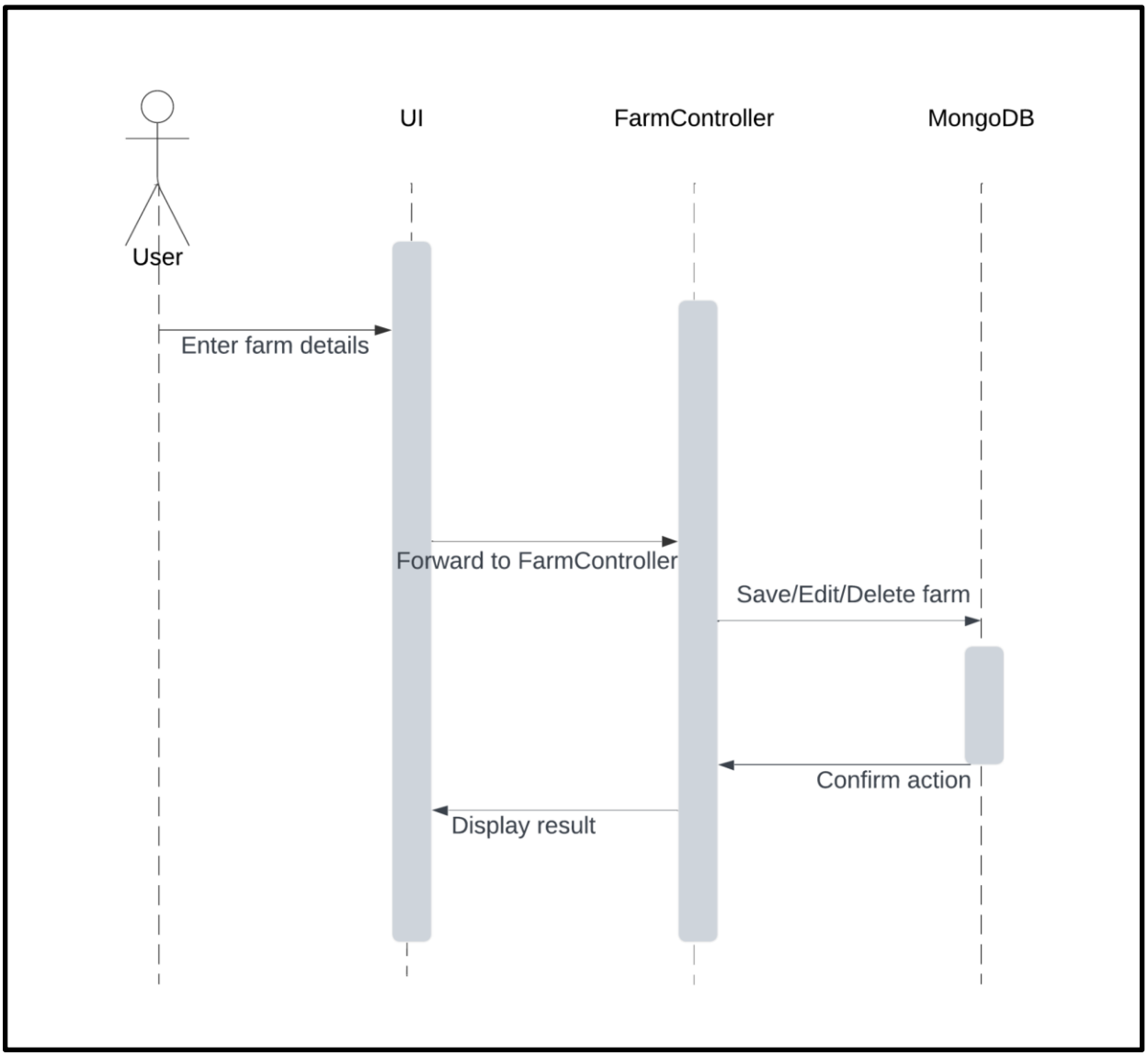

Figure 3.4

[Farm Management]

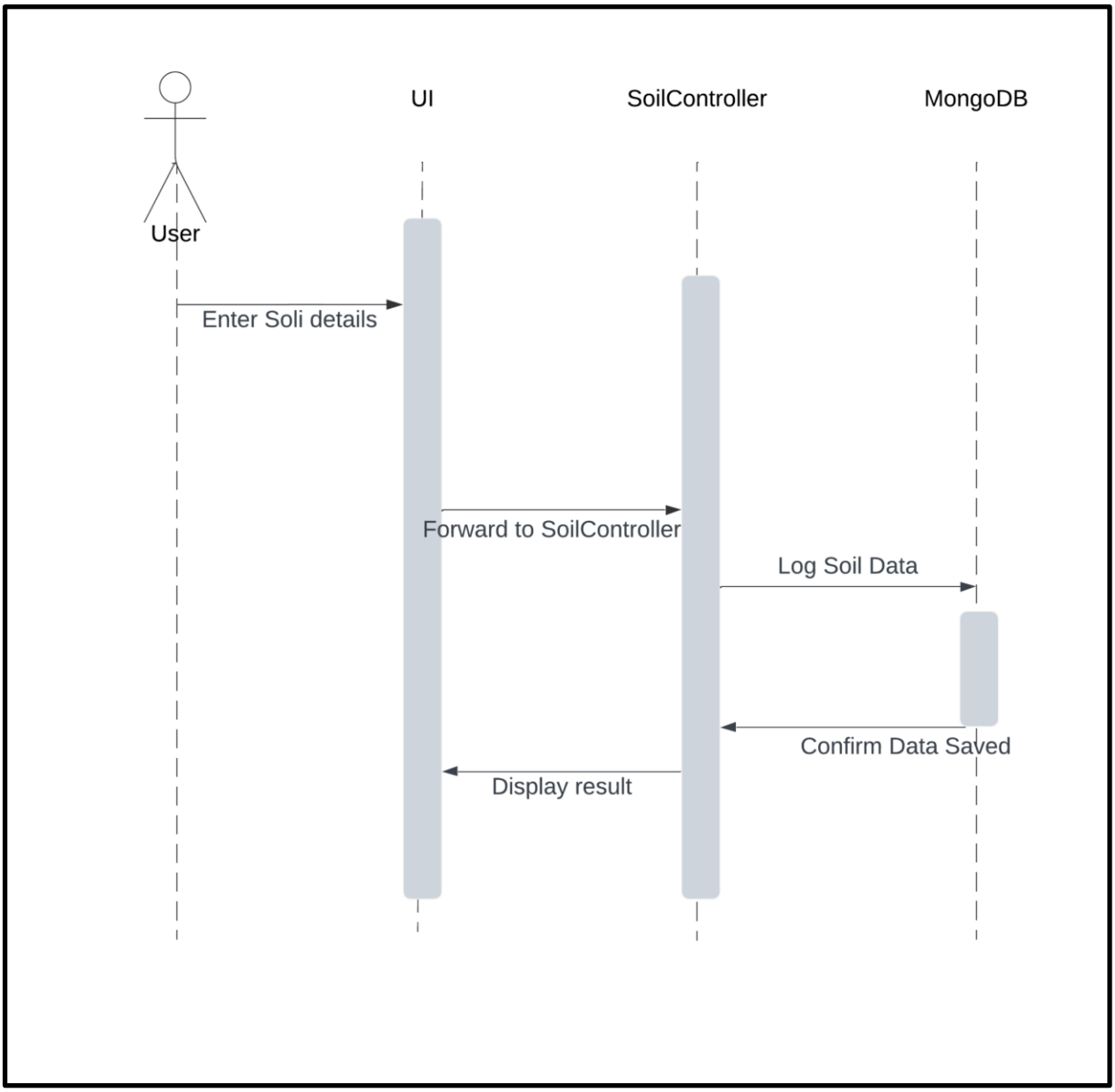

Figure 3.5

[Soil Management]

