Technical Requirements Document for the “AgroBoost” Project

1. **Full name of the work: “AgroBoost”**

AgroBoost is a cutting-edge platform aimed at transforming agriculture by integrating advanced analytics and automation. It utilizes the latest technologies to offer farmers data-driven insights, helping to optimize resource use and maximize crop productivity.

2. **Team name and technology stack used**

Our team named “Al Xorazmiy”. Project employs Django for backend operations, Vue.js for frontend interfaces, and Kubernetes for scalable deployment, with a primary focus on Python as the core programming language.

✓ ***Backend Framework***: Django (Python)

Core Responsibilities:

- Serve RESTful API endpoints.
- Manage data storage, retrieval, and processing of analytical results from ML models.
- Integrate with machine learning services for image/video analysis.
- Implement user authentication and role-based access controls.

✓ ***Frontend Framework***: Vue.js

Core Responsibilities:

- Interface with the backend APIs to present analysis results.
- Provide an interactive, responsive dashboard for data visualization and user interaction.
- Facilitate functionalities such as video/image uploads, field monitoring, and reporting.

✓ ***Machine Learning Module Technologies***: OpenCV, Python

Core Responsibilities:

- Image and video analysis to detect plant health status, diseases, pests, and nutrient deficiencies.

- Generate quantitative metrics such as NDVI (Normalized Difference Vegetation Index) and other vegetation indices.

- Suggest appropriate interventions based on detected issues.

✓ ***Deployment and Orchestration Platform***: Kubernetes

Responsibilities:
- Manage containerized deployments of backend, frontend, and ML modules.
- Ensure system scalability, high availability, and automated fault tolerance.

✓ ***Drone Integration* Technologies**: DJI SDK/MAVSDK, Python

Responsibilities:
- Capture high-resolution imagery and transmit data to the backend.
- Support real-time data streaming and ensure seamless integration with ML processing workflows.

3. **Project Assignment**

The **"AgroBoost"** project aims to deliver an advanced agricultural monitoring solution utilizing drone imagery, machine learning (ML), and computer vision technologies. The solution enables comprehensive analysis of farmland, offering actionable insights on crop health, pest identification, and nutrient deficiencies.

Our project offers various functionalities by client type:

For farmers:

- Get crop area information through an easy interface.
- Monitoring the condition of crops in real time.
- Receive automatic alerts for fertilizer and pest control measures.

For customers:

- Order food online.
- Product delivery service directly from farms.

For operators:

- Analysis of data from satellites and drones.
- Configure automated flight and monitoring plans for drones.

4. **Functional Requirements**

✓ ***Image/Video Analysis:***

- **Data Capture**: Drones equipped with high-resolution cameras should autonomously capture farmland imagery;
- **Health Assessment**: Analyze plant health using vegetation indices and ML models to provide greenness measurements;
- **Disease & Pest Identification**: Employ computer vision techniques to detect diseases, pests, and other anomalies.
- **Nutrient Analysis**: Analyze leaf coloration to detect deficiencies and provide actionable nutrient management recommendations.
- **Pesticide Recommendations**: Suggest targeted interventions based on pest/disease identification.

4.1. **User Interface**

- **Dashboard Visualization**: Display a comprehensive view of scanned fields with health indicators using geospatial mapping techniques.

- **Data Reporting**: Offer detailed reports and analytics on crop health, diseases, and suggested treatments, accessible via the user dashboard.

- **Alert Mechanism**: Real-time alerts for detected issues, tailored to user roles (e.g., farmers, agronomists).

4.2**. Data Management**

- **Storage:** Use a structured database system for storing drone footage, analysis results, and historical data.

- **Retrieval and Search**: Implement advanced filtering and querying capabilities based on parameters such as date, geographic location, or crop type.

4.3. **User Authentication and Access Management**

Authentication Mechanism: Implement secure user authentication using JWT tokens and Django's built-in authentication modules.
Role-Based Access Control: Differentiate access and functionalities based on predefined user roles.

5. **Performance and Scalability**

Throughput: The system must handle multiple concurrent drone uploads and processing requests efficiently.
Latency: Ensure analysis results are delivered with minimal delay, achieving near-real-time feedback.

