

Design Document and Project Plan  
Earth Observation and Blockchain Integration with Syngenta Foundation's  
Agri-Entrepreneur Program

Project Swaminathan

May 30th, 2025

# Contents

<b>1</b>	<b>Overview</b>	<b>4</b>
1.1	Data Flow Architecture . . . . .	4
<b>2</b>	<b>User Interface (UI) Design</b>	<b>7</b>
2.1	Core Objectives . . . . .	7
2.2	Technical Specification . . . . .	7
2.3	Screen Reference . . . . .	8
2.4	User Management . . . . .	19
<b>3</b>	<b>DApp API Documentation</b>	<b>25</b>
<b>4</b>	<b>Data Structure and Blockchain Interaction</b>	<b>26</b>
4.1	Key Components . . . . .	26
4.2	User Ecosystem . . . . .	26
4.3	Oracle Datum Structure . . . . .	26
4.4	Reference UTxO Oracle Architecture . . . . .	26
4.5	DID NFT Minting Policy . . . . .	27
4.6	Signed Message Oracle Architecture . . . . .	27
4.7	Integration Strategy . . . . .	27
4.8	Blockfrost Configuration . . . . .	28
4.9	API Endpoints for Oracle Integration . . . . .	28
<b>5</b>	<b>Gamma Earth Satellite Integration</b>	<b>28</b>
5.1	S2DR3 RISC API – Version 0.2 . . . . .	28
5.2	Application Implementation . . . . .	29
<b>6</b>	<b>Testing Plan</b>	<b>30</b>
6.1	Datum Integrity Tests . . . . .	30
6.2	Script Unit Tests . . . . .	30
6.3	Application Unit Testing . . . . .	32
<b>7</b>	<b>Security Testing Approach</b>	<b>32</b>
7.1	Backend Security Measures . . . . .	32
7.2	Security Testing Checklist . . . . .	33
7.3	Frontend Security Measures . . . . .	34
<b>8</b>	<b>Scalability Measurement Criteria</b>	<b>34</b>
8.1	Performance & Memory Optimizations (React Native) . . . . .	34
<b>9</b>	<b>Deployment Plan</b>	<b>36</b>
9.1	Deployment Overview . . . . .	36
9.2	Deployment Environments . . . . .	36
9.3	Frontend Deployment (React Native + Expo) . . . . .	36

9.4	Backend Deployment (Node.js/Express) . . . . .	36
9.5	Database Setup . . . . .	37
9.6	Single Sign-On (SSO) Integration . . . . .	37
9.7	Rollback Strategy . . . . .	37
<b>10</b>	<b>Project Plan and Implementation Timeline</b>	<b>37</b>

# 1 Overview

This document outlines the design and project plan for a proof-of-concept (PoC) oracle system built on the Cardano blockchain to empower smallholder farmers in India. The system integrates earth observation, farm, and market data to facilitate trusted data exchange among farmers, Agri-Entrepreneurs (AEs), buyers, and government agencies.

The solution leverages the Cardano blockchain for immutable data storage, Gamma Earth satellite APIs for crop insights, and provides a comprehensive mobile-first experience for agricultural entrepreneurs.

## 1.1 Data Flow Architecture

### Authentication Flow

- User enters credentials in mobile app
- Backend validates against MongoDB user collection
- JWT token issued and stored securely (Keychain/Keystore)
- Token used for all subsequent API requests

### Farmer Registration Flow

- Agent submits farmer data through mobile app
- Backend saves farmer record to MongoDB
- Cardano transaction created with metadata:

---

```
1 {  
2   "farmerId": "123",  
3   "farmerName": "John Doe",  
4   "timestamp": "2025-05-28T10:00:00Z"  
5 }
```

---

- Transaction hash stored for audit trail

### Field Registration Flow

- Agent submits field polygon (GeoJSON format)
- MongoDB stores with 2dsphere geospatial indexing
- Optional blockchain transaction for field verification
- Integration with Gamma Earth API for satellite data

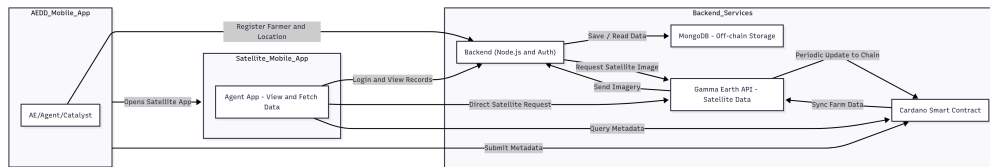
## Crop Insights Flow

- System queries Gamma Earth API with field coordinates
- Receives NDVI data, crop health metrics, and satellite imagery
- Data processed and cached in MongoDB
- Results displayed in mobile app with visual analytics

## View Registered Farmer Data Flow

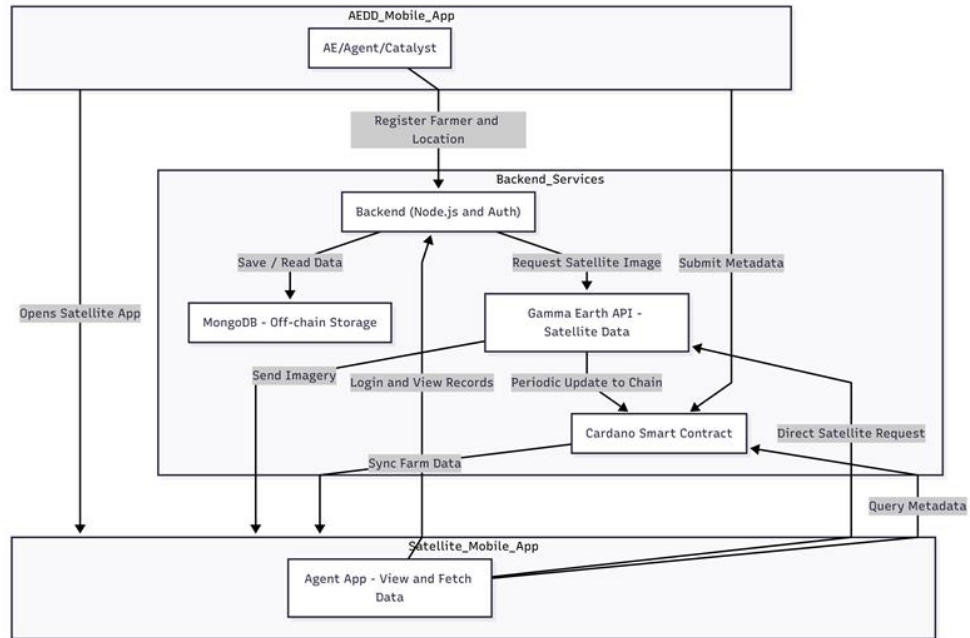
- Agent/AE/Catalyst requests a list of registered farmers or fields
- Backend retrieves this data directly from MongoDB and returns it to the app
- No blockchain interaction required

