UNIVERSITY OF THE WESTERN CAPE

MODULE: IFS 325

GROUP ASSIGNMENT

DEPLOYMENT DOCUMENTATION

GROUP 3 –

CROP DEVELOPMENT VISION SYSTEM

SUBMITTED BY GROUP 5:

RAEESAH DARBAR: 4356374

MUBASHIER OMAR: 4348127

MUAATH SALIE: 4369122

DYLAN-THOMAS PUGH: 43553847

NTOKOZO THOKOZANI MHLAMBI: 4337274

MAAJIDA JAKOET: 4227672

SUBMITTED TO: RUCHEN WYNGAARD

DATE OF SUBMISSION: 05 NOVEMBER 2025

Contents

1. Introduction 2
1.1 Overview of the Deployment Architecture 2
1.2 Components in the Deployment Pipeline 2
2. System Requirements 3
2.1 Hardware Requirements 3
2.2 Software Requirements 3
2.3 Network Requirements 4
2.4.1 Functional Requirements 4
2.4.2 Non-Functional Requirements 4
3. Environment Setup 5
3.1 Repository Structure Overview 5
3.2 Installing Dependencies 5
3.3 Environment Variables & API Keys 5
4. AI Model Integration & Raspberry Pi Setup 6
4.1 Placing Trained Models on Device 6
4.2 Running the AI Prediction Script 6
5. Cloud Integration 7
5.1 Firebase Setup (Storage & Database) 7
5.2 Connecting Raspberry Pi to Firebase 7
5.3 MQTT Communication Setup (Topics & Broker) 8
5.4 Data Management Team Integration 8
6. Flutter Frontend Deployment 9
6.1 Connecting App to Firebase 9
6.2 Fetching and Displaying Data 9
6.3 Testing Real-Time Updates 10
7. End-to-End Deployment Steps 10
8. Testing and Troubleshooting 11
8.1 Common Errors (Network, MQTT, Firebase) 11
Most issues can be resolved by verifying network, MQTT, and Firebase configurations. 11
8.2 Debugging Tips 11
9. Maintenance 11
9.1 Updating Models or Scripts 11
9.2 Version Control & Backup Procedures 12
10. Conclusion 12
11. Appendix 13
11.1 Command Reference Summary 13
11.2 Example Configuration Files (config.py, firebase.json) 14
11.3 Deployment Checklist 15

1. Introduction

1.1 Overview of the Deployment Architecture

FarmEye follows a modular deployment architecture that integrates artificial intelligence, edge computing, and cloud-based services to deliver an end-to-end crop monitoring solution.

The system operates on a Raspberry Pi, which serves as the central processing unit for running optimized deep learning models. These models analyse live camera input to detect crop growth stages, diseases, and pests in real time.

To ensure scalability and data accessibility, the system uses:

- Firebase Cloud Storage for captured images and diagnostic outputs.
- Oracle Apex for structured data management.
- A Flutter-based mobile/web application that connects to these cloud platforms, allowing users to view historical records, track crop development, and receive recommendations remotely.

This distributed architecture ensures that computationally intensive tasks are handled efficiently at the edge while maintaining real-time performance and reliable cloud synchronization.

1.2 Components in the Deployment Pipeline

The deployment pipeline consists of three key components working together to deliver intelligent crop monitoring and data management:

- Raspberry Pi (Edge Device)-
  Acts as the local inference unit. It captures images via the camera module, runs AI models on-device, and sends results to Firebase and Oracle Apex. The Pi's graphical interface includes dropdown menus for selecting crop types and detection modes, providing on-site monitoring capabilities.
- Firebase cloud integration -
  Used as secure cloud storage for images and system outputs. Each image captured by the Raspberry Pi is uploaded to Firebase Storage, while metadata (crop name, timestamp, confidence) is stored in Firebase Firestore. This ensures secure and synchronized access for connected applications.
- Flutter frontend application (FarmEye Farmer Dashboard) -
  Provides an intuitive interface that retrieves data from Firebase and Oracle Apex. Users can view captured images, analyse detected issues, and read AI-generated recommendations. The app also supports manual uploads and dashboard analytics for soil and temperature data shared by the Data Management team.

Together, these components form a seamless pipeline that enables continuous crop monitoring, real-time insights, and efficient data sharing between edge devices, cloud platforms, and end users.