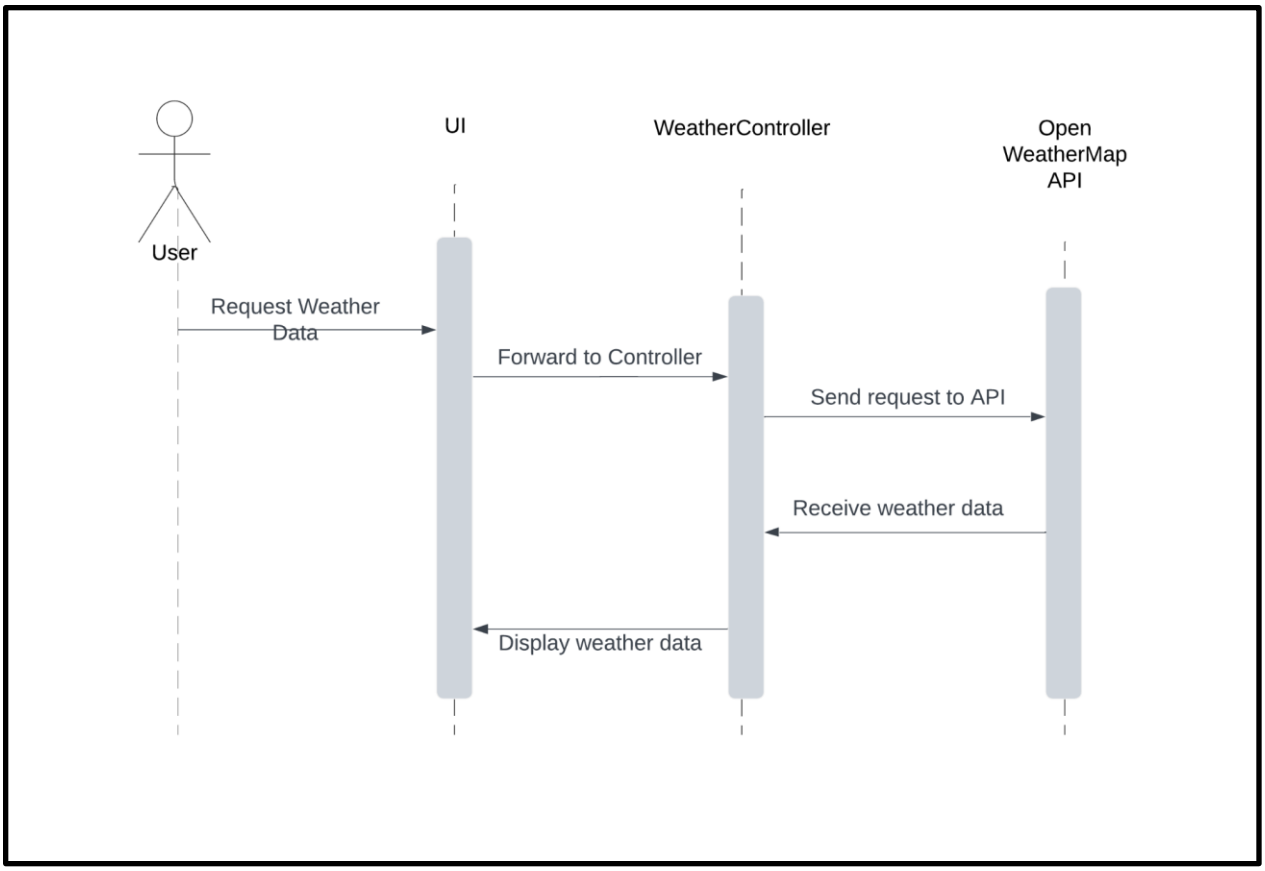

Figure 3.6

[Weather Data]

4.Software Requirement Specifications (SRS)

Climate-Smart Agriculture Platform

4.1. Introduction

4.1.1 Purpose

The purpose of this document is to provide a detailed Software Requirements Specification (SRS) for the Weather-Integrated Farm Management application. This application aims to assist farmers in making informed decisions by providing real-time weather data, forecasts, and actionable insights to optimize agricultural practices.

4.1.2 Scope

The Weather-Integrated Farm Management Dashboard is a web application developed using the MERN stack that will provide farmers with a dashboard that integrates **real-time weather data, historical data analysis, and crop management recommendations, price prediction**. The application will include features such as **weather widgets, planting calendars.**

4.1.3 Definitions, Acronyms, and Abbreviations

- **API**: Application Programming Interface
- **CRUD**: Create, Read, Update, Delete
- **SRS**: Software Requirements Specification
- **MERN**: MongoDB, Express.js, React.js, Node.js
- **UI**: User Interface
- **UX**: User Experience

4.1.4 References

- OpenWeatherMap API documentation
- MongoDB documentation
- Chart.js documentation

4.1.5 Overview

This SRS document includes functional and non-functional requirements, system architecture, data design, and user interface design for the Weather-Integrated Farm Management application.

4.2. General Description

4.2.1 Product Perspective

The Weather-Integrated Farm Management application is a standalone system designed to assist farmers in managing their crops efficiently by leveraging real-time weather data.

4.2.2 Product Functions

- Display real-time weather data
- Offer recommendations for planting, irrigation, and harvesting
- User-friendly dashboard

4.2.3 User Characteristics

- **Farmers**: Primary users who use the dashboard for weather updates, recommendations.
- **Agricultural Experts**: Users who may analyze data and provide additional insights
- **System Administrators**: Users who will manage the application, including user accounts & data and Monitor Weather API.

4.2.4 Operating Environment

- Web-based application accessible via modern browsers (Chrome, Firefox, Safari)
- Backend services hosted on cloud infrastructure
- Integration with third-party weather APIs and ML models

4.2.5 Constraints

- Reliable and continuous access to weather APIs
- Desktop compatibility
- Data privacy and security

4.2.6 Assumptions and Dependencies

- Reliable internet connection for real-time data updates
- Weather APIs provide accurate and timely data

4.3 Specific Requirements

4.3.1 Functional Requirements

4.3.1.1 Real-Time Weather Updates

Input: PinCode of the area where farm belongs

Process:

- The system retrieves real-time weather data (temperature, humidity, wind speed, precipitation) for the specified farm location from an external weather API (e.g., OpenWeatherMap).

Output:

- The system displays current weather conditions including:
    - Temperature
    - Humidity
    - Wind Speed
    - Precipitation

4.3.1.2 Planting Calendar

Input:

- **Farmer's Soil Data**: Soil conditions such as nutrient levels and moisture content.
- **Available Resources**: Tools, equipment, and other resources available to the farmer.
- **Location**: The location of the farm, which influences planting schedules based on regional weather patterns.