5.1. **Reliability and Availability**

Redundancy: Utilize Kubernetes for failover and redundancy to ensure high availability.
Data Backup: Configure automated data backup strategies with appropriate retention policies.
Here's detailed information about MinIO that you can add to your requirements document for object storage:

6. **Additional integrations**

MinIO is an open-source, high-performance, S3-compatible object storage system designed to handle unstructured data such as photos, videos, log files, backups, and container/VM images. It can be deployed on-premises, in the cloud, or as part of a hybrid infrastructure.

Key Features:

- **S3 Compatibility**: MinIO offers full compatibility with Amazon S3, enabling seamless integration with applications and services using the S3 API. This makes migration or hybrid deployments easier.
- **Scalability**: MinIO can scale horizontally by adding additional nodes, which allows it to manage petabytes of data efficiently. It supports erasure coding and bit-rot detection, ensuring data integrity.
- **High Performance**: MinIO is optimized for high throughput and low latency, making it suitable for applications that require real-time data access, such as big data, AI/ML workloads, and analytics.
- **Data Protection**:Erasure Coding: Provides data protection against hardware failures, allowing data reconstruction even if multiple drives or servers fail.
- **Replication**: Supports active-active, bucket-level replication across clusters, ensuring data is available across different geographic locations.
- **IAM Integration**: MinIO supports Identity and Access Management (IAM) for defining policies and controlling access to data, ensuring fine-grained access control.
- **Multi-Cloud and Kubernetes Support**: Can be deployed in various environments, including on-premises, multi-cloud, or Kubernetes clusters, providing flexibility and compatibility with modern infrastructure setups.
- **Server-Side Processing**: MinIO allows server-side processing of objects using Lambda-like functions, enabling event-driven workflows and data processing tasks directly on the storage layer.

6.1. **Use Cases of MinIO**

- **Big Data and Analytics**: Storing and retrieving large datasets, logs, and media files efficiently.
- **Machine Learning and AI**: Storing training datasets and models, with high throughput for real-time training and inference.
- **Backup and Disaster Recovery**: Provides an efficient solution for storing backups with replication and encryption.
- **Hybrid Cloud Storage**: Facilitates data storage across on-premises and cloud infrastructure with S3 compatibility.

6.2 Pros and Cons

Pros:

- High performance with low latency.

- Easy scalability and deployment.

- Strong security and encryption features.

- Complete S3 compatibility.

- Suitable for both small and large deployments.

Cons:

- Lack of support for traditional file system protocols (e.g., NFS, SMB) natively.

- Not ideal for workloads requiring high IOPS (e.g., databases).

6.3 **Licensing and Support**

- MinIO offers an open-source community edition under the Apache 2.0 License.

- Enterprise Edition provides advanced features like multi-site replication, technical support, and other enterprise-grade functionalities.

7. **Integrated Open Source Model**

DeepForest is an open-source deep learning framework specifically designed for the detection of individual trees from high-resolution aerial and satellite imagery. It leverages a pre-trained model based on the Faster R-CNN architecture, which is widely recognized for its ability to perform object detection tasks with high accuracy and efficiency. This integration allows DeepForest to effectively identify and delineate trees within complex and dense forested landscapes, even in cases where overlapping canopies and varying vegetation types pose challenges.

The framework is highly adaptable and can be integrated with various data pipelines, making it suitable for use in large-scale ecological monitoring, forestry management, and environmental research projects. By utilizing pre-labeled datasets and allowing further training with custom data, DeepForest provides flexibility and precision in detecting trees across diverse geographic regions. Its ability to analyze high-resolution imagery and provide detailed, object-level insights makes it an invaluable tool for applications such as land use monitoring, carbon accounting, biodiversity assessment, and the management of natural resources.

8. **Instructions to run project**

Here's the updated guide with separate instructions for running the project locally and with Docker. Setting Up Django and Vue.js Project:

- Docker: Download from [Docker's official website](https://www.docker.com/get-started) if running with Docker.

- Git: Install from [Git's official website](https://git-scm.com/downloads).

- Python: Required for local Django setup. Download from [Python's official site](https://www.python.org/downloads/).

- Node.js & Yarn: Required for local Vue.js setup. Download Node.js from [Node’s website](https://nodejs.org/en/) and Yarn from [Yarn’s website](https://classic.yarnpkg.com/en/docs/install/).