## Data Flow Diagram

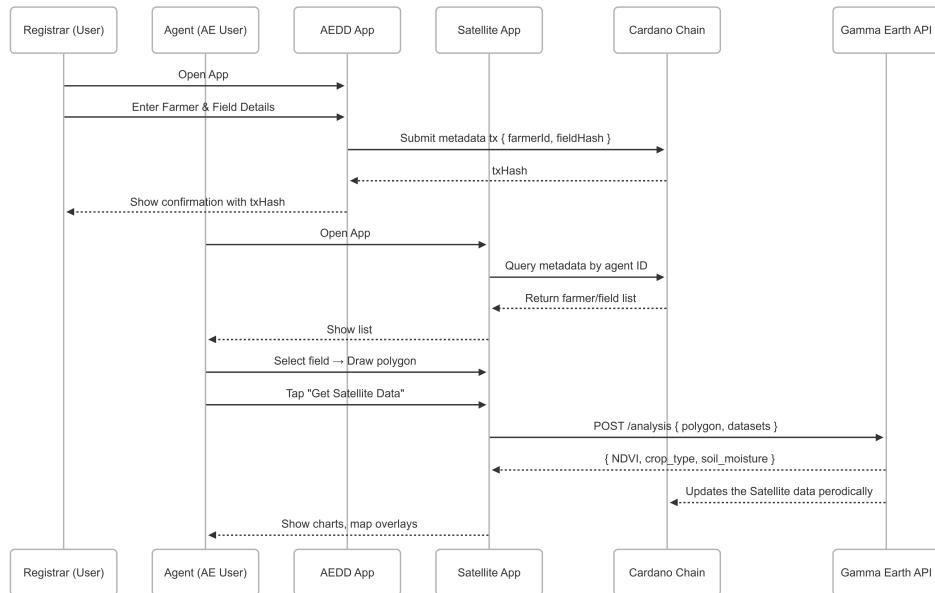


## High Level Diagram

### High Level Diagram for Syngenta Satellite App



## Low Level Diagram



## 2 User Interface (UI) Design

### 2.1 Core Objectives

The system is designed to provide real-time farm metrics stored on-chain using the Cardano blockchain, enabling transparent and immutable agricultural data management for farmers, admins, and agricultural entrepreneurs (AEs).

### 2.2 Technical Specification

This section outlines the technical requirements and integration design for the project leveraging the Cardano blockchain.

- **UI Mockups:** Provided by design team (Figma link to be added)
- **Implementation:**
  - React Native with EXPO Framework
  - Mobile-first design for Agent/AE/Catalyst workflows

