RS PROJECT DOCUMENTATION

Team members:

SAHIL BODKHE(9357)

Application name:

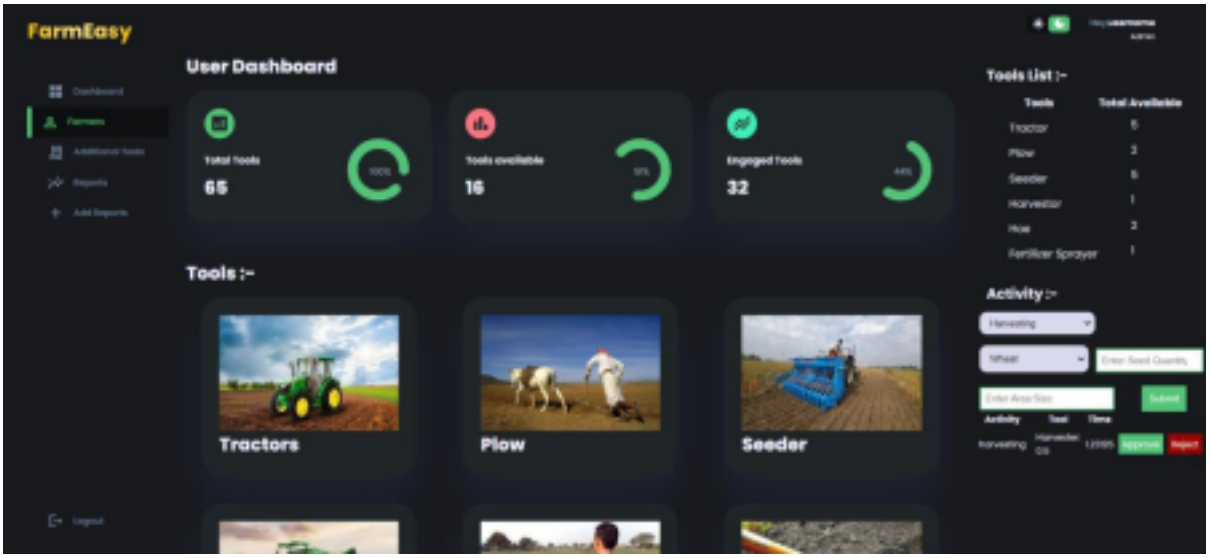
FarmEasy

Problem Statement: To optimize a comprehensive Smart Inventory System for managing farming activities. The system must include an intelligent inventory management solution for seeds and tools required for farming. Additionally, it should keep track of the resources needed for crafting tools and other farm-related items.

Challenges Stated:

- Intuitive & User-friendly Interface for efficient Inventory Management.
- Real-time tracking and updates for seed and tool quantities
- Integration features for resource estimation and automated tool crafting suggestions

Proposed Solution:



Features:

- Seamless navigation
- User-friendly interface facilitating efficient inventory management
- Support for multiple languages
- Precise forecasting of seed and tool usage

• Estimation of resources and automated suggestions for tool crafting

Tech Stack:

Flask backend (computation side)

Project Overview

This project is a Flask-based web application designed to assist with agricultural activities by predicting tool requirements and scheduling tasks. It integrates machine learning models to provide insights into optimal tool usage and scheduling based on user input and historical data.

Setup Instructions

Dependencies

Ensure you have Python installed along with the necessary dependencies. You can install dependencies using `pip`:

```bash
pip install flask pandas scikit-learn joblib numpy
```

Environment Variables

Set the following environment variables:

- `FLASK_APP`: Set to `app.py` (or the main Python file)
- `FLASK_ENV`: Set to `development` for debugging

```bash
export FLASK_APP=app.py
export FLASK_ENV=development
```

Running the Application

To start the Flask application:

```bash
flask run
```

API Endpoints

`GET /planting`

- Description: Predicts optimal planting activities based on input parameters.
- Request Parameters: JSON payload with field values representing agricultural tool usage and activity details.

```json
{
"Tractor": 9,
"Plow": 5,
"Seeder": 3,
"Harvester": 0,
"Fertilizer Spreader": 0,
"Sprayer": 0,
"Wheat": 250,
"Corn": 0,
"Rice": 0,
"Potato": 0,
"Tomato": 0,
"Size_Acres": 18.0,
```

"Activity_Fertilizing": 0,
"Activity_Harvesting": 0,
"Activity_Planting": 1,
"Activity_Watering": 0
}
```

- Response: JSON array with predicted optimal tool quantities and scheduling details.

