



COMSATS University Islamabad (CUI)

**Software Design Description
(SDS DOCUMENT)**

for

AgroTech

Version 1.0

By

Mohammad Ammar Ali CIIT/FA21-BSE-042/ISB

Muhammad Moiz CIIT/FA21-BSE-044/ISB

Muaaz Bin Mukhtar CIIT/FA21-BSE-045/ISB

Supervisor

Dr. Saira Beg

Bachelor of Science in Software Engineering (2021-2025)

Table of Contents

1. Introduction.....	1
1.1 Scope	1
1.2 Modules	1
1.2.1 User Profiling	1
1.2.2 Soil Analysis	1
1.2.3 Climate Analysis	1
1.2.4 Crop Recommendation.....	2
1.2.5 AgroTech Chatbot (AI powered assistant).....	2
1.2.6 Crop Health Monitoring	2
1.2.7 Yield Estimation and Prediction	2
1.2.8 Crop Maturity Assessment	2
1.2.9 Harvesting Assist	2
1.2.10 Transportation Assist	2
1.2.11 Storage Assist.....	3
1.2.12 Online Marketplace.....	3
1.2.13 Reports and Analytics	3
2. Design Methodology and Software Process Model	3
2.1 Design Methodology	3
2.2 Software Process Methodology	3
3. System Overview	4
3.1 Architectural Design.....	4
3.1.1 Box and Line Diagram	4
3.1.2 Architecture Design	5
4. Design Models.....	6
4.1 Activity Diagrams	6
4.1.1 Sign up	6
4.1.2 Sign in	7
4.1.3 Manage Profile	8
4.1.4 Manage Registered Users.....	9
4.1.5 Manage Complaints	10
4.1.6 Manage Reports	11
4.1.7 Analyze Soil.....	12
4.1.8 Analyze Climate.....	13

4.1.9	Crop Recommendation.....	14
4.1.10	AI Chatbot.....	15
4.1.11	Register Complaints	16
4.1.12	Identify Crop Stresses	17
4.1.13	Identify Crop Diseases	18
4.1.14	Estimate Crop Yield.....	19
4.1.15	Crop Health Monitoring.....	20
4.1.16	Crop Maturity Assessment.....	21
4.1.17	Crop Quality Assessment.....	22
4.1.18	Calculate Optimal Harvest Time.....	23
4.1.19	Recommend Harvesting Tools.....	24
4.1.20	Recommend Transport.....	25
4.1.21	Packaging Assist	26
4.1.22	Climate Controlled Storage Reservation.....	27
4.1.23	Crop Quality Grading.....	28
4.1.24	Bidding.....	29
4.1.25	Access Products	30
4.1.26	Upload Product.....	31
4.1.27	Bidding.....	32
4.1.28	Reports and Analytics	33
4.1.29	Feature Products.....	34
4.2	Data Flow Diagrams.....	35
4.2.1	Level 0.....	35
4.2.2	Level 1.....	36
4.2.3	Level 2.....	37
4.3	State Transition Diagrams	43
4.3.1	Sign up	43
4.3.2	Sign in	44
4.3.3	Manage Profile.....	44
4.3.4	Switch Profile.....	45
4.3.5	Crop Maturity Assessment.....	45
4.3.6	Crop Health Monitoring.....	45
4.3.7	Soil Analysis	46
4.3.8	Yield Estimation and Prediction	46
4.3.9	Online Marketplace.....	46
4.3.10	Bidding.....	47

4.3.11	Transportation Assist	47
4.3.12	Manage User Complaints	47
4.3.13	Feedback and Reviews	48
4.3.14	Storage Assist.....	48
4.3.15	Reports and Analytics	48
5.	Data Design.....	49
5.1	Data Dictionary	49
5.2	Data Schemas:	51
5.2.1	User.js.....	51
5.2.2	CropRecommendation.js	52
5.2.3	Complaint.js	52
5.2.4	Report.js	53
5.2.5	ChatBot.js.....	54
6.	Human Interface Design.....	55
6.1	Screen Images.....	55
6.2	Screen Objects and Actions.....	60
7.	Implementation	61
7.1	Algorithm	61
7.1.1	Register User.....	61
7.1.2	Login User.....	62
7.1.3	Google Login	62
7.1.4	Logout User.....	63
7.1.5	Recommend Crop Using Random Forest Classifier:	63
7.2	External APIs/SDKs	66
7.3	User Interface	66
7.4	Deployment	70
8.	Testing and Evaluation.....	70
8.1	Unit Testing	70
8.2	Functional Testing	76
8.3	Business Rules Testing.....	82
8.3.1	Admin Controller	82
8.3.2	AUTH Controller	82
8.3.3	User Controller.....	82
8.4	Integration Testing.....	83
9.	Plagiarism Report	86