## 2.3 Screen Reference

AEDD application with Oracle Satellite app deep linking

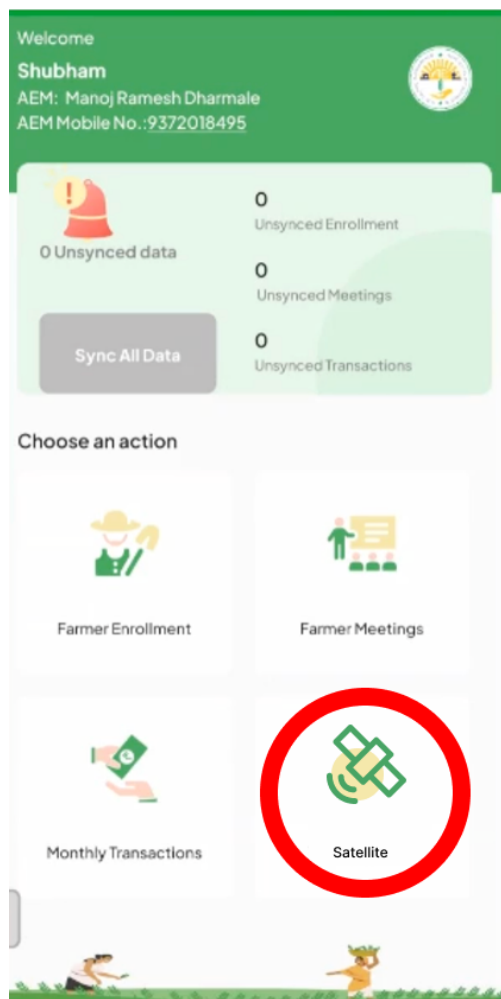


Figure 1: AEDD application with Oracle Satellite app deep linking



## App splash screen

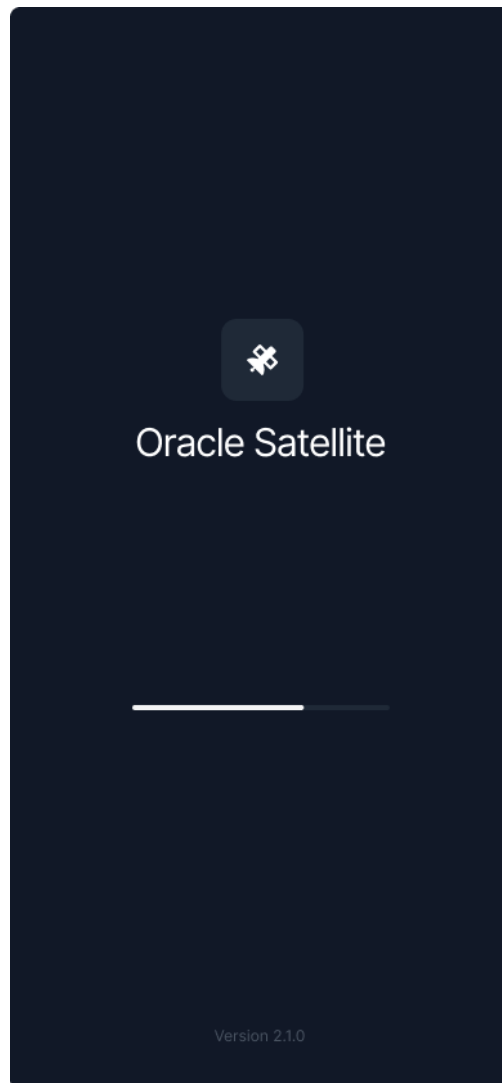


Figure 2: App splash screen

## FieldView search screen

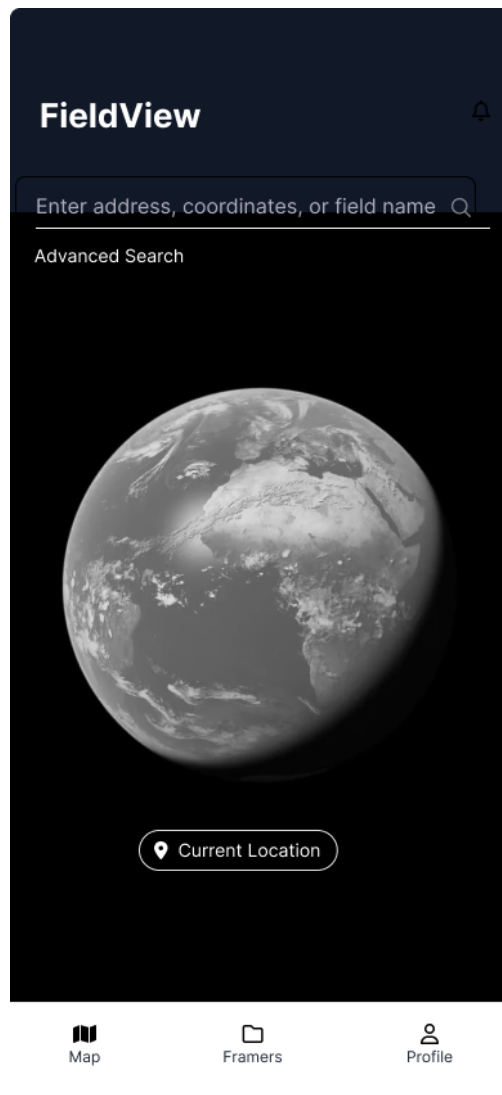
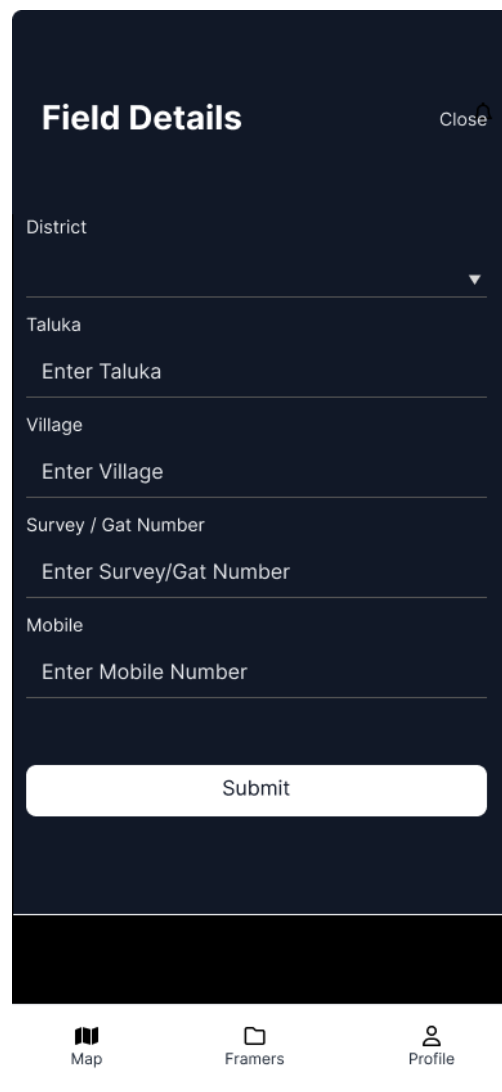


Figure 3: FieldView search screen

## Detailed search filters



The image shows a mobile application interface for 'Field Details'. The form is set against a dark blue background. At the top, the title 'Field Details' is in white, with a 'Close' button and a back arrow icon to its right. Below the title, there are five input fields, each with a label and a placeholder text: 'District' (with a dropdown arrow), 'Taluka' (placeholder: 'Enter Taluka'), 'Village' (placeholder: 'Enter Village'), 'Survey / Gat Number' (placeholder: 'Enter Survey/Gat Number'), and 'Mobile' (placeholder: 'Enter Mobile Number'). A white 'Submit' button is positioned below the inputs. At the bottom of the screen is a navigation bar with three icons and labels: 'Map' (with a map icon), 'Framers' (with a folder icon), and 'Profile' (with a person icon).