2. System Requirements

2.1 Hardware Requirements

FarmEye relies on lightweight, yet capable hardware components designed for greenhouse and field environments:

Component	Purpose
Raspberry Pi 5	Main processing unit for running AI models and managing local inference.
Camera Module	Captures live crop images for disease, pest, and growth stage analysis.
MicroSD card	Stores OS, Python environment, model files, and application code.
Power supply	Provides stable power for continuous operation.
HDMI display/monitor	Displays live detection interface and dashboard.
Keyboard & Mouse	Used for setup and manual testing.
Network connectivity (Wi-Fi or Ethernet)	Enables communication with Firebase, MQTT broker, and Oracle Apex.

2.2 Software Requirements

The following tools and frameworks power FarmEye's AI, data storage, and user interface components:

Software Components	Requirements
Operating system	Raspberry Pi OS (64-bit)
Programming language	Python 3.11+
Deep learning framework	Tensorflow/Keras
Computer vision library	OpenCV
GUI Framework	CustomTkinter
Cloud integration	Firebase SDK and Firebase Admin
Messaging protocol	Paho-MQTT
Frontend Framework	Flutter SDK
Database management	Oracle Apex
Version control	Git and GitHub

Together, these tools ensure smooth coordination between the backend (Python and TensorFlow) and cloud storage (Firebase) and frontend (Flutter).

2.3 Network Requirements

For reliable connectivity across components:

- Internet Connectivity: Stable Wi-Fi or Ethernet required for image upload and cloud sync.
- MQTT Broker: Broker configured with correct topic, port (1883), and authentication.
- Firewall Rules: Keep outbound ports 1883 (MQTT) and 443 (HTTPS) open.
- Bandwidth: Minimum 2 Mbps upload speed recommended.
- Security: Use MQTT authentication or TLS where possible.

2.4.1 Functional Requirements

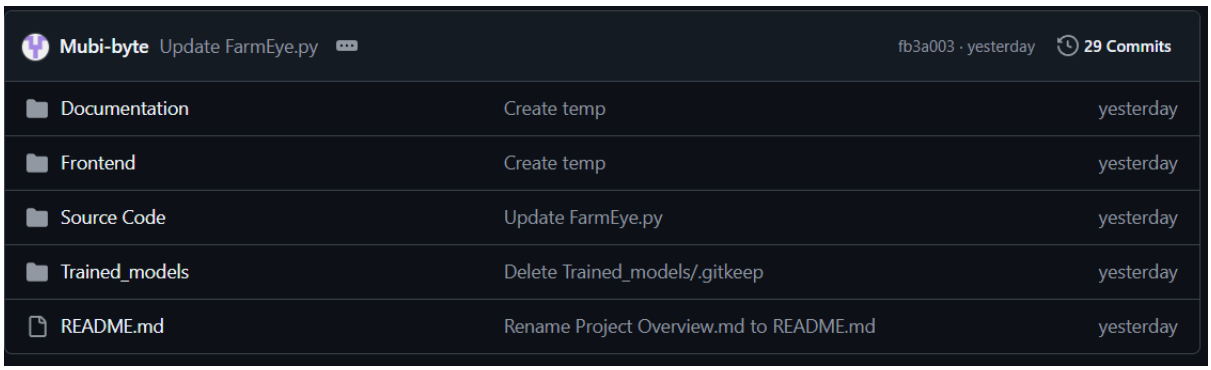
ID	Requirement	Summary
FR1	Crop Detection	Capture and analyze crop images using AI on Raspberry Pi.
FR2	Cloud Sync	Upload images and metadata to Firebase and Oracle Apex.
FR3	Farmer Dashboard	Display uploads, timestamps, and confidence levels in the Flutter app.
FR4	Analytics	Show soil moisture and temperature data from Team 3's sensors.
FR5	Recommendations	Provide AI-based advice for crop care.
FR6	Offline Access	Allow data viewing and uploads when offline.

2.4.2 Non-Functional Requirements

ID	Requirement	Summary
NFR1	Performance	AI results within 5–10 s per image.
NFR2	Usability	Simple, intuitive interface for farmers.
NFR3	Reliability	No data loss during cloud sync
NFR4	Security	Encrypted communication and user authentication.
NFR5	Scalability	Support multiple devices uploading simultaneously.
NFR6	Availability	System uptime of at least 95%.