Process:

- The system analyzes historical weather data and current forecasts to determine optimal planting dates.
- The system manages planting schedules and allows users to adjust them manually as needed.

Output:

- **Suggested Planting Dates**: Optimal planting dates based on soil data, location, and weather forecasts.
- **Calendar Interface**: The user is able to view, create, and manage planting schedules.

4.3.1.3 Crop Prediction Recommendations

Input:

- Farmer's Soil Data, Weather Conditions (temperature, humidity, etc.), and Soil Nutrient Levels (N, P, K).

Process:

- The system uses a machine learning model to predict the most suitable crop based on soil data, available resources, and weather conditions.

Output:

- **Crop Recommendation**: Predicted crop that would thrive best under current conditions.

4.3.1.4 User Registration and Login

Input:

- **User Data:** User's registration details (username, password, email, etc.).

Process:

- The system registers new users by saving their information in the database.
- The system verifies user credentials during login.

Output:

- **User Registration:** Successful registration with required fields (username, password, email).
- **User Login:** Successful login if credentials match stored records.
- **Redirection:** The system redirects users to appropriate pages based on their authentication status:
    - After successful registration: Redirect to the login page.
    - After successful login: Redirect to the dashboard.

4.3.1.5 Navigation and User Interface

Input:

- **User Authentication State**: Determines whether the user is logged in or not.

Process:

- The system adjusts the navigation menu based on the user's authentication status.

Output:

- **Logged-in Users**: Show options including **"Home", "Profile", "Predict Crop", "Weather Data", "Farm Details", "About Us", and "LogOut"**.
- **Non-logged-in Users**: Show options including **"Home", "Login", "Register", and "About Us"**.

4.3.1.6 Crop Price Prediction

Input:

- **Predicted Crop**: The crop that the farmer wishes to predict the price for.

- **Market Data**: Historical crop price data, current demand.

Process:

- The system analyzes historical price trends and current market demand to predict the price of the selected crop using the ML model.

Output:

- **Predicted Crop Price**: Estimated market price for the predicted crop.

4.3.2 Non-Functional Requirements

4.3.2.1 Performance

- The system shall handle multiple users and farms simultaneously with efficient data retrieval and processing.
- The system shall provide real-time updates with minimal latency.

4.3.2.2 Scalability

- The system shall be scalable to accommodate an increasing number of users and data points over time.

4.3.2.3 Security

- The system shall ensure data privacy and implement security measures such as encryption and secure access controls.
- The system shall comply with relevant data protection regulations.

4.3.2.4 Usability

- The system shall provide a user-friendly interface with easy navigation and accessibility.
- The system shall be responsive and compatible with both desktop and mobile devices.

4.4. System Architecture

4.4.1 Frontend

- **Framework**: React.js
- **Components**: Dashboard interface, weather widgets, forecast graphs, planting calendar, alerts, notifications.

4.4.2 Backend

- **Framework**: Node.js (Express)

- **Components**: API integration with weather services, data processing, algorithm development, user authentication and authorization
- **Price Prediction Module:** Integrates market data and crop-specific information to predict future crop prices using a machine learning model.

4.4.3 Database

- **Database**: MongoDB
- **Components**: Data storage for user preferences, farm locations, historical weather data, personalized recommendations, crop disease detection results

4.5. Data Design

4.5.1 Database Schema

- Users(_id, username, password, email, notificationFrequency, createdAt, updatedAt)
- Farms(_id, user_id, location, crop_type, planting_schedule, soil_type, irrigation_system, size)
- WeatherData(_id, farm_id, temperature, humidity, wind_speed, precipitation, timestamp)

4.6. User Interface Design

4.6.1 Dashboard

- **Weather Widgets**: Display current weather conditions (temperature, humidity, wind speed, precipitation)
- **Planting Calendar**: Show optimal planting dates and schedules
- **Weather Data Display**: The dashboard will present real-time weather data (temperature, rain) for the user's selected location using OpenWeatherMap API.
- **Crop Prediction System**: Users can input various parameters (e.g., soil type, crop type, planting schedule) and receive crop recommendations based on predictive analytics.
- **Farm Details Management**: Users can add and manage their farm details, including location, crop types, irrigation systems, and sizes.
- **Price Prediction Interface:** After a user inputs the crop parameters, the system will provide a real-time prediction of not only suitable crops but also expected market prices. This data will be displayed in a dedicated "Price Prediction" section, enabling users to make informed decisions regarding sales.

4.7. Implementation Considerations

4.7.1 Scalability

- Use cloud services for hosting and scaling the application.
- Implement load balancing to distribute traffic efficiently.

4.7.2 Mobile Compatibility

- Design the dashboard to be responsive and ensure compatibility with various screen sizes and devices.

4.7.3 API Reliability

- Select weather APIs with high reliability and accuracy.
- Implement fallback mechanisms in case of API failures or downtime.

4.8. Conclusion

The Weather-Integrated Farm Management application aims to provide farmers with a comprehensive tool to manage their crops effectively by leveraging real-time weather data and forecasts. By implementing the features and requirements outlined in this SRS, the application will support informed decision-making, optimize agricultural practices, and improve productivity.

5.Database Design

Farm Schema

Purpose: Stores detailed information about the farms managed by users.

```
const FarmSchema = new mongoose.Schema({
  user_id: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  location: { type: String, required: true },
  crop_type: { type: String, required: true },
  planting_schedule: { type: Date, required: true },
  soil_type: { type: String },
  irrigation_system: { type: String },
  size: { type: Number, required: true },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});
```

Soil Schema

Purpose: Tracks soil parameters and predicted crop types for a particular farm or plot.

```
const SoilSchema = new mongoose.Schema({
  farm_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Farm', required: true },
  N: { type: Number, required: true },
  P: { type: Number, required: true },
  K: { type: Number, required: true },
  temperature: { type: Number, required: true },
  humidity: { type: Number, required: true },
  ph: { type: Number, required: true },
  rainfall: { type: Number, required: true },
  predictedCrop: { type: String }
});
```

User Schema

Purpose: Stores user credentials and preferences.

```
const UserSchema = new mongoose.Schema({
  username: { type: String, unique: true, required: true },
  password: { type: String, required: true },
```

```
  email: { type: String, unique: true, required: true },
  preferences: {
    notification_frequency: { type: String, default: 'daily' },
    preferred_units: { type: String, default: 'metric' }
  }
});
```

Weather Schema

Purpose: Holds real-time weather data related to farms and their locations.

```
const WeatherSchema = new mongoose.Schema({
  farm_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Farm', required: true },
  zip: { type: String },
  coord: {
    lon: { type: Number },
    lat: { type: Number }
  },
  weather: [{
    id: { type: Number },
    main: { type: String },
    description: { type: String },
    icon: { type: String }
  }],
  main: {
    temperature: { type: Number },
    humidity: { type: Number },
    pressure: { type: Number }
  },
  wind: {
    speed: { type: Number },
    deg: { type: Number }
  },
  clouds: { coverage: { type: Number } },
  sys: {
    country: { type: String },
    sunrise: { type: Date },
    sunset: { type: Date }
  },
  timezone: { type: Number },
  date: { type: Date, default: Date.now }
});
```

6.Implementation Detail

Software / Framework / Libraries / Database Versions:

Dependencies:

- @mui/icons-material: ^6.1.1
- axios: ^1.7.7
- bcrypt: ^5.1.1
- body-parser: ^1.20.2
- bootstrap: ^5.3.3
- cors: ^2.8.5
- dotenv: ^16.4.5
- ejs: ^3.1.10
- express: ^4.19.2
- express-validator: ^7.2.0
- mongoose: ^8.5.4
- morgan: ^1.10.0
- nodemon: ^3.1.4
- react: ^18.3.1
- react-bootstrap: ^2.10.5
- react-dom: ^18.3.1
- react-redux: ^9.1.2
- react-router-dom: ^6.26.2
- react-scripts: ^5.0.1
- redux: ^5.0.1
- redux-thunk: ^3.1.0
- sass: ^1.79.4

Dev Dependencies:

- @babel/plugin-proposal-private-property-in-object: ^7.21.11

1. Farm Module

- **Purpose**: Manages and stores data related to farms.
- **Key Features**:
    - **Register a New Farm**: Allows users to create new farm entries.
    - **Link Farm Data to Users**: Associates each farm with a specific user for personalized management.
    - **Edit and Delete Farm Details**: Provides functionality for users to update or remove farm records as needed.

- **Prototype:**

  ```
  // Register a new farm
  void registerFarm(String userID, String location, String cropType, Date
  plantingSchedule);

  // Edit an existing farm
  void editFarm(String farmID, String location, String cropType, Date plantingSchedule);
  ```

---

2. Soil Module

- **Purpose**: Handles all information related to soil conditions and nutrient levels.
- **Key Features**:
  - **Log Soil Nutrient Levels**: Records data for key nutrients such as Sodium (Na), Potassium (K), and pH levels.
  - **Analyze Soil Conditions**: Tracks changes in soil health over time to support farming decisions.
  - **Monitor Moisture Content**: Collects data on soil moisture.
- **Prototype:**

  ```
  // Log soil nutrient levels
  void logSoilNutrients(String farmID, int N, int P, int K, float pH, float moisture);

  // Get soil conditions
  SoilData getSoilConditions(String farmID);
  ```

---

3. User Module

- **Purpose**: Manages user authentication and account settings.
- **Key Features**:
  - **User Authentication**: Facilitates sign-up and login processes for users.
  - **Role-Based Access Control**: Implements different access levels for farm owners, admins, and guests.
  - **Account Management**: Allows users to update their profile and settings.
- **Prototype:**

  ```
  // Register a new user
  void signup(String username, String password, String email, String
  notificationFrequency, String preferredUnits);
  ```

```
// Login a user
User login(String email, String password);

// Get user data by ID
User getUserData(String userID);

// Update user data by ID
void updateUserData(String userID, String username, String email, String
notificationFrequency, String preferredUnits);
```

---

4. Weather Module

- **Purpose**: Provides real-time weather data relevant to farming operations.
- **Key Features**:
    - **Fetch Real-Time Weather Updates**: Retrieves current weather conditions for the farm's location.
- **Prototype:**

```
// Fetch weather data based on coordinates
WeatherData getWeatherData(String zipCode, String tempMetric);

// Get coordinates from pincode
Coordinates getCoordinatesFromPincode(String pincode);
```

---

5. Crop Prediction Module

- **Purpose**: Utilizes machine learning to recommend suitable crops based on environmental conditions.
- **Key Features**:
    - **Input Acceptance**: Gathers user input such as nutrient levels, weather conditions, and soil data.
    - **Returns Recommended Crops**: Provides a list of crops that would thrive under the given conditions.
- **Prototype:**

```
// Predict crop based on environmental data
CropPrediction predictCrop(int N, int P, int K, float temperature, float humidity, float ph,
float rainfall, String userID);
```

6. Machine Learning Prediction Module (Flask-based)

- **Purpose**: Implements machine learning to predict suitable crops based on various agricultural factors.
- **Key Features**:
    - **Load Pre-Trained Models**: Utilizes a saved machine learning model and scaler for predictions.
    - **Environmental Data Processing**: Accepts and processes input data from users to make predictions.
    - **Return Predictions**: Provides the predicted crop and associated confidence scores.
- **Key Routes**:
    - **POST /predict**: Endpoint for users to submit data and receive crop predictions based on their inputs.
- **Prototype:**

  // Make crop predictions using ML model : Python
  CropPrediction predictCropFromML(float N, float P, float K, float temperature, float humidity, float ph, float rainfall);

7.Price Prediction Module

- **Purpose**: Implements machine learning algorithms to predict the future market price for crops based on historical market data and predicted crop information.
- ***Key Features:***
    - Fetches crop names from the crop prediction module.
    - Processes historical market data to provide a price estimate for the predicted crops.
    - Allows users to view predicted prices on the dashboard to help with their sales planning.
- **Prototype:**

  // Predict crop price based on crop name
  float predictPrice(String cropName);