Revision History

Name	Date	Reason for changes	Version

Application Evaluation History

Comments (by committee)	Action Taken
Farmers are getting data manually for climate analysis.	System will get climate information automatically through weather API.
Farmers are getting data and entering manually in soil analysis.	Farmers can upload report files and system will extract information from it or farmer can fill form manually provided by system.
Diseases are not specified.	Bacterial, Viral, Powdery Mildew, rust, late/early blight, anthracnose etc.

Supervised by

Supervisor's Name

Signature_____

1. Introduction

AgroTech is a web-based app which helps farmers, owners and customers. This application provides assistance to users throughout the lifecycle of a crop i.e. from recommending a crop to monitoring it and from harvesting it to selling it. It will provide recommendation about optimal crops based on soil and climate data, monitoring crops periodically, detect diseases and pests and recommend pesticides and medicines, optimizing resource allocation, streamline harvest operations, facilitating access to market and pricing decisions. It will provide decision and recommendation support along with training for farmers to continuously improve to meet ever growing agricultural needs. By achieving these goals, our software aims to empower farmers, enhance productivity of crops and facilitate comprehensive market linkage for both farmers and customers.

1.1 Scope

AgroTech is a web-based application. It revolutionizes agricultural practices by empowering farmers with advanced tools and data driven decision making capabilities along with streamlining soil and crop monitoring process, utilization of resources and market place availability. AgroTech includes crop recommendation, monitoring, and marketing. It uses artificial intelligence, machine learning, and image processing to provide efficient crop recommendations based on soil and climate data. The system facilitates crop health monitoring, disease detection, and pest prevention to optimize resource allocation and prevent yield loss. AgroTech includes four main users, Admin, Farmers, Sellers and Customers. Admin holds the highest level of authority and access and can easily add, remove and view anything at any time. Farmers can access the recommendation and monitoring functionalities along with option to buy and sell crops. Customers only have access to marketplace for buying agricultural products and reviewing them. They can also participate in bidding. However, they have to register for Seller profile to provide services. Sellers have access to marketplace to sell their products/services. They can bid their products. Moreover, they can view reviews and reports.

1.2 Modules

1.2.1 User Profiling

FE-1: Sign-up (Select User Type (Ecommerce Admin, Farmer and customer)).
FE-2: Sign-in.
FE-3: Manage Profile (Edit, View, Deactivate Account, Change Password).
FE-4: Recover Account.
FE-5: Social Sign-in (Gmail etc.).

1.2.2 Soil Analysis

FE-1: Analyzes Nitrogen, Phosphorus. Potassium content in soil.
FE-2: Analyzes previous soil record.
FE-3: Farmers can give input according to specific needs to analyze the soil.

1.2.3 Climate Analysis

FE-1: Analyzes seasonal weather.
FE-2: Analyzes climate according to specific location.
FE-3: Fetches weather report from API.

1.2.4 Crop Recommendation

FE-1: Using the reports of soil and climate analysis to recommend crops.
FE-2: Provides different options according to budget.
FE-3: provides pros and cons of different crops on that soil.
FE-4: Provides tutorials for the methods for planting crops.

1.2.5 AgroTech Chatbot (AI powered assistant)

FE-1: Provides assistance according to user queries.
FE-2: Provides educational content, tutorials to help user learn about the system.
FE-3: Provides facility of different channels like email and phone support.

1.2.6 Crop Health Monitoring

FE-1: Identify stresses in crops including water, nutrient and pest infestations stress etc.
FE-2: Identify visual symptoms of diseases in crops and classify them against databases.
FE-3: Provides information about crop health periodically using images for timely intervention to minimize damage.

1.2.7 Yield Estimation and Prediction

FE-1: Utilizes ML models on factors such as temperature, rainfall, soil moisture, and nutrient availability to estimate crop yield.
FE-2: Provides accurate yield prediction for planning and resource allocation.
FE-3: Assesses yield risk factors such as drought, disease outbreaks, and pest infestations, allowing farmers to implement risk mitigation measures.