8.1. Running the Project Locally

Step One: Clone the Project Repository

“bash

git clone https://github.com/AAmir007-code/AgroBoost

cd AgroBoost”

Step Two: Setting Up the Backend (Django)

A) Navigate to the `api` directory**:**

“bash

cd api”

B) Create a virtual environment**:**

```***bash***

***python -m venv venv***

```

Activate it**:**

- Windows: ***`venv\Scripts\activate`***

- macOS/Linux: ***`source venv/bin/activate`***

C) Install Python dependencies:

```***bash***

***pip install -r requirements.txt*** ```
```

D) Set up environment variables:

Create a `.env` file in the `api` directory with the following:

```env
DEBUG=1
SECRET_KEY=your-secret-key
DJANGO_ALLOWED_HOSTS=localhost 127.0.0.1 [::1]
DATABASE_URL=sqlite:///db.sqlite3
```

E) Run migrations:

```bash
python manage.py migrate
```

F) Start the Django server:

```bash
python manage.py runserver
```

Step Three: Setting Up the Frontend (Vue.js)

A) Navigate to the `frontend` directory:

```bash
cd ../frontend
```

B) Install Node.js dependencies:

```bash
```

yarn install

C) Start the Vue.js development server:

```bash

yarn serve

```

Access your project locally:

- **Django Backend:** http://localhost:8000

- **Vue.js Frontend:** http://localhost:8080

8.2. Running the Project with Docker

Step One: Clone the Project Repository

If not already done, clone the project:

```bash

git clone https://github.com/AAmir007-code/AgroBoost

cd AgroBoost

```

Step Two: Ensure Directory Structure

The project should have the following structure:

```/project-root
│
├── api/            # Django backend
│   ├── Dockerfile
│   ├── requirements.txt
│   └── manage.py
│
```

```
├── frontend/       # Vue.js frontend
│   ├── Dockerfile
│   └── package.json
│
└── docker-compose.yml
```

Step Three: Docker Setup

A) Create Dockerfile for Backend (Django)

Ensure there's a `Dockerfile` in the `api` folder:

```Dockerfile
# Backend Dockerfile

FROM python:3.11-slim


# Set environment variables

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1


# Set work directory

WORKDIR /api


# Install dependencies

COPY requirements.txt /api/

RUN pip install --upgrade pip && pip install -r requirements.txt


# Copy project files

COPY . /api/
```

B) Create Dockerfile for Frontend (Vue.js)

Ensure there's a `Dockerfile` in the `frontend` folder:

```Dockerfile
# Frontend Dockerfile
FROM node:18.16.1


# Set working directory
WORKDIR /frontend


# Install dependencies
COPY package.json yarn.lock /frontend/
RUN yarn install


# Copy project files
COPY . /frontend/


# Build the project
RUN yarn build
```

C) Create `docker-compose.yml` File

Ensure you have a `docker-compose.yml` file in your project root:

```yaml
```

```
version: '3.8'

services:

  backend:

    build:

      context: ./api

    container_name: django-backend

    command: python manage.py runserver 0.0.0.0:8000

    volumes:

      - ./api:/api

    ports:

      - "8000:8000"

    depends_on:

      - db


  frontend:

    build:

      context: ./frontend

    container_name: vue-frontend

    stdin_open: true

    environment:

      - CHOKIDAR_USEPOLLING=true

    volumes:

      - ./frontend:/frontend
    ports:
      - "8080:8080"
    command: yarn serve
    depends_on:
      - backend

  db:
    image: postgres:15
    container_name: postgres-db
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: mydatabase
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

volumes:
  postgres_data:
```

Replace `myuser`, `mypassword`, and `mydatabase` with your PostgreSQL credentials.

D) Set Up Environment Variables

In the `api` folder, create a `.env` file with:

```env
DEBUG=1
SECRET_KEY=your-secret-key
DJANGO_ALLOWED_HOSTS=localhost 127.0.0.1 [::1]
DATABASE_URL=postgres://myuser:mypassword@db:5432/mydatabase
```

Step Four: Build and Start Containers

Run the following command from your project root:

```bash
docker-compose up --build
```

Docker will build the images and start the Django backend, Vue.js frontend, and PostgreSQL database services.

Access Your Project:

- Django Backend: http://localhost:8000
- Vue.js Frontend: http://localhost:8080

Stopping the Project

To stop all running containers:

```bash
docker-compose down
```

`` ``` ``

This command will stop and remove all containers, networks, and volumes created by Docker Compose.

Additional Tips for Docker Setup

- Run Django Migrations:

  ```bash
  docker-compose exec backend python manage.py migrate
  ```

- Collect Static Files (for production):

  ```bash
  docker-compose exec backend python manage.py collectstatic --noinput
  ```