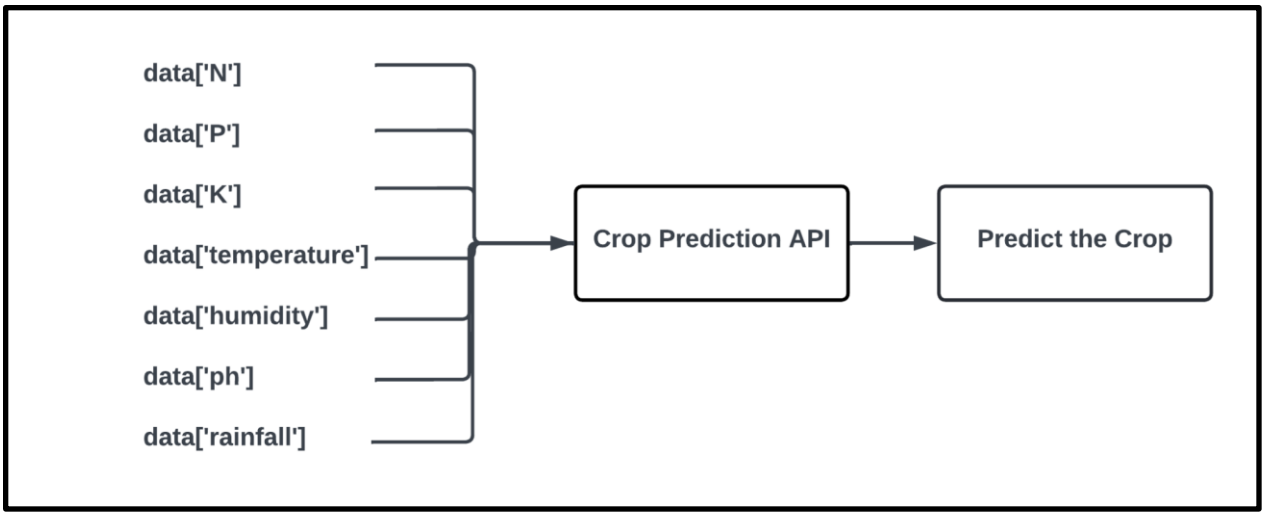

Figure 5.1

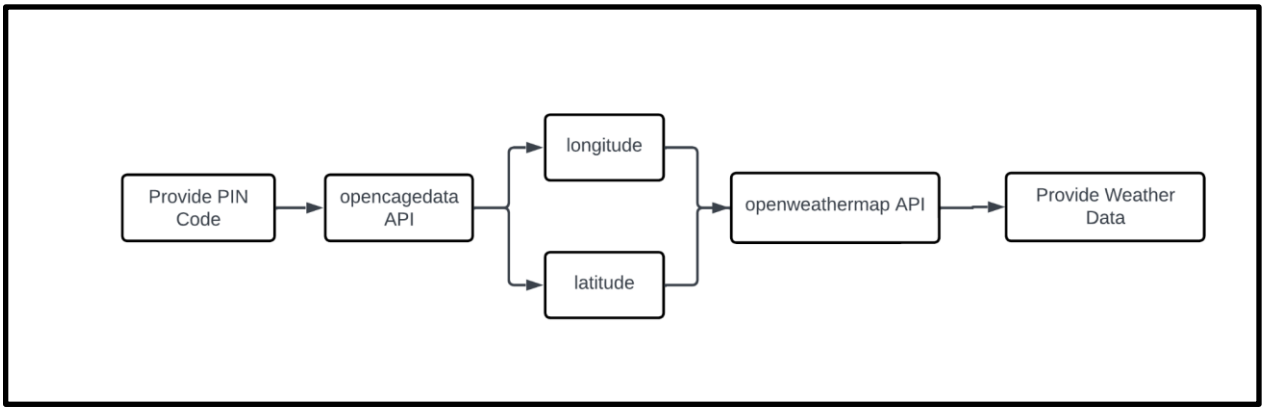

Figure 5.2

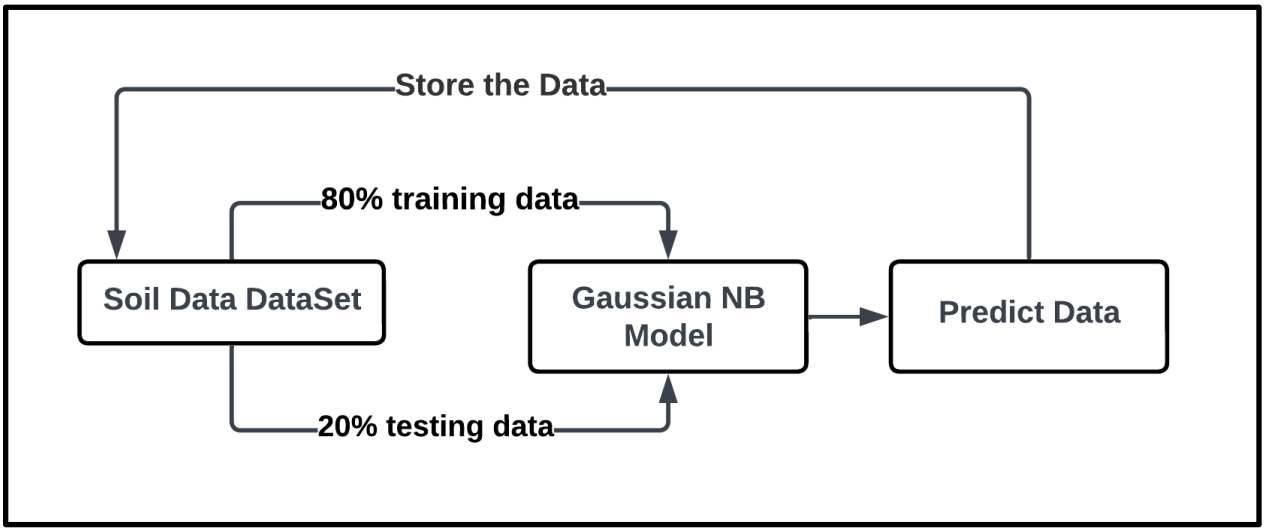

Figure 5.3

7.Testing

Testing Method

- **Methodology**: **Unit Testing** and **Integration Testing**

    - **Unit Testing**: Focuses on testing individual components or functions of the application in isolation to ensure each unit performs as expected.

    - **Integration Testing**: Tests the interactions between different modules of the application to ensure they work together correctly.

Test Cases Used to Test the System

Table 6.1

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Result	Actual Result	Status
1	Verify that a new user can sign up with valid credentials.	1. Go to the signup page. 2. Enter valid email, password, and username. 3. Click submit.	Email, password, username	User should be created successfully with a 201 status code.	User created successfully	Pass
2	Verify that users can log in with valid credentials.	1. Go to the login page. 2. Enter valid email and password. 3. Click submit.	Email, password	User should be authenticated and receive a session token with a 200 status code.	User authenticated successfully	Pass
3	Check if the user data can be retrieved based on user ID.	1. Send a GET request with a valid user ID.	User ID	User data should be retrieved with a 200 status code.	User data retrieved successfully	Pass

4	Verify that a user can update their profile information.	1. Go to the profile page. 2. Enter updated details. 3. Click submit.	Updated user details	User data should be updated with a 200 status code.	User data updated successfully	Pass
5	Ensure that a new farm can be created.	1. Go to the farm creation page. 2. Enter valid farm details (user ID, location, crop type, etc.). 3. Click submit.	User ID, farm details	Farm should be created with a 201 status code.	Farm created successfully	Pass
6	Test if a farm can be retrieved using its ID.	1. Send a GET request with a valid farm ID.	Farm ID	Farm details should be retrieved with a 200 status code.	Farm details retrieved successfully	Pass
7	Validate that farm details can be updated.	1. Go to the edit farm page. 2. Enter updated farm details. 3. Click submit.	Updated farm details	Farm details should be updated with a 200 status code.	Farm details updated successfully	Pass
8	Check if a farm can be deleted using its ID.	1. Send DELETE request with valid farm ID.	Farm ID	Farm should be deleted with a 204 status code.	Farm deleted successfully	Pass
9	Verify that soil nutrient levels can be logged.	1. Go to the log nutrient levels page. 2. Enter valid soil nutrient data. 3. Click submit.	Soil nutrient data	Soil nutrient data should be logged with a 201 status code.	Soil nutrient data logged successfully	Pass
10	Check if soil data can be retrieved based on farm ID.	1. Send a GET request with a valid farm ID.	Farm ID	Soil data should be retrieved with a 200 status code.	Soil data retrieved successfully	Pass
11	Ensure that weather data can be fetched from an external API and stored.	1. Send a request to the weather API with a valid zip code. 2. Store response in MongoDB.	Zip code	Weather data should be fetched and saved with a 201 status code.	Weather data fetched and saved successfully	Pass
12	Check if saved weather data can be retrieved by zip code.	1. Send a GET request with a valid zip code.	Zip code	Weather data should be retrieved with a 200 status	Weather data retrieved successfully	Pass

				code.		
13	Verify that the system can predict the most suitable crop based on environmental data.	1. Send environmental data for prediction. 2. Receive crop prediction.	N, P, K, temperature, humidity, rainfall	Predicted crop should be returned with a confidence score.	Crop predicted successfully	Pass
14	Verify that the system can predict the future price of a crop.	1. Send predicted crop name for price prediction. 2. Receive price prediction.	Predicted crop name	Predicted price should be returned.	Price predicted successfully	Pass

8.Screenshots

Home page:

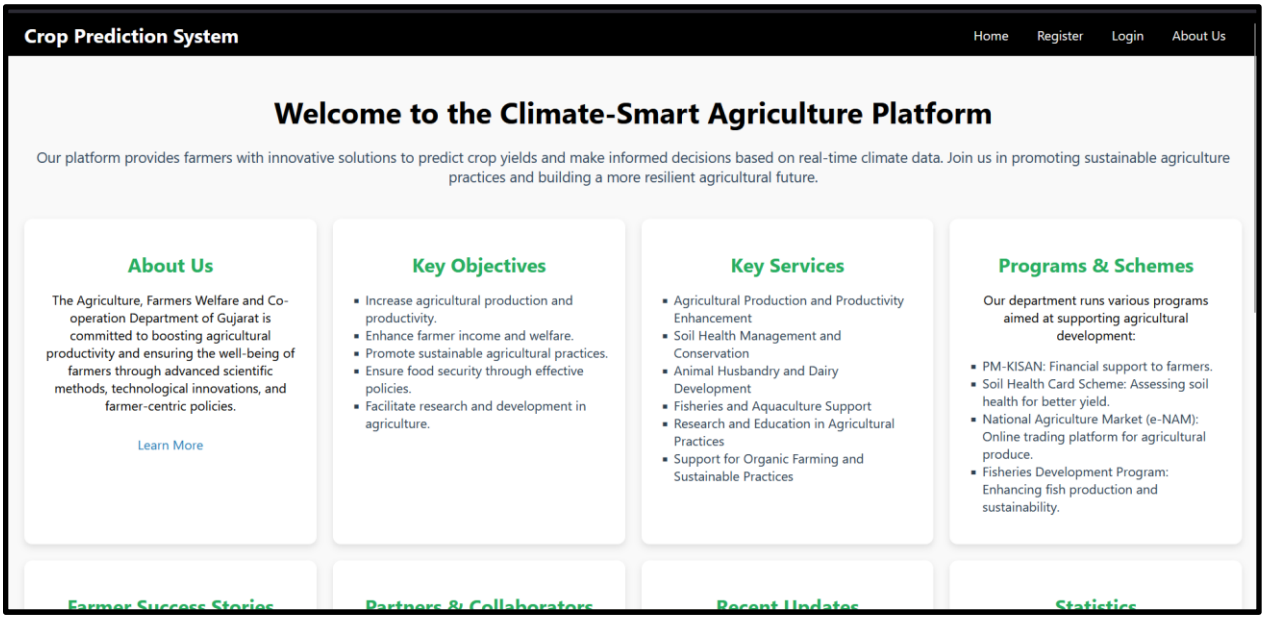

Figure 7.1
[Home Page]

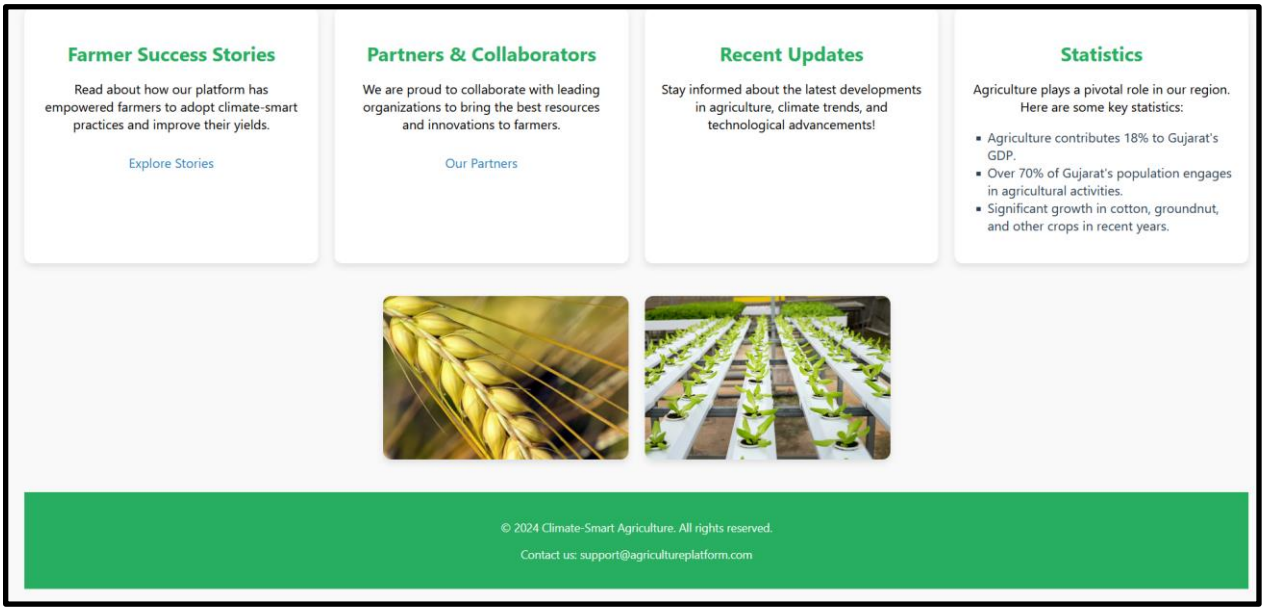

Figure 7.2
[Home Page]

Predict Crop

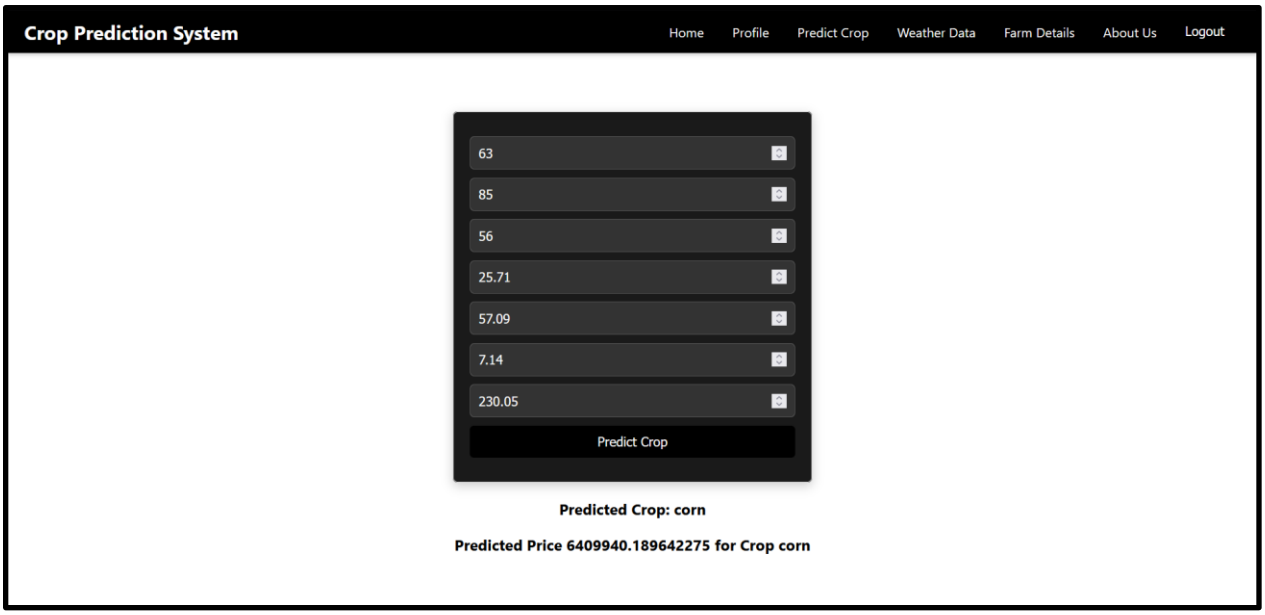

Figure 7.3
[Predict Crop Page]