**Field Details** Close

District ▼

Taluka  
Enter Taluka

Village  
Enter Village

Survey / Gat Number  
Enter Survey/Gat Number

Mobile  
Enter Mobile Number

**Submit**

**Map** **Framers** **Profile**

Figure 4: Detailed search filters

## Interactive map display

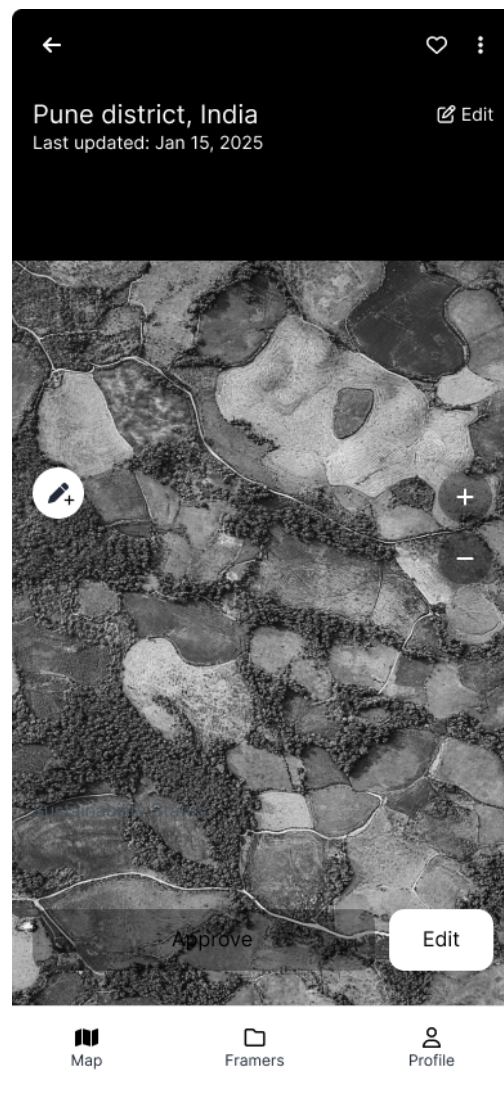


Figure 5: Interactive map display

## Boundary drawing tool



Figure 6: Boundary drawing tool

## Precision editing interface

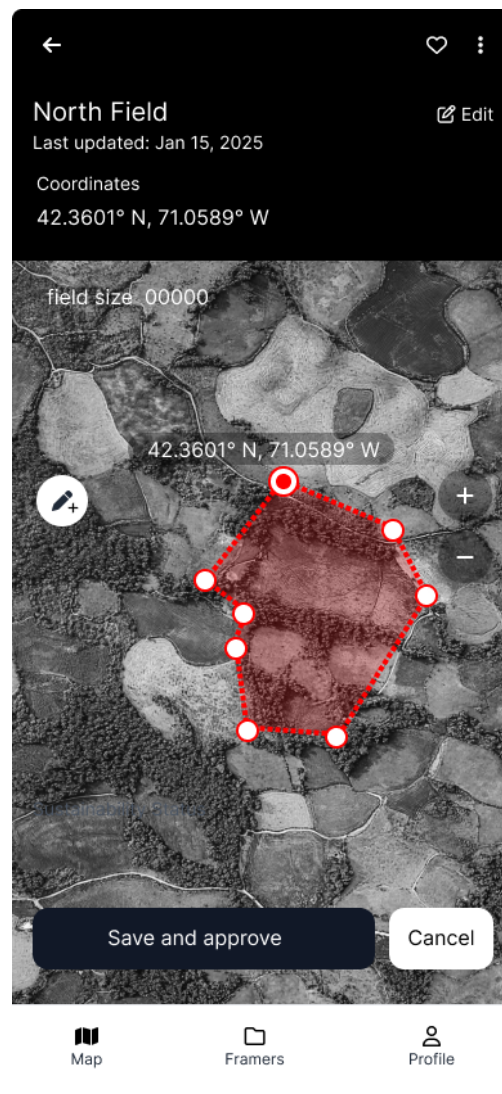


Figure 7: Precision editing interface

## Shape modification screen

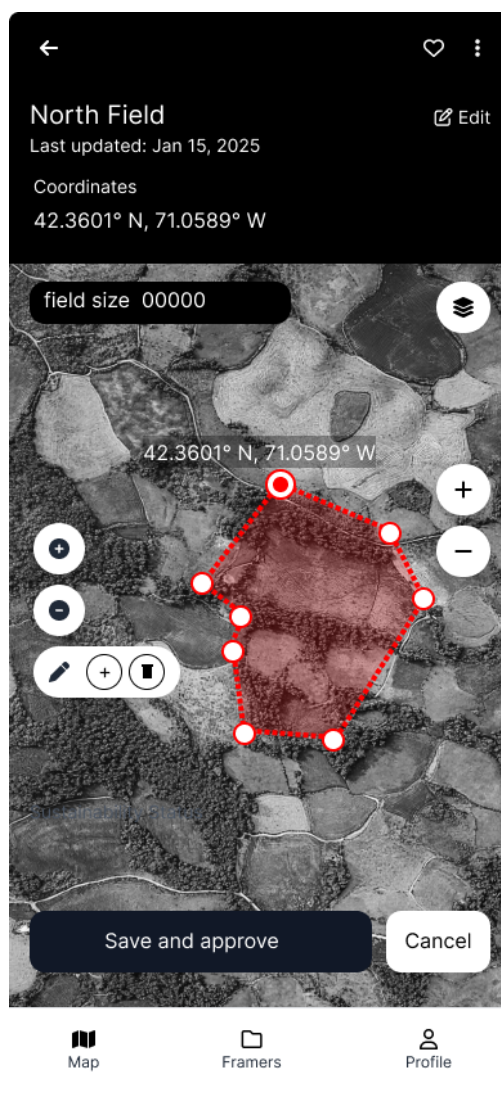


Figure 8: Shape modification screen

## Data export options

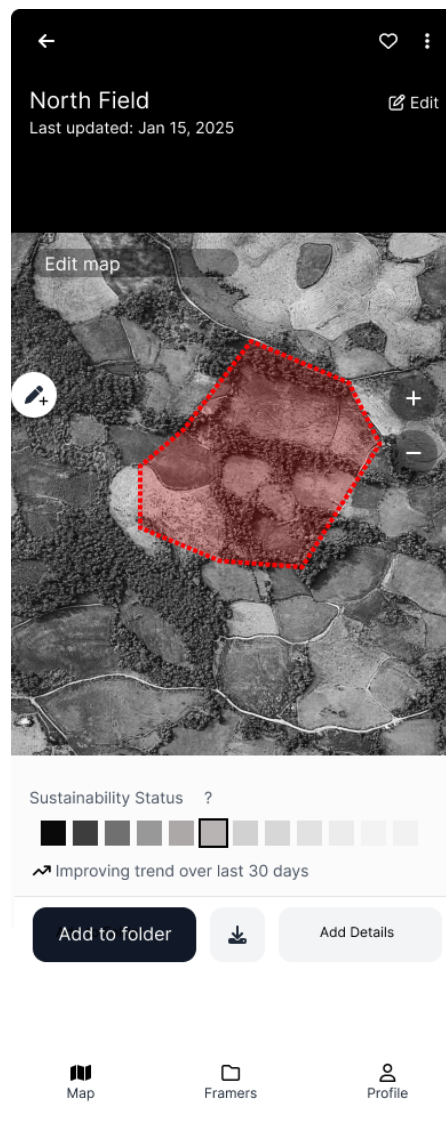


Figure 9: Data export options



## Extended metadata entry

Land Use & Crop Details

×

Current Crop

e.g., wheat, cotton

Sowing DateHarvest Date

mm/dd/yyyymm/dd/yyyy

Previous Crops (last 2–3 seasons)