3. Environment Setup

3.1 Repository Structure Overview



3.2 Installing Dependencies

In order to make use of the application, specific Python libraries are required to be installed on the Raspberry Pi 5. The following steps need to be followed in order to get the application functioning as it should:

1. *Update System Packages*	sudo apt update && sudo apt upgrade -y
2. *Install System-Level Dependencies*	sudo apt install -y python3-pip python3-venv libatlas-base-dev libopenblas-dev libhdf5-dev
3. *Create a Python Virtual Environment*	python3 -m venv cropvision_env source cropvision_env/bin/activate
4. *Install all the Python libraries in the virtual environment using pip*	pip install --no-cache-dir --prefer-binary tensorflow opencv-python pillow customtkinter paho-mqtt firebase-admin picamera2

3.3 Environment Variables & API Keys

The application makes use of configuration files for sensitive keys. They should not be hardcoded into the script at any time. Below is how to set up the sensitive keys for the application

3.3.1. Firebase Service Account Key

- Obtain from Firebase Console → Project Settings → Service Accounts.
- Generate and download a private JSON key (firebase-key.json).
- Place it in the project root directory.
- Add to .gitignore before committing to GitHub.

3.3.2. MQTT Broker configuration

- Define constants in the script:

```
MQTT_BROKER = "broker_ip"
MQTT_PORT = 1883
MQTT_TOPIC = "arc/Crop-monitoring/group5/Crop-sensor"
```

- Ensure broker IP, port, and topic match the Data Management team configuration.

4. AI Model Integration & Raspberry Pi Setup

4.1 Placing Trained Models on Device

Store all trained models in the /models directory on the Raspberry Pi.

Model	Filename	Input	Purpose
Apple Diseases	apple_disease_model.keras	224×224 RGB	Detect apple leaf diseases
Grape Diseases	grape_disease_model.keras	224×224 RGB	Detect grape leaf diseases
Tomato Fruit Diseases	tomato_fruit_model.keras	224×224 RGB	Detect fruit-based diseases
Tomato Leaf Diseases	tomato_leaf_model.keras	224×224 RGB	Detect leaf-based diseases
Pest Detection	pest_mobilenetv2.keras	224×224 RGB	Detect common pest
Lettuce Growth Stage	lettuce_growth_stage.h5	112×112 RGB	Classify lettuce growth
Basil Growth Stage	basil_growth_stage.h5	112×112 Grayscale	Classify basil growth

4.2 Running the AI Prediction Script

Once everything is set up, you can launch the application.

1. Navigate to the project directory
2. Activate the virtual environment:

source cropvision_env/bin/activate

3. Run the main script:

python3 crop_vision_app.py
The GUI window should open, and the application should be ready to use

Using the App:

- Start Live Detection → Select category & model → Click Start.
- Capture Screenshot → Saves locally + uploads to Firebase + publishes via MQTT.
- Upload Image → Manual upload for remote analysis.
- Stop Detection → Ends camera stream safely.

5. Cloud Integration

5.1 Firebase Setup (Storage & Database)

- Create a Project on Firebase Console.
- Enable Cloud Storage → "Build → Storage → Get Started".
- Set in Test Mode for development.
- Generate Service Account Key and download JSON file.

5.2 Connecting Raspberry Pi to Firebase

This process describes how to link your Python Application on the Raspberry Pi to your Firebase Project.

1. Transfer the key file
   - The JSON key file, which was previously downloaded after generating the service account key, should now be moved to the same directory as the project on the Raspberry Pi.
2. Configure the Application
   - In the crop_vision_app.py script, ensure the following configuration variables that are at the top of the file match the Firebase project:
     FIREBASE_BUCKET = "your-project-name.firebasestorage.app"
     FIREBASE_KEYFILE = "firebase-key.json"
3. Verify connection to the Firebase project
   - When the application is run, look for a message within the terminal:"[LOG]Firebase initialized successfully."
   - When you capture or upload an image, check for the message:"[LOG]Uploaded to Firebase :[PUBLIC_URL]". You can open this URL in a browser to make sure that the image is accessible.

5.3 MQTT Communication Setup (Topics & Broker)

MQTT is a lightweight protocol used to push analysis results in real-time to the Data Management team's central platform.

```
MQTT_BROKER = "broker_ip"
MQTT_PORT = 1883
MQTT_TOPIC = "arc/Crop-monitoring/group5/Crop-sensor"
```

- Message Payload (JSON Format):

The data that is sent to the MQTT broker is a JSON object containing all the essential information for the data platform and farm managers.

```
{
  "Group_ID": "Group5",
  "Crop_ID": "capture_1734567890",
  "Label": "Early Blight",
  "Confidence": "89.5%",
  "Recommendation": "Remove infected leaves and apply copper-based spray.",
  "Image_URL": "https://storage.googleapis.com/.../capture_1734567890.jpg"
}
```

5.4 Data Management Team Integration

The Data Management Team receives crop health data via MQTT in real time.
Each detection event publishes a JSON payload with:

- Group ID
- Crop ID
- Label
- Confidence
- Recommendation
- Firebase image URL

This ensures seamless integration between FarmEye, Oracle Apex, and the central dashboard.

6. Flutter Frontend Deployment

6.1 Connecting App to Firebase

The FarmEye Farmer Dashboard (Flutter frontend) connects to Firebase to enable real-time synchronization between the Raspberry Pi detection system and the farmer's mobile or web interface.

A dedicated Firebase project was created with Cloud Firestore as the primary database and Firebase Storage for hosting captured crop images. Each detection result from the Raspberry Pi is stored in the Firestore collection *"detections."*

- The following dependencies were added:

firebase_core: ^3.0.0

cloud_firestore: ^5.0.0

firebase_storage: ^12.0.0

- Firebase was initialized in the app's entry point (main.dart) as shown below:

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const FarmEyeApp());
}
```

The app is now ready to read and display data that the Raspberry Pi uploads to Firestore in real time.

6.2 Fetching and Displaying Data

The FarmEye dashboard retrieves crop detection results from two main sources:

- Firebase Storage – Stores captured crop images.
- Oracle Apex – Stores structured metadata (timestamp, crop type, detected disease, confidence score).

Each record in Oracle includes an image identifier that matches an image in Firebase Storage. The app retrieves both data sources, combines them, and displays results in the dashboard using Flutter widgets such as ListView and Card.

Each entry displays:

- Crop image

- Disease or pest name
- Confidence level
- Timestamp

Separating image and metadata storage improves performance, scalability, and synchronization between the Raspberry Pi detections and the farmer's dashboard.

6.3 Testing Real-Time Updates

Integration testing confirmed seamless synchronization between Raspberry Pi, Firebase, and Oracle Apex.
During testing:

- The Raspberry Pi successfully uploaded images and metadata after each detection.
- The FarmEye app refreshed to display the latest detection results in real time.

This verified that FarmEye maintains accurate, up-to-date information, ensuring farmers always have access to the latest crop health insights.

7. End-to-End Deployment Steps

7.1 Deployment Flow (Pi → Firebase → Flutter)

The FarmEye deployment workflow operates in a streamlined three-step process:

1. Raspberry Pi: Captures images, runs AI models, and uploads results to Firebase.
2. Firebase Cloud: Stores both images and metadata securely.
3. FarmEye Dashboard: Displays the processed results for farmers in real time.

This setup creates a robust data pipeline from edge to cloud to user interface.

7.2 Verifying Data Flow and Model Predictions

End-to-end verification confirmed that:

- The Raspberry Pi correctly uploads images and model results to Firebase.
- The FarmEye dashboard retrieves and displays data from Firebase and Oracle Apex without delay.
- AI model outputs are accurate and consistent when compared with known results.

These validations ensure a smooth, reliable, and accurate data flow across all components.

8. Testing and Troubleshooting

8.1 Common Errors (Network, MQTT, Firebase)

During deployment, several common errors may occur due to connectivity issues, configuration mismatches, or hardware limitations.

Issue	Cause	Solution
Network Errors	Unstable Wi-Fi or incorrect credentials	Verify Wi-Fi configuration, ensure ports 1883 (MQTT) and 443 (HTTPS) are open, and restart the Pi.
MQTT Failures	Incorrect broker address, port, or credentials	Verify IP, topic name, and port; restart the Mosquitto service before launching the app.
Firebase Upload Errors	Invalid credentials or misconfigured key file	Check service account key path, re-authenticate Firebase, and confirm read/write permissions.
Camera Detection Errors	Disabled or disconnected camera module	Enable via raspi-config, check cable connections, and test using libcamera-still.

Most issues can be resolved by verifying network, MQTT, and Firebase configurations.

8.2 Debugging Tips

To maintain smooth operation, the following best practices should be used when diagnosing any issues:

- Use print statements or logging to trace system states, MQTT messages, and Firebase uploads.
- Test each module individually (Camera, MQTT, Firebase) before full integration.
- Enable verbose mode for Firebase and MQTT SDKs for detailed debugging.
- Verify TensorFlow model file paths and version compatibility.
- Re-test after library or API updates to prevent incompatibility issues.

Following these debugging practices maintains FarmEye's reliability and minimizes downtime.

9. Maintenance

9.1 Updating Models or Scripts

To maintain FarmEye's accuracy over time, AI models and backend scripts should be regularly updated.

- Retrain models using new greenhouse image data to improve accuracy.

- Validate updated models before redeployment to the Raspberry Pi.
- Use clear versioning (e.g., model_v2.h5) for easy tracking.
- Review and update Python scripts to remain compatible with TensorFlow and Firebase updates.
- Test all changes in a controlled environment before live deployment.

9.2 Version Control & Backup Procedures

Version control and data backup play a critical role in maintaining the system's stability and protecting valuable project assets:

- Store all source code, models, and configurations in a GitHub repository using branching for safe updates.
- Schedule regular backups for Firebase Storage, Oracle Apex databases, and Raspberry Pi configurations.
- Create system images or clones of the Pi's SD card for quick recovery.

Strong version control and backup routines ensure FarmEye remains stable, recoverable, and scalable for future enhancements.

10. Conclusion

The FarmEye deployment successfully integrates AI-powered edge detection with scalable cloud infrastructure and a real-time farmer dashboard. This modular pipeline ensures accurate crop monitoring, reliable data synchronization, and easy future updates through Firebase and Oracle Apex. By combining AI inference on the Raspberry Pi with a cloud-connected Flutter interface, FarmEye delivers a practical, end-to-end smart agriculture solution ready for greenhouse and field deployment.

11. Appendix

11.1 Command Reference Summary

Purpose	Command
Update Raspberry pi packages	sudo apt update & sudo apt upgrade -y
Enable camera interface	sudo raspi-config
Install Python dependencies	pip install -r requirements.txt
Run main application	python3 main.py
Start MQTT broker service	sudo systemctl start mosquitto
View active MQTT topics	mosquitto_sub -t “#” -v
Test Firebase connection	python3 test_firebase.py
Check model loading	python3 verify_model.py

11.2 Example Configuration Files (config.py, firebase.json)

- Config.py:

```
MQTT_BROKER = "broker_ip"
MQTT_PORT = 1883
MQTT_TOPIC = "farmeye/data"
FIREBASE_BUCKET = "farmeye-app.appspot.com"
FIREBASE_KEYFILE = "firebase-key.json"
MODEL_PATHS = {
  "growth_stage": "/home/pi/models/growth_stage_model.h5",
  "pests": "/home/pi/models/pest_model.h5",
  "diseases": "/home/pi/models/disease_model.h5"
}
```

- Firebase.json

```
{
  "project_id": "farmeye-app",
  "storage_bucket": "farmeye-app.appspot.com",
  "database_url": "https://farmeye-app.firebaseio.com"
}
```

These configuration files define key system parameters, cloud credentials, and local model required for smooth operation.

11.3 Deployment Checklist

Before finalizing deployment, ensure the following steps are completed:

Hardware setup:

1. Raspberry Pi powered and connected to stable Wi-Fi.
2. Camera module installed, configured and enabled.
3. All peripheral devices tested and functional.

Software setup:

1. Python dependencies installed.
2. AI models loaded into the /models directory.
3. MQTT broker configured and verified.
4. Firebase credentials correctly set in configuration files.

System verification:

1. Live detection and image capture tested successfully.
2. Images and metadata uploading to firebase.
3. Flutter app retrieving data and displaying predictions.
4. Oracle Apex integration confirmed with synchronized data.

Maintenance Preparedness:

1. Git repository updated and backed up.
2. Regular model retraining schedule established.
3. Firebase and Oracle Apex data backups configured.

Following this checklist ensures the crop development vision system is fully functional, stable, and ready for real-world deployment in ARC's greenhouse environment.