1.2.8 Crop Maturity Assessment

FE-1: Using image recognition (IR) to assess crop maturity and readiness for harvesting.
FE-2: Determines optimal time for harvesting based on crop growth and maturity indicators.
FE-3: Provides feedback to farmers on crop readiness to optimize harvest scheduling.
FE-4: Assess the Quality of crops based on factors such as sugar content, starch levels, size, color, and overall visual appearance.

1.2.9 Harvesting Assist

FE-1: Provides a scheduling tool to plan and manage harvesting activities, specifying crop types, quantities and preferred dates.
FE-2: Recommends tools and techniques for harvesting.
FE-3: Provides an access of a catalog of harvesting equipment for rent and purchasing.
FE-4: Provides detailed guidelines for proper handling of tools and freshly harvested crops, helping farmers maintain quality of crops.

1.2.10 Transportation Assist

FE-1: Provides a facility of transportation to farmer directly from platform, specifying pickup and delivery locations.
FE-2: Provides option of different recycle and biodegradable packaging material for sustainable transportation practices.
FE-3: Provide shortest route to desired location for fuel efficient transportation, minimizing emissions and transportation costs.
FE-4: Provides facility of logistic planning for scheduling time and routes to deliver.

FE-5: Provides facility of different delivery options based on types and quantity of crops.

FE-6: Provides facility of guide lining to farmer for packaging crops securely for better transportation.

1.2.11 Storage Assist

FE-1: Provides facility to reserve climate-controlled storage facilities through this platform.

FE-2: Provides option to assess crop quality and assign grades providing buyers an accurate information.

FE-3: Provides different training suggestions to improve farmer's grading skills.

FE-4: Provides guidelines for complete process of storage and handling techniques for each quality grade.

1.2.12 Online Marketplace

FE-1: Provides facility to list item with their name, description, price, quality grade and image.

FE-2: Provides facility to manage inventory, track orders through single dashboard.

FE-3: Provides facility to perform digital marketing campaigns directly through platform.

FE-4: Provides description of farming practices used to harvest these crops to highlight their eco-friendly processes.

FE-5: Provides facility to customer to directly buy crops at listed prices.

FE-6: Provides facility to customer to participate in bidding process.

FE-7: Provides facility to farmers to manage inventory levels.

FE-8: Provides secure online payment gateways.

1.2.13 Reports and Analytics

FE-1: System will provide comparative reports and analytics about user engagement (before and after using our tool).

FE-2: System will provide reports containing various terms of webpage (visits, web pages visited at max etc.).

FE-3: System will provide reports about sales and revenues. (total crop sale, total pesticides and fertilizers sale, total renting equipment).

FE-4: System will provide user with various filters for user to filter out required data.

2. Design Methodology and Software Process Model

2.1 Design Methodology

AgroTech will be developed using **Procedural Programming Design Methodology**. We will be using MERN stack and this approach aligns well with it. This methodology simplifies processes such as user management, recommendations and analysis by step-by-step execution. This method ensures that code is organized, making it easier to debug, maintain and integrate with middleware.

2.2 Software Process Methodology

Incremental Process Methodology will be used for AgroTech development due to its cost efficiency and flexibility. Milestones will be achieved in iterations and it is easy to change requirements. At each iteration, some functionality will be delivered. Furthermore, debugging and testing is done in every iteration improving the overall quality of the system.

3. System Overview

AgroTech is an entirely new product. Below is its architectural diagram.

3.1 Architectural Design

3.1.1 Box and Line Diagram

This diagram illustrates the interconnected modules of AgroTech, focusing on user profiling, crop recommendation, and supporting services like soil and climate analysis, health monitoring, AI chatbot, and marketplace functionalities, aimed at enhancing agricultural productivity and decision-making.