List previous crops

Cropping Pattern

Fertilizer or Input Usage

Describe inputs used

Environmental & Soil Data

Irrigation Type

Soil Type

Known Soil Health Issues or Ratings

Describe if any

Water Source / Availability

e.g., Well, River, Uncertain

Save DetailsCancel

Figure 10: Extended metadata entry

Field details screen

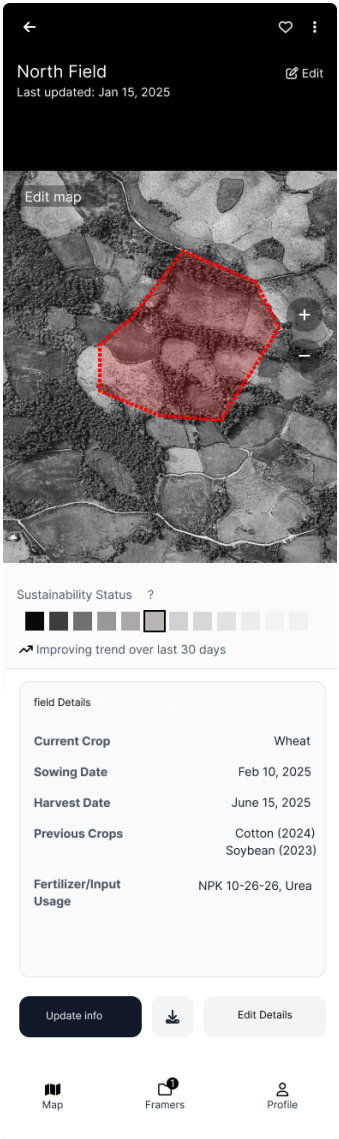


Figure 11: Field details screen

## 2.4 User Management

### Agricultural property dashboard



Figure 12: Agricultural property dashboard

## Registered farmer list

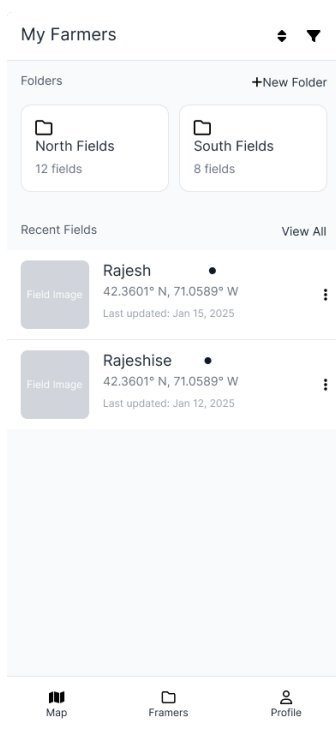


Figure 13: Registered farmer list

Individual farmer details

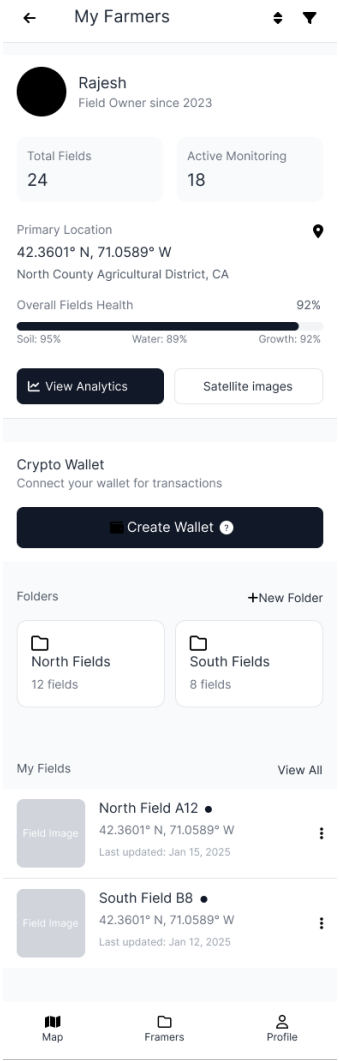


Figure 14: Individual farmer details

## Cardano wallet authentication

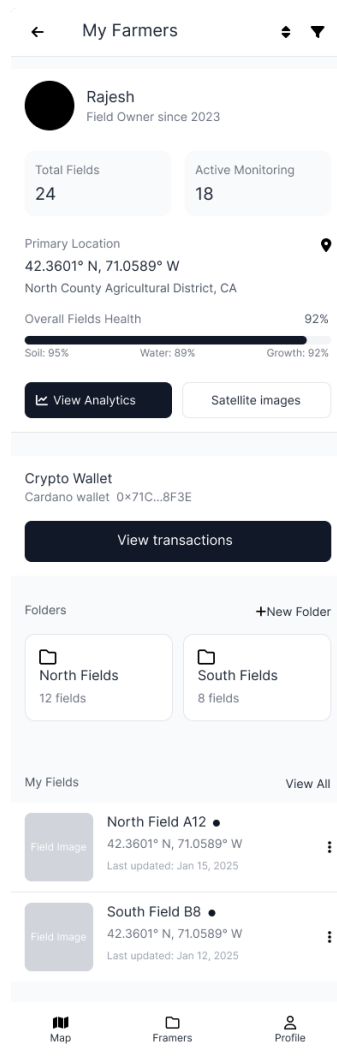


Figure 15: Cardano wallet authentication

## Digital asset interface

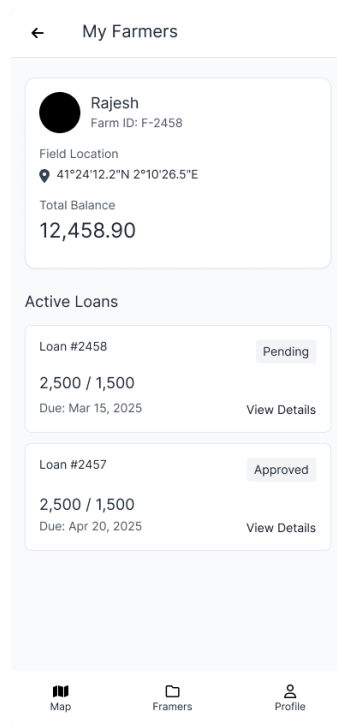


Figure 16: Digital asset interface

## Agri Entrepreneur profile screen

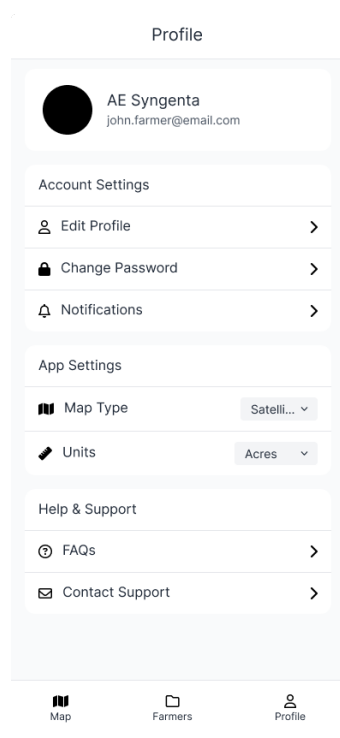


Figure 17: Agri Entrepreneur profile screen



### 3 DApp API Documentation

This section details the REST API endpoints for managing administrators, farmers, fields, and crop data.

Task	Method	Path	Description
Post Admin	POST	/admins/	Create a new admin
Admin Login	POST	/admins/login	Authenticate admin and return token
Get All Admins	GET	/admins/	Retrieve all admins
Get Admin by Id	GET	/admins/{id}	Fetch details of a specific admin
Update Admin	PUT	/admins/{id}	Update admin data
Post Farmer Details	POST	/farmers/	Register new farmer
Get Farmers by Registor	GET	/farmers/registor/{regId}	Retrieve farmers registered under a specific admin
Get All Farmers	GET	/farmers/	Retrieve all farmer entries
Get Farmer by Id	GET	/farmers/{id}	Retrieve details for a specific farmer
Update Farmer Details	PUT	/farmers/{id}	Update existing farmer data
Post Field	POST	/fields/	Register a new field
Get All Fields	GET	/fields/	List all field entries
Get Field by Id	GET	/fields/{id}	Retrieve specific field info
Update Field Details	PUT	/fields/{id}	Update an existing field
Post Crop Data	POST	/cropData/	Submit crop-related data for a field
Get All Crop Data	GET	/cropData/	Retrieve all stored crop datasets
Get Crop Data by Id	GET	/cropData/{id}	Fetch specific crop dataset
Update Crop Data by Id	PUT	/cropData/{id}	Modify an existing crop dataset

## 4 Data Structure and Blockchain Interaction

### 4.1 Key Components

Component	Technology	Purpose
Mobile Frontend	React Native + Expo	User interface for data entry and visualization
Backend API	Node.js + Express	Business logic and blockchain integration
Database	MongoDB	Application data storage with geospatial indexing
