BhoomiAI: Technical Documentation & Project Report

Project: BhoomiAI - An AI-Powered Crop Yield Prediction and Optimization Platform

Team: GreenBots

Version: 1.0 (Smart India Hackathon 2025 Submission)

1. Project Overview

1.1. Introduction

BhoomiAI is a full-stack web application developed for the Smart India Hackathon 2025 under the theme of "Agriculture, FoodTech & Rural Development." It addresses the critical need for data-driven agronomic insights among small-scale farmers, who often lack access to expensive precision agriculture technologies. By leveraging modern web technologies and a custom-trained machine learning model, BhoomiAI provides a simple, accessible, and powerful tool for predicting crop yields and making informed farming decisions.

1.2. Problem Statement

Small-scale farmers form the backbone of India's agricultural sector, yet they frequently operate with high uncertainty regarding potential crop yields. Factors such as unpredictable weather, varying soil health, and non-optimized fertilizer usage lead to suboptimal yields, financial instability, and resource wastage. The lack of accessible, affordable, and easy-to-understand predictive tools creates a significant barrier to improving productivity and profitability. BhoomiAI was built to directly tackle this challenge.

1.3. Proposed Solution

BhoomiAI is a "farmer-first" platform that simplifies the complexities of data science into an intuitive workflow. Our solution is twofold:

Data-Driven Prediction: We utilize a custom machine learning model, specifically trained on years of historical agricultural data from the state of Odisha, to provide a quantitative yield prediction. The model considers a rich set of parameters, including historical yearly data for fertilizer consumption (N, P, K), cropping intensity, and user-provided inputs like cultivated area and season.

Actionable Insights through Automation: The platform seamlessly integrates with external services to fetch live, location-specific weather data (temperature and humidity), reducing the data entry burden on the farmer and increasing the accuracy of the inputs fed into our AI system.

The result is a single, clear prediction in kilograms per hectare (kg/ha), empowering farmers to better plan logistics, manage resources, and forecast their income for the season.

2. System Architecture

BhoomiAI is designed using a modern, decoupled three-tier architecture, ensuring scalability, maintainability, and clear separation of concerns.

Frontend (The User Interface): A static web application built with pure HTML, CSS, and vanilla JavaScript.

Backend (The Data Orchestrator): A robust REST API server built with Java and the Spring Boot framework.

Machine Learning Service (The Brain): A specialized microservice built with Python and FastAPI to handle all AI-related computations.

2.1. Architectural Flowchart
(Referencing the flowchart we created earlier)
The system operates on two primary data flows: User & Weather Management (Flow 1) and AI Prediction (Flow 2).
Flow 1 (User & Weather): The Frontend communicates with the Spring Boot Backend for all user-related tasks (identification, profile management) and for orchestrating calls to external services like the Weather API. The Backend is the single source of truth for user and historical farm data, which is stored in a PostgreSQL database.
Flow 2 (AI Prediction): When a prediction is requested, the Frontend gathers all necessary data (from the user, from the Backend via the Weather API) and sends it directly to the Python ML Service. This decoupling ensures that computationally intensive AI tasks do not block the primary backend, allowing the system to remain responsive.
2.2. Technology Stack
Frontend:
Languages: HTML5, CSS3, JavaScript (ES6+)
Libraries: Axios (for API calls), GSAP (GreenSock Animation Platform) for high-performance animations.
Philosophy: A framework-less approach to ensure stability, speed, and simplicity, avoiding the overhead of complex build tools for the hackathon demo.
Backend:
Framework: Spring Boot 3+
Language: Java 21
Core Libraries: Spring Web, Spring Data JPA, Spring Boot DevTools.
Database: PostgreSQL
ORM: Hibernate
Machine Learning Service:
Framework: FastAPI
Language: Python 3.11+
Core Libraries:
scikit-learn: For training the Random Forest model.
pandas: For all data manipulation and pre-processing.
joblib: For serializing and deserializing the trained model.
uvicorn: As the ASGI server to run the FastAPI application.
pydantic: For robust data validation of API requests.
3. The Machine Learning Pipeline: From Raw Data to Prediction
The "Brain" of BhoomiAI is the custom model trained on localized data. The pipeline to create this model was a critical part of the project.
3.1. Data Sourcing and Pre-processing
We sourced four distinct, real-world datasets pertaining to the state of Odisha, reflecting the complexities of official agricultural records:
Seasonal Rice Yield: A multi-level CSV containing historical rice yield and cultivated area, broken down by Autumn, Winter, and Kharif seasons.