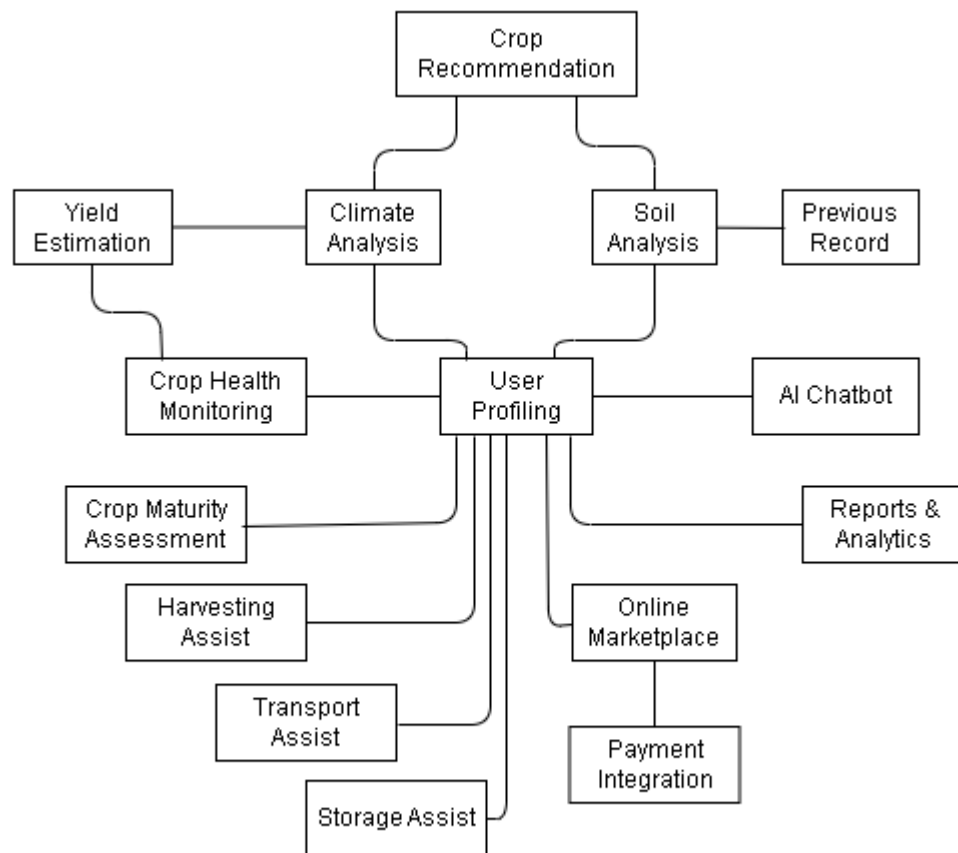


Figure 1: Box and Line Diagram

3.1.2 Architecture Design

This diagram shows AgroTech's architecture, where the **Controller** processes user requests, updates the **Model**, and retrieves data. The **View** displays this data to users. The system uses **MongoDB** for storage, with APIs managed through **Node** and **Fast API Servers**.