Weather Data:

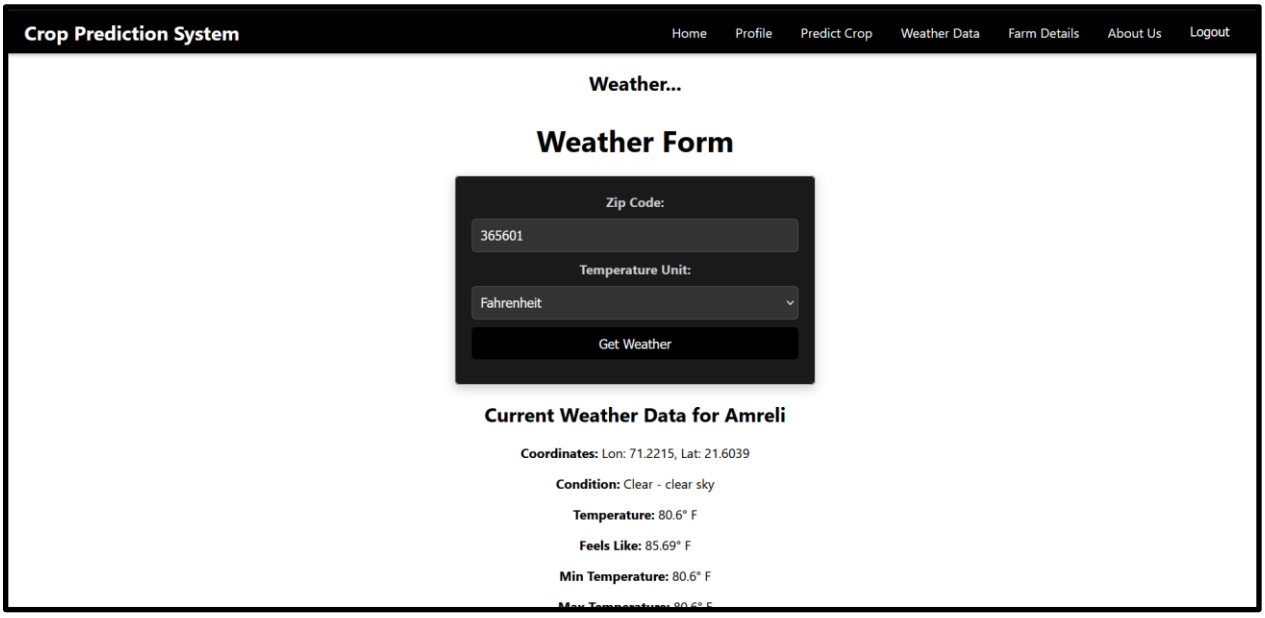

Figure 7.4
[Weather Data Page]

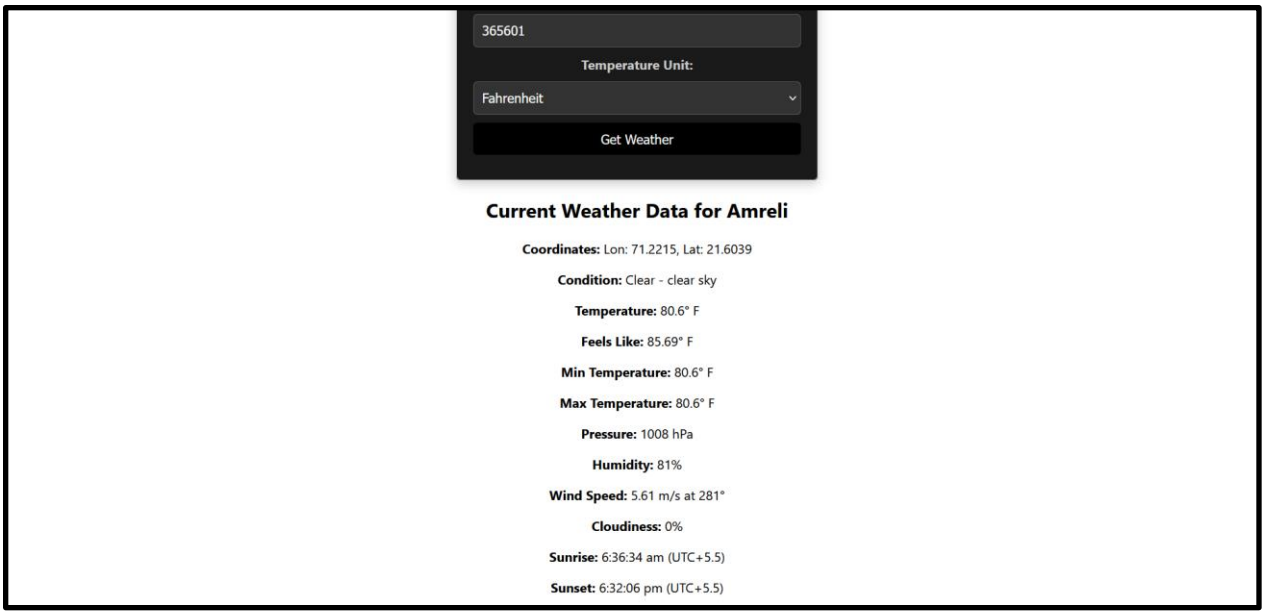

Figure 7.5
[Weather Data Page]

Farm Details

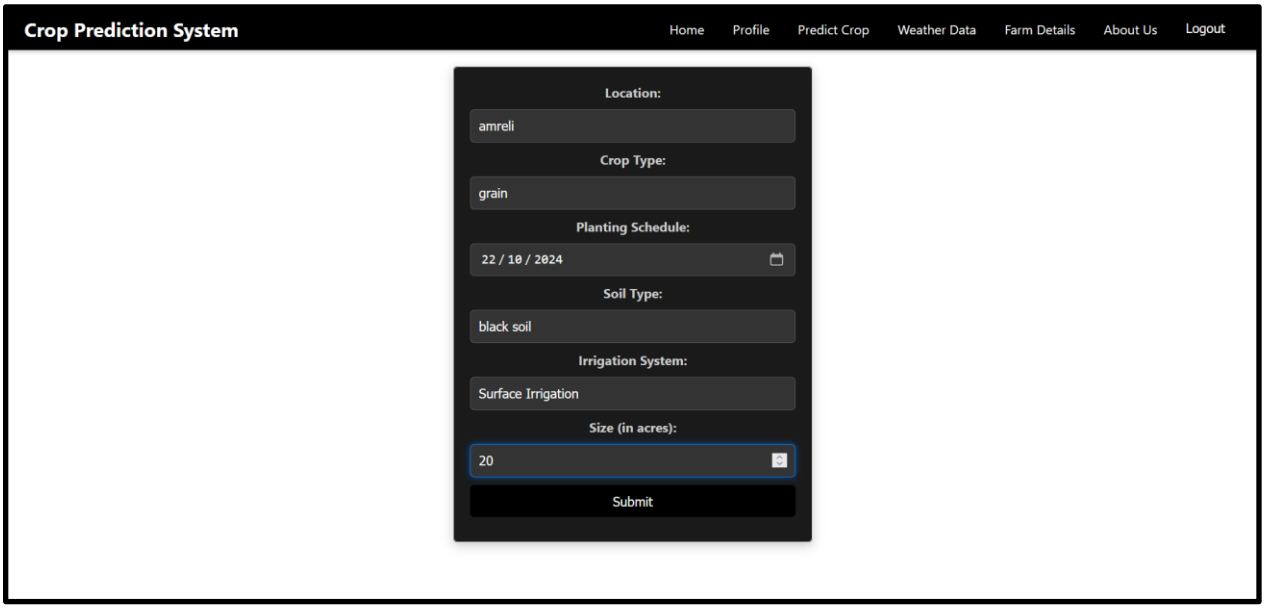

Figure 7.6
[Farm Detail Page]

About Us

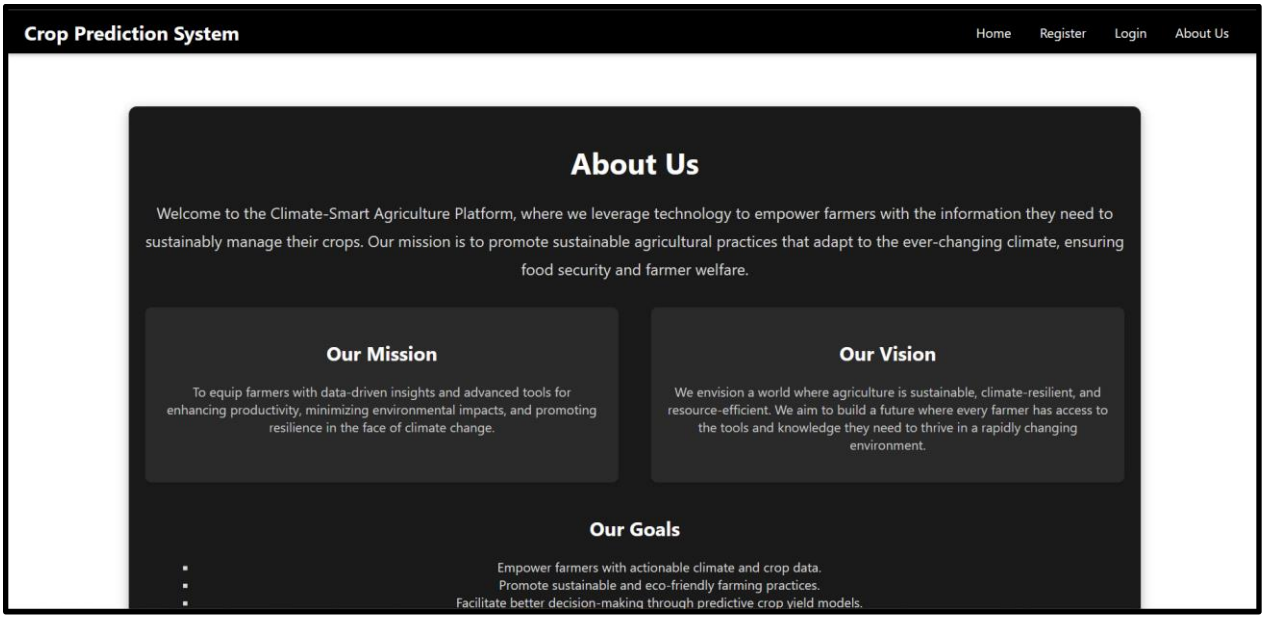

Figure 7.7
[About Us Page]

9.Conclusion:

In this project, the following functionalities have been successfully implemented:

Real-Time Weather Data Integration: The system fetches real-time weather updates for farm locations using the OpenWeatherMap and OpenCage Geocoding APIs. These updates provide essential data, including temperature, humidity, wind speed, and precipitation, helping users make timely and informed farming decisions.

Crop Prediction: A crop prediction module was developed, using machine learning models to recommend suitable crops. The module considers key environmental factors such as soil nutrient levels (Nitrogen, Phosphorus, Potassium), temperature, humidity, soil pH, and rainfall. By analyzing these parameters, the system predicts the crops that would thrive best under the current conditions.

Crop Price Prediction: A price prediction module was developed to estimate future crop prices based on the predicted crop and market data, enabling better financial planning.

Farm Management: The platform offers farm management functionality, enabling users to create, edit, and delete farm details. Users can manage various farm attributes, including the farm's location, crop types, planting schedules, soil types, and irrigation systems.

User Authentication: A user authentication system has been implemented, allowing users to register, log in, and access personalized data. This system ensures that each user can manage farm-related details and retrieve weather and prediction data specific to their farms and user-specific functionality.

10.Limitation and Future Extension

Limitations of the Project

1. **Data Dependency**:
    a. The accuracy of the crop prediction model & Price Prediction model heavily relies on the quality and availability of input data such as soil nutrients, weather conditions, and historical crop data. Inaccurate or incomplete data can lead to unreliable predictions.
2. **Weather API Limitations**:
    a. The project depends on external weather APIs for real-time data. Any downtime, rate limits, or changes in API endpoints can affect the system's functionality and data availability.
3. **Scalability**:
    a. The current architecture may face challenges in handling large datasets or high user traffic, especially if many farms and users are added to the system. Performance optimization may be necessary to scale effectively.
4. **Machine Learning Model Limitations**:
    a. The Gaussian Naive Bayes model may not capture complex relationships in the data.
5. **Notifications and Alerts**:
    a. The system does not currently include features for sending notifications or alerts to users based on weather changes, crop health, or soil conditions. Implementing a notification system could enhance user experience and engagement.
6. **7-Day Weather History**:
    a. The project lacks the capability to store and retrieve historical weather data for the past 7 days. Implementing this feature would enable users to analyze weather trends over time, facilitating better decision-making regarding crop management.

Possible Future Extensions

1. **Advanced Machine Learning Models**:
    - Integrate more sophisticated machine learning algorithms to improve the accuracy and reliability of crop predictions.
2. **Real-Time Monitoring and Alerts**:
    - Implement a system for real-time monitoring of soil and weather conditions with automated alerts for significant changes that may affect crop health or yield.
3. **Data Visualization**:
    - Incorporate data visualization tools to present insights on crop predictions, soil health, and weather trends through graphs and charts, helping users make informed decisions.

4. **Multi-Language Support**:
    - Add support for multiple languages to cater to a broader audience and improve accessibility for users from different regions.
5. **Community and Resource Sharing Platform**:
    - Create a feature for users to share farming tips, resources, and experiences with other users to foster community engagement and knowledge sharing.
6. **Collaboration with Agricultural Experts**:
    - Develop partnerships with agronomists and agricultural experts to provide personalized advice and recommendations based on user data and needs.

11.Bibliography

OpenWeatherMap API documentation: https://openweathermap.org/appid
MongoDB documentation: https://docs.mongodb.com/
React documentation: https://react.dev/
Node.js documentation: https://nodejs.org/docs/latest/api/
Express.js documentation: https://expressjs.com/
MERN Stack Guide: https://www.mongodb.com/mern-stack
OpenCage Geocoding API: https://opencagedata.com/api