State-wide Fertilizer Consumption: Yearly data for N, P, and K usage across the state.
Land Usage Statistics: Yearly data for the state's cropping intensity.
Cleaned Seasonal Yield Data: A user-provided clean dataset which became the final source of truth for yield.
These files were in disparate, "wide" formats unsuitable for machine learning. We developed a master pre-processing script, master_preprocess.py, with the following responsibilities:
Data Ingestion: Loads all four CSV files using pandas, handling complex multi-level headers and inconsistent column names.
Data Cleaning: Standardizes the Year column across all datasets (e.g., converting "1970-71" to the integer 1970). It intelligently renames columns and strips whitespace.
Data Fusion: Performs a series of pandas.merge operations, joining all four datasets into a single, cohesive "master" DataFrame. The Year serves as the primary key for merging the yearly supplemental data (fertilizer, land use) with the primary yield data.
Output: Saves the final, clean, feature-rich dataset as odisha*master*data_final.csv, ready for training.
3.2. Model Training and Evaluation (train_master.py)
With the clean dataset, we proceeded to train our prediction model.
Feature Engineering: The script performs one-hot encoding on the categorical Season column, converting "Autumn," "Winter," and "Kharif" into numerical Season*Autumn, Season*Winter, and Season_Kharif columns (0 or 1). This is a crucial step for the model to understand non-numeric data.
Model Selection: We chose a Random Forest Regressor from scikit-learn. This model is an excellent choice as it is robust, handles non-linear relationships well, is less prone to overfitting than a single decision tree, and provides insights into feature importance.
Training and Validation: The master dataset was split into an 80% training set and a 20% testing set. The model was trained on the training data.
Performance Evaluation: The trained model's performance was evaluated on the unseen testing data. We achieved an R-squared (R^2) value of approximately 0.81, indicating that our model can explain about 81% of the variance in the historical crop yield. The Mean Absolute Error (MAE) was approximately 143 kg/ha , signifying a high degree of accuracy for real-world agricultural data.
Model Serialization: The final, trained model object was saved to a single file, bhoomi*ai*final _model.joblib, using the joblib library for efficient storage and retrieval.
3.3. API Deployment (main.py)
The trained model is deployed via a FastAPI server, which provides a clean, fast, and documented REST API endpoint.
Model Loading: On startup, the server loads the bhoomi*ai*final_model.joblib file into memory.
CORS Configuration: The API includes CORS (Cross-Origin Resource Sharing) middleware, allowing it to accept requests from our frontend application, which is served on a different origin.
Data Validation: It uses pydantic to define a strict data model for incoming requests. This ensures that any call to the /predict endpoint contains all the required features (Year, Area, N,

P, K, etc.) with the correct data types, preventing errors.
Prediction Endpoint (/predict): This POST endpoint receives the feature data in a JSON body. It performs the same one-hot encoding transformation on the Season as was done during training, ensures the data is in the correct column order, and feeds it to the loaded model. The resulting numerical prediction is then packaged into a clean JSON response and sent back to the caller.

4. Backend Service (Spring Boot)

The Spring Boot application acts as the data management and orchestration layer.

4.1. Core Responsibilities

User Management: Provides the /api/v1/users/identify endpoint. The UserService handles the logic to find an existing user by email or create a new one, persisting the data to the PostgreSQL database via Spring Data JPA.
External Service Integration: Provides the /api/v1/weather/fetch-current endpoint. The WeatherService uses WebClient, Spring's modern non-blocking HTTP client, to call the external WeatherAPI.com service. It parses the response and saves a WeatherData record to the database, linking it to the user.
Data Persistence: Uses Spring Data JPA repositories (UserRepository, WeatherDataRepository, etc.) for seamless, boilerplate-free communication with the PostgreSQL database. Hibernate, the underlying ORM, manages the database schema automatically based on the @Entity annotated classes.
Global CORS Configuration: A dedicated WebConfig class implements WebMvcConfigurer to set up global CORS rules, ensuring that requests from the HTML frontend are not blocked by the browser's security policies.

5. Frontend Application (HTML, CSS, JavaScript)

The frontend is built with a "simplicity and beauty" philosophy, ensuring that the powerful backend is accessible through an intuitive interface.

5.1. Structure and User Flow

Multi-Page, Single-Feel Design: The application is split into three core HTML files: index.html (landing), login.html (identification), and dashboard.html (the main application). This separation keeps the code for each view clean and distinct.
State Transfer via localStorage: When a user successfully identifies themselves on login.html, their user details (name, email, location) are saved into the browser's localStorage. When dashboard.html loads, its JavaScript reads this data to personalize the experience, creating a seamless transition between pages without needing a complex state management library.

5.2. Styling and Animation

Custom CSS: Each HTML page has its own dedicated stylesheet (style.css, login.style.css, etc.), allowing for highly targeted and unique designs for each part of the user journey.
Vibrant, Thematic Design: The UI uses a professional color palette of vibrant greens, warm blues, and clean whites, coupled with a custom background image and "glassmorphism" transparency effects to create a modern, high-tech aesthetic.
GSAP for High-Fidelity Animations: The GreenSock Animation Platform (GSAP) library is used to power all animations. This includes sophisticated scroll-triggered animations on the landing

page, smooth page transitions, and a dynamic "number countdown" for the final prediction result, providing a premium user experience.

5.3. Functionality and API Integration (dashboard.js)

The core logic resides in dashboard.js. It handles all user interactions on the main dashboard.

Event-Driven: It uses addEventListener to listen for clicks on the "Fetch Weather" and "Predict Yield" buttons.

Asynchronous API Calls: It uses the axios library to make asynchronous POST requests to both the Spring Boot backend (for weather) and the Python ML Service (for predictions). async/await syntax is used for clean, readable code.

Dynamic UI Updates: Upon receiving successful responses from the APIs, the script uses document.getElementById to find the relevant elements on the page and dynamically update their content (innerHTML or textContent), presenting the live weather and final AI prediction to the user in real-time.

6. Conclusion & Future Scope

BhoomiAI successfully demonstrates a complete, end-to-end implementation of an AI-powered agricultural tool. By fusing real-world historical data, training a specialized machine learning model, and wrapping it in a seamless and intuitive full-stack application, we have built a platform that is not only technologically impressive but also directly addresses a critical need in the agricultural community.

Future enhancements could include:

Training separate models for different crops (e.g., Wheat, Maize).

Integrating district-level historical weather data to complement the live API.

Adding a "recommendations" module to the Spring Boot backend that suggests optimal N-P-K values based on the farmer's desired yield.

Developing a progressive web app (PWA) for better offline access and mobile experience.