Blockchain	Cardano + Blockfrost API	Immutable data verification and audit trail
Satellite Data	Gamma Earth API	Real-time crop health and NDVI analytics

### 4.2 User Ecosystem

- **Agents/AEs/Catalysts:** Register farmers and manage field data
- **Farmers:** Benefit from crop insights and blockchain-verified records
- **System Administrators:** Monitor and maintain the platform

### 4.3 Oracle Datum Structure

#### Datum Fields

- **Farm Land Area (Integer):** Represents the area in square yards.
- **IPFS Hash for Farm Borders (ByteString):** Points to farm boundary file.
- **Arbitrary Data (BuiltinData):** Flexible metadata (e.g., soil PH, crop type).

Listing 1: OracleDatum Schema

```
data OracleDatum = OracleDatum
{ farmArea :: Integer
, ipfsHash :: BuiltinByteString
, arbitraryData :: BuiltinData
}
```

### 4.4 Reference UTxO Oracle Architecture

- Uses CIP-68 NFTs: user-token for farm parcel, reference-token for oracle UTxO.
- Validator ensures only authorized oracle providers can update oracle UTxOs.

```
issuanceOperator :: PubKeyHash
issuanceOperator = ...
```

```
farmParcelOracleProviders :: [PubKeyHash]
farmParcelOracleProviders = [...]
```

#### 4.5 DID NFT Minting Policy

- Mints unique user-token and reference-token pair per parcel.
- Reference-token sent to validator with valid datum.
- Enforced by operator signature.

#### 4.6 Signed Message Oracle Architecture

- Oracle backend signs CBOR-encoded oracle data.
- Contracts verify signature, timestamps, and public key match.
- Cost-effective, frequent updates, no on-chain state needed.

##### Advantages

- Low cost
- Real-time updates
- Concurrent consumption

##### Trade-offs

- No native data availability
- High off-chain responsibility

#### 4.7 Integration Strategy

```
data OracleDatum = OracleDatum
  { farmArea :: Integer
  , ipfsHash :: BuiltinByteString
  , arbitraryData :: BuiltinData
  }
```

```
interface OracleDatum {
  farmArea: number;
  ipfsHash: string;
  arbitraryData: any;
}
```

## 4.8 Blockfrost Configuration

```
import { Lucid, Blockfrost } from "lucid-cardano";
const lucid = await Lucid.new(
  new Blockfrost("https://cardano-mainnet.blockfrost.io/api/v0", process.env.I
    "Mainnet"
  );
```

## 4.9 API Endpoints for Oracle Integration

Endpoint	Method	Description
/api/oracle/:address	GET	Fetch latest OracleDatum
/api/oracle/history/:address	GET	Historical oracle data
/api/oracle/verify/:txHash	GET	Verify specific transaction

# 5 Gamma Earth Satellite Integration

## 5.1 S2DR3 RISC API – Version 0.2

**Note:** The API is asynchronous. A POST request launches the job and immediately returns metadata. Processed imagery appears in a Google Cloud bucket a few minutes later.