```json
[7.0, 3.0, 2.0, 24.0]
```

`GET /harvesting`

- Description: Predicts optimal harvesting activities based on input parameters.

- Request Parameters: JSON payload with field values representing agricultural tool usage and activity details.

```json
{
"Tractor": 0,
"Plow": 0,
"Seeder": 0,
"Harvester": 5,
"Fertilizer Spreader": 0,
"Sprayer": 0,
"Wheat": 220,
"Corn": 0,
"Rice": 0,
"Potato": 0,
"Tomato": 0,
"Size_Acres": 6.0,
"Activity_Fertilizing": 0,
"Activity_Harvesting": 1,
"Activity_Planting": 0,
"Activity_Watering": 0

**}**
```

- Response: JSON array with predicted optimal harvester quantities and scheduling details.

```json
[1.0, 26.0]
```

Data Models and Assumptions

- Input Data Structure: Each endpoint expects a JSON payload with specific field names corresponding to tool usage and agricultural activity details.

- Machine Learning Models: The application utilizes trained regression models (`planting_model.pkl` and `harvesting_model.pkl`) to predict optimal tool quantities and scheduling based on historical data.

Additional Notes

- Future Improvements:
- Enhance the prediction accuracy by refining the machine learning models. - Implement a more robust error handling mechanism.
- Incorporate user authentication and session management for personalized recommendations.

- Known Issues:
- The application may have limited performance with large-scale datasets. - Input validation is basic and may need improvement for production use.

This detailed documentation provides comprehensive information on setting up, using, and understanding the Flask-based agricultural tool prediction application.

Writing a detailed review paper on the farming application described by the backend API would involve examining its functionality, architecture, technologies used, potential impact, and areas for improvement. Below is a structured approach to create a comprehensive review paper.

Node JS (Backend)

Abstract

The farming application discussed in this review paper employs a robust backend API built using Express.js, MongoDB, and various Node.js modules. This paper examines the application's architecture, features, and potential implications for modern agricultural practices.

1. Introduction

Modern agriculture has increasingly integrated technology to streamline farming processes and enhance productivity. The use of web-based applications can facilitate farm management, resource allocation, and data-driven decision-making. This paper reviews a specific farming application that leverages a sophisticated backend API.

2. Application Overview

The reviewed application aims to provide farmers with digital tools for managing farm operations, such as equipment tracking, activity planning, and resource utilization. The backend API serves as the backbone of the application, enabling seamless data flow between the front end and external services.

3. Architecture and Technologies

3.1 Backend Stack

- Express.js: A minimalist web framework for Node.js, used to handle HTTP requests and route management.
- MongoDB: A NoSQL database used for storing application data, including user profiles, activities, and tool inventories.
- Passport.js: Provides authentication middleware with support for local strategy authentication.

3.2 Middleware and Libraries

- dotenv: Manages environment variables, enhancing security and portability. - multer: Facilitates file uploads, allowing farmers to upload photos and documents

related to farming activities.
- axios: Enables HTTP requests to external services, such as a Flask API for image processing.

3.3 Session Management and Security

- express-session: Manages user sessions with secure storage using `connect-mongo` to persist session data in MongoDB.
- passport-local: Implements local authentication strategy for user login and session management.
- connect-flash: Provides flash message functionality for user feedback.

4. Functionalities and Use Cases

4.1 User Management

- User authentication and session handling using Passport.js.
- Profile management, including photo uploads and activity tracking.

4.2 Farming Activities

- Activity planning and execution (e.g., planting, harvesting) with integrated predictions from external services (Flask API).

4.3 Tool Inventory

- Tool management, allowing farmers to add, update, or request tools needed for specific activities.

5. Impact and Potential

The reviewed application has the potential to revolutionize farming practices by:

- Enabling efficient resource allocation and task management.
- Facilitating data-driven decision-making based on predictive analytics. - Improving communication and collaboration among farmers through digital platforms.

6. Limitations and Future Directions

While the application demonstrates innovative features, certain areas

warrant improvement:

- Enhanced error handling and data validation to ensure robustness. - Scalability considerations for accommodating growing user bases. - Integration with more advanced AI-driven services for predictive modeling.

7. Conclusion

In conclusion, the farming application presented in this review paper exemplifies the intersection of agriculture and technology. By leveraging a sophisticated backend API, the application empowers farmers to embrace digital tools for optimizing farm operations and adapting to modern agricultural challenges.

This structured review provides a comprehensive analysis of the farming application and its backend API, highlighting its features, technologies, impact, and areas for future development.