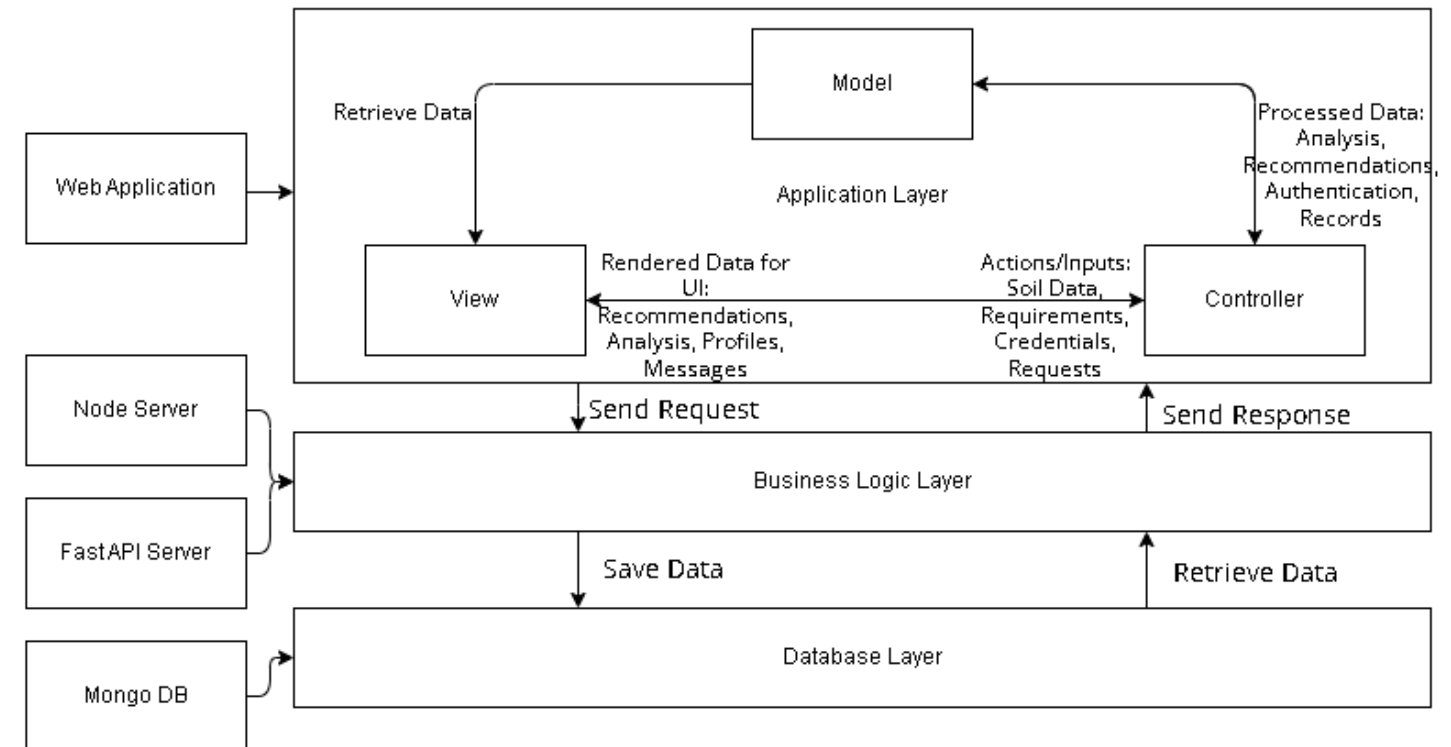


Figure 2: Architecture Diagram

4. Design Models

4.1 Activity Diagrams

General Functionalities for all users

4.1.1 Sign up

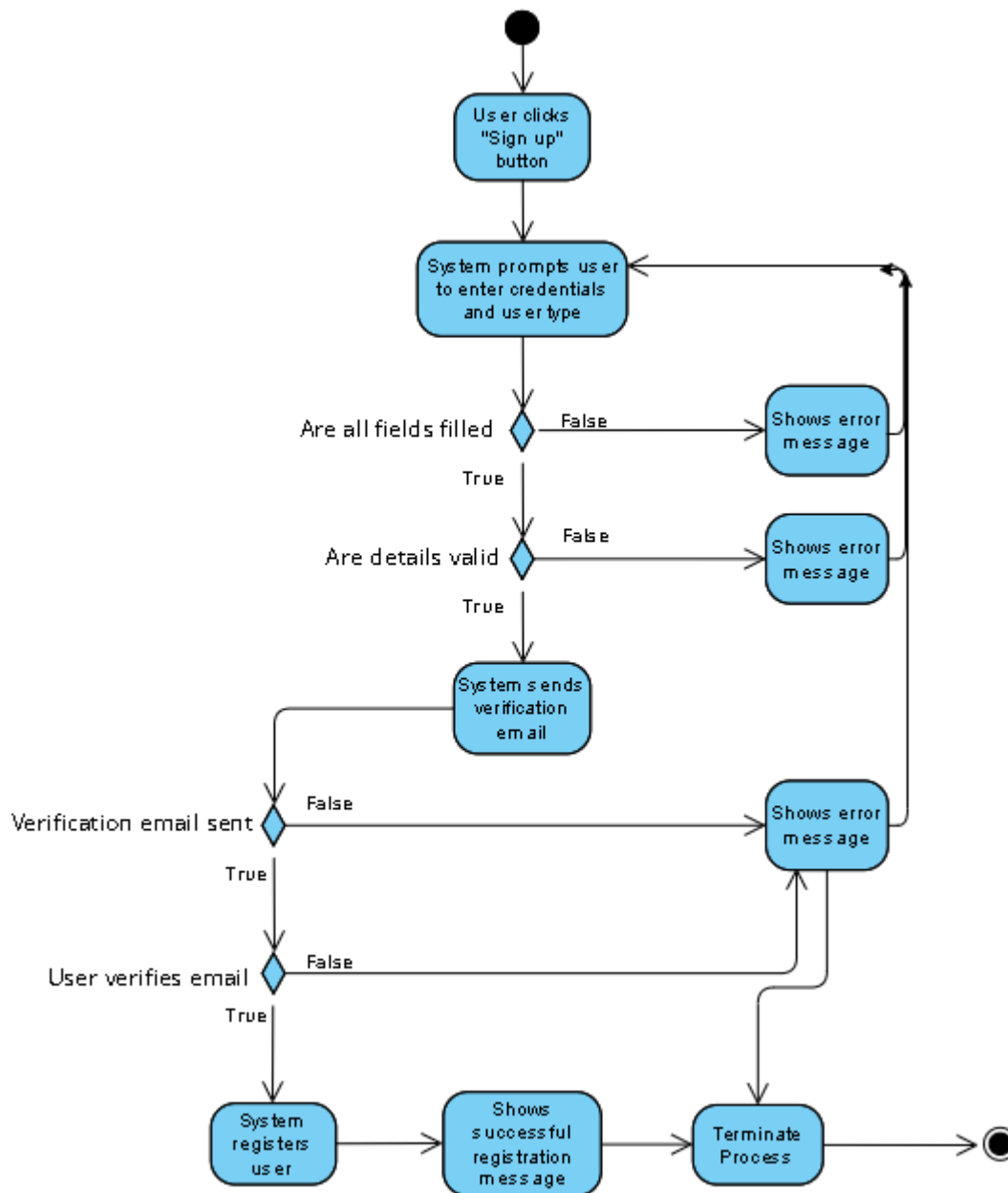


Figure 3: Sign up Activity Diagram

4.1.2 Sign in

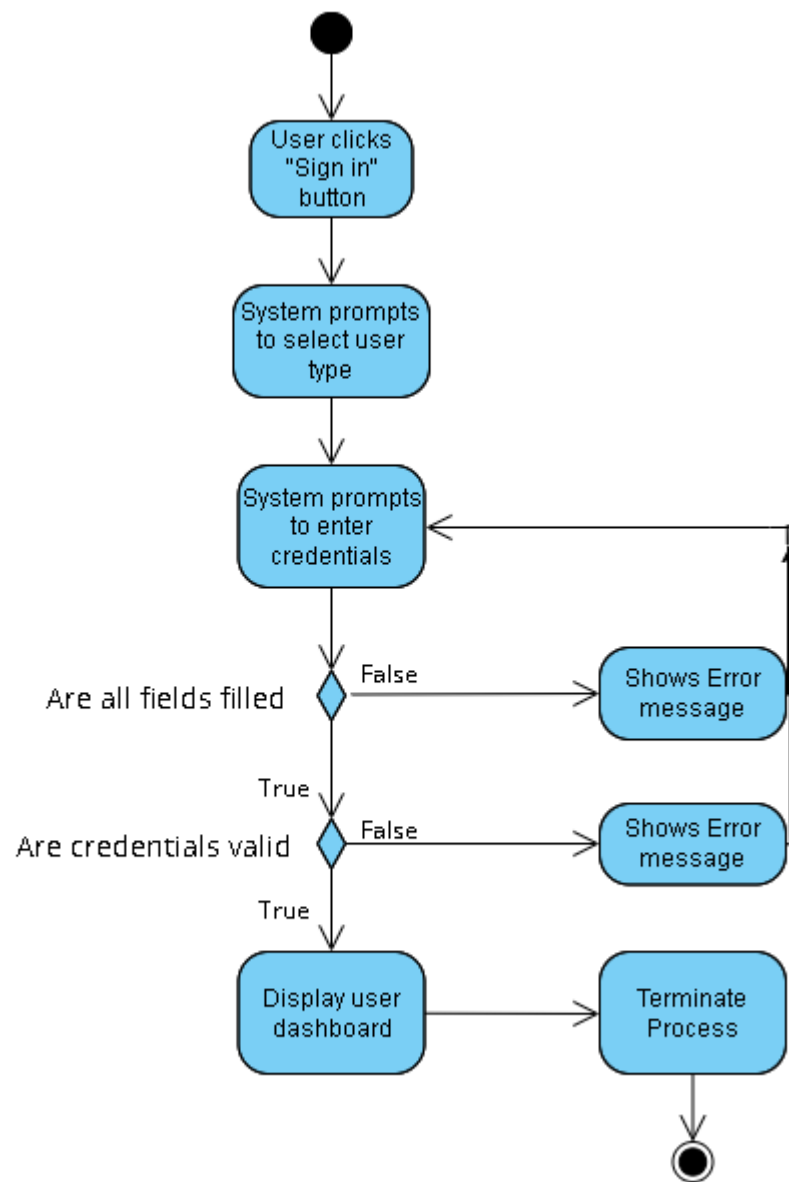


Figure 4: Sign in Activity Diagram

4.1.3 Manage Profile

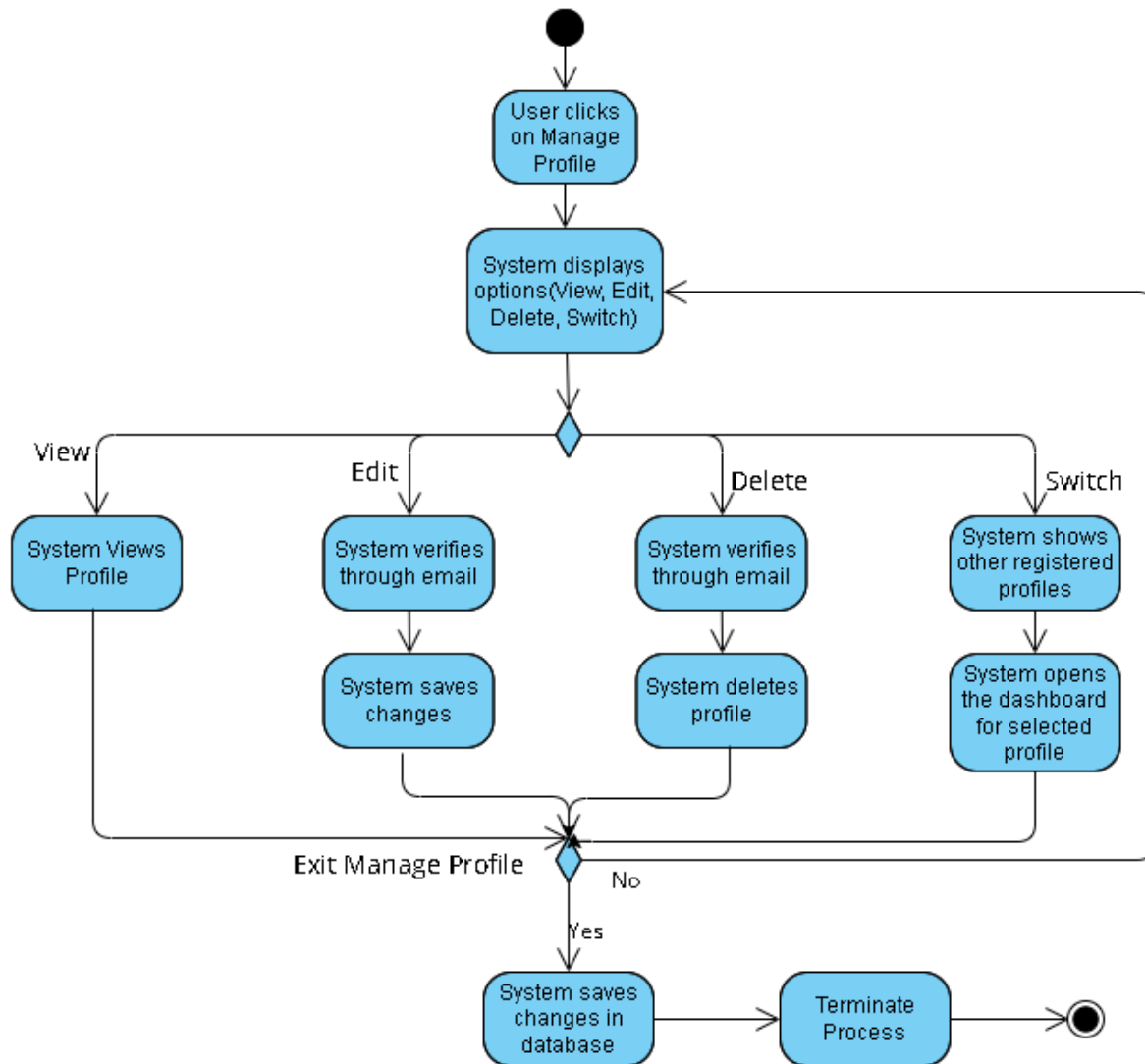


Figure 5: Manage Profile Activity Diagram

Admin Functionalities

4.1.4 Manage Registered Users

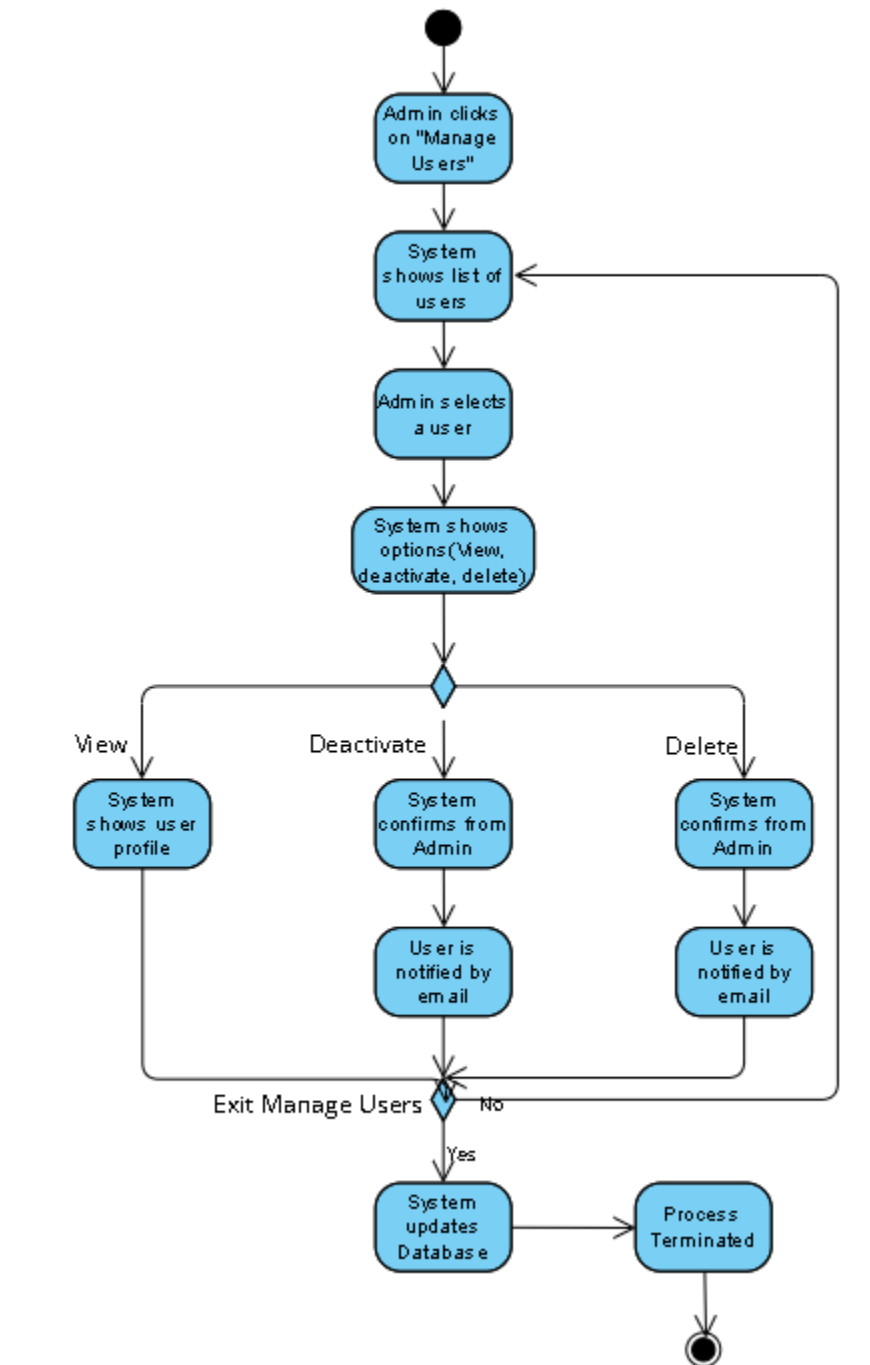


Figure 6: Manage Registered Users Activity Diagram

4.1.5 Manage Complaints