### A. Job Submission Endpoint

Listing 2: Job Request Payload

```
1 POST https://s2dr3-job-20250428-862134799361.europe-west1.run.
   app/{USER_ID}
2
3 {
4   "date": "2023-05-01",
5   "aoi": "19.93 49.28 20.00 49.35"
6 }
```

## Response Example

---

```
1 {  
2   "ISO": "UA",  
3   "MGRS": "35UQR",  
4   "PID": "T35UQR-20230810-u8600146",  
5   "aoi_overlap": "1.0",  
6   "bbox": "30.85 50.39 30.88 50.42",  
7   "date": "20230810",  
8   "job_id": "e26bb408-d330-11ef",  
9   "save_path_MS": "..._MS.tif",  
10  "save_path_TCI": "..._TCI.tif"  
11 }
```

---

## B. Check Job Status

GET [https://s2dr3-job-20250428-862134799361.europe-west1.run.app/{USER\\_ID}/{job\\_id}](https://s2dr3-job-20250428-862134799361.europe-west1.run.app/{USER_ID}/{job_id})

---

```
1 {  
2   "PID": "T35UQR-u8600146-20230810",  
3   "State": "completed",  
4   "jobID": "e26bb408-d330-11ef"  
5 }
```

---

## C. Tile Server Example URLs

- NDVI, TCI, and IRP overlays provided as dynamic tiles via AWS Lambda tile server.

[https://.../tiles/WebMercatorQuad/{z}/{x}/{y}@2x?url=s3://sentinel-s2dr3/{MGRS}/{job\\_id}/{aoi}.tif](https://.../tiles/WebMercatorQuad/{z}/{x}/{y}@2x?url=s3://sentinel-s2dr3/{MGRS}/{job_id}/{aoi}.tif)

## D. Download Raw GeoTIFFs with wget

wget [https://.../S2L2Ax10\\_T34UDV-20240501-ucc1a562\\_IRP.tif](https://.../S2L2Ax10_T34UDV-20240501-ucc1a562_IRP.tif)

## 5.2 Application Implementation

### A. Submit a Job (React Native)

```

export async function submitSatelliteJob(userId, date, aoi) {
  const url = `https://.../${userId}`;
  const payload = { date, aoi };
  const res = await fetch(url, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload),
  });
  return await res.json();
}

```

## B. Poll Job Status

```

export async function getJobStatus(userId, jobId) {
  const url = `https://.../${userId}/${jobId}`;
  const res = await fetch(url);
  return await res.json();
}

```

## 6 Testing Plan

To ensure the correctness, reliability, and upgradability of the oracle system, a comprehensive unit testing strategy is defined across all smart contract components, datum structures, and integration workflows.

### 6.1 Datum Integrity Tests

- **Signed Message Structural Integrity:** Validate that serialized oracle datum, timestamps, and concatenated data conform to the required format.
- **Precision Integrity:** Ensure accurate arithmetic with scaled land area values (e.g.,  $500 * 1,000,000$ ).
- **Arbitrary Data Robustness:** Test 'arbitraryData' field with various Plutus-compatible structures.
- **Boundary Validation:** Validate minimum and maximum field lengths and land area values.

### 6.2 Script Unit Tests

#### Oracle Management Validator

Positive Tests:

- Authorized oracle provider updates with valid structure.
- UTXO continuity preserved.

**Negative Tests:**

- Unauthorized spending attempts.
- Invalid or missing output datum.
- Tampered reference-token handling.

**DID NFT Minting Policy**

**Positive Tests:**

- Valid issuance with operator signature.
- Proper initialization of oracle UTXO.

**Negative Tests:**

- Missing signature.
- Duplicate or malformed tokens.
- Missing or invalid oracle output.

**Signed Message Validation**

**Positive Tests:**

- Signature matches expected key.
- Valid timestamp range.

**Negative Tests:**

- Invalid signature.
- Expired or invalid timestamps.
- Data mismatch.

## 6.3 Application Unit Testing

### Backend

**Testing Stack:** Jest, Supertest

```
describe('Oracle Data API', () => {
  test('GET /api/oracle/:address - valid', async () => {
    const res = await request(app).get('/api/oracle/addr_test123');
    expect(res.body).toHaveProperty('farmArea');
  });
});
```

### Blockchain Integration

```
test('should decode OracleDatum', () => {
  const decoded = parseOracleDatum('hex_data');
  expect(decoded.farmArea).toBe(100);
});
```

### Frontend

**Testing Stack:** Jest, React Testing Library

```
test('renders FarmCard with data', () => {
  const { getByText } = render(<FarmCard data={{ farmArea: 42, ndvi: 0.67 }} />);
  expect(getByText('42 hectares')).toBeTruthy();
});
```

## 7 Security Testing Approach

### 7.1 Backend Security Measures

#### Authentication & Authorization

```
const jwt = require('jsonwebtoken');
const rateLimit = require('express-rate-limit');

app.use('/api/', rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100,
  message: 'Too many requests'
}));
```



## Input Validation

```
const { check, validationResult } = require('express-validator');

app.post('/api/farmers', [
  check('name').isLength({ min: 2, max: 50 }).escape(),
  check('email').isEmail().normalizeEmail(),
  check('phone').isMobilePhone(),
], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }
});
```

## Blockchain Security

```
const validateCardanoAddress = (address) => {
  const mainnet = /^addr1[a-z0-9]{98}$/;
  const testnet = /^addr_test1[a-z0-9]{98}$/;
  return mainnet.test(address) || testnet.test(address);
};
```

## 7.2 Security Testing Checklist

### API Security Tests

- Input validation: SQL injection, XSS, command injection
- JWT token validation, session management
- Role-based access control
- Rate limiting and DDoS protection
- HTTPS and SSL certificate validation
- CORS configuration

### Database Security Tests

- Role-based access control
- Encryption of data at rest and in transit
- Secure backup and audit logs
- SSL database connections and firewalls

## Blockchain Security Tests

- Cardano address validation
- Secure private key management
- Input validation in smart contract interfaces

## 7.3 Frontend Security Measures

### Secure Storage

```
import * as SecureStore from 'expo-secure-store';

const storeToken = async (token) => {
  await SecureStore.setItemAsync('authToken', token, {
    keychainService: 'agritech-app',
    encrypt: true
  });
};
```

### Network Security

```
const API_BASE_URL = __DEV__
  ? 'https://dev-api.agritech.com'
  : 'https://api.agritech.com';

const secureAxios = axios.create({
  baseURL: API_BASE_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json'
  }
});
```

## 8 Scalability Measurement Criteria

### 8.1 Performance & Memory Optimizations (React Native)

To ensure smooth performance and efficient memory usage in the React Native mobile application, the following best practices are implemented:

### **Efficient List Rendering**

- Use `FlatList` or `SectionList` for rendering large datasets.
- Implement `getItemLayout` for optimal scroll performance.

### **Hermes Engine**

- Hermes compiles JavaScript to bytecode ahead-of-time.
- Benefits include faster app startup, lower memory usage, and reduced runtime overhead.

### **Lazy Loading**

- Use `React.lazy()` and `Suspense` to defer loading of heavy components.
- Keeps the initial bundle small to improve cold start time.

### **Render Optimization**

- Use `React.memo`, `useMemo`, `useCallback` to avoid unnecessary re-renders.
- Favor immutable data structures to simplify change detection.

### **Image Optimization**

- Compress and resize images before bundling.
- Use modern formats like WebP.
- Employ `react-native-fast-image` for efficient caching.
- Avoid animating large images on the JS thread.

### **Memory Management**

- Clean up timers, API calls, and subscriptions in component lifecycle.
- Monitor app with React Native's Performance Monitor to detect frame drops and memory spikes.

Together, these techniques ensure the UI remains responsive at 60 FPS and supports growing datasets with minimal memory impact.

## 9 Deployment Plan

### 9.1 Deployment Overview

The deployment plan outlines the strategies to deliver and maintain the mobile application across development, staging, and production environments. The stack includes React Native (Expo) for the frontend and Node.js with Express for the backend. MongoDB and PostgreSQL are used for data storage.

### 9.2 Deployment Environments

- **Development Environment**
  - Local Expo CLI and Docker-based backend
  - Internal testing only
- **Production Environment**
  - Expo EAS build for iOS/Android (App Store / Google Play)
  - Node.js hosted on cloud (e.g., AWS)
  - Managed MongoDB and PostgreSQL

### 9.3 Frontend Deployment (React Native + Expo)

- **Development Build:** Via Expo Go QR scanning
- **Production Build:**
  - Expo EAS Build
  - EAS Submit to:
    - \* App Store Connect (iOS)
    - \* Google Play Console (Android)

### 9.4 Backend Deployment (Node.js/Express)

- Docker containerization
- CI/CD with GitHub Actions
- Hosted on AWS EC2 or similar

## 9.5 Database Setup

- **MongoDB:**
  - Dev: local or MongoDB Atlas free tier
  - Prod: Atlas with backup and scaling
- **PostgreSQL:**
  - Dev: Local or Docker
  - Prod: Managed PostgreSQL (e.g., AWS RDS)

## 9.6 Single Sign-On (SSO) Integration

- OAuth2/OpenID Connect planned
- Secure session/token sharing via API handshake or deep linking
- Requires identity model upgrade and collaboration with parent app team

## 9.7 Rollback Strategy

- **Frontend:** Use EAS Update rollback
- **Backend:** Use last known Docker image
- **Database:** Point-in-time recovery (MongoDB Atlas / RDS)

# 10 Project Plan and Implementation Timeline

The overall implementation of the Satellite Oracle and digital service infrastructure has been structured across multiple phases. This phased design ensures feasibility in field settings, continuous feedback loops with stakeholders, and a smooth progression from technical development to full-scale deployment.

To enhance clarity and readability, the Gantt chart has been divided into two parts:

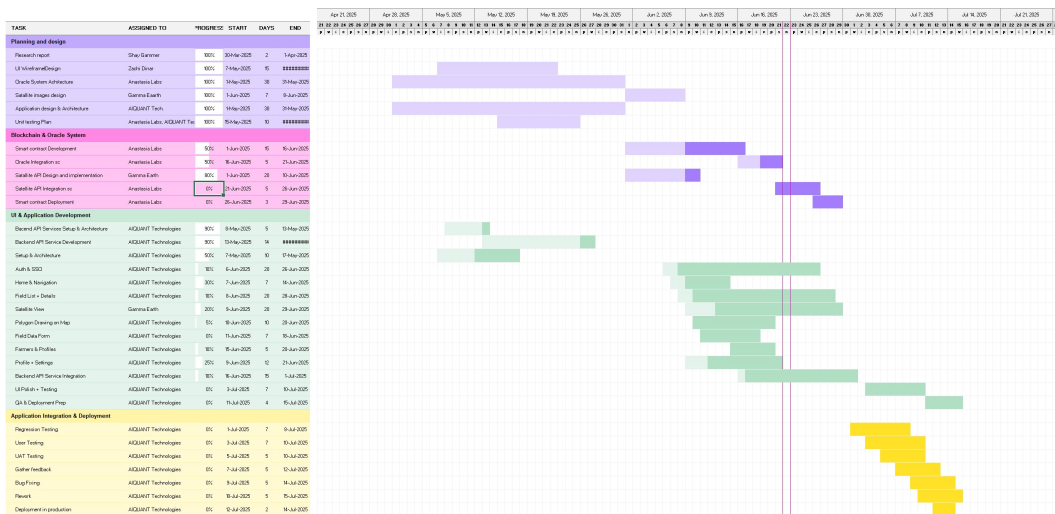


Figure 18: Project Timeline – Part 1: Planning, Development, and Integration

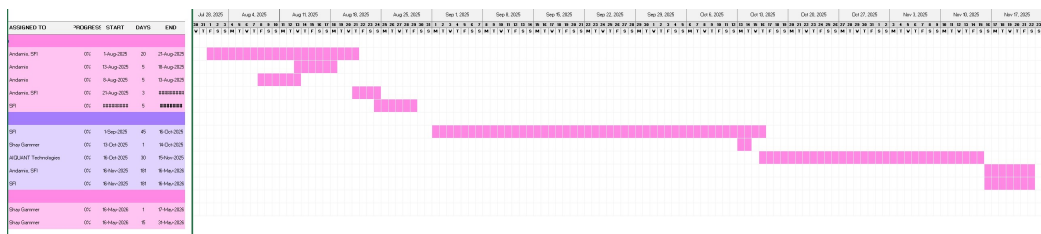


Figure 19: Project Timeline – Part 2: Rollout, Training, and Scaling

Part 1 outlines the foundational phases of the project, including:

- Technical architecture and wireframe design
- Oracle system integration and blockchain component prototyping
- Full-stack application development and UI deployment
- Preparation for integration and pilot testing

Part 2 transitions into:

- Training rollouts via Andamio Learn-to-Work Platform
- Field testing and user feedback loops
- Long-term sustainability monitoring and scaling phases

This plan allows for iterative feedback from the Agri-Entrepreneurs and partner institutions, while de-risking the final deployment through early validation and targeted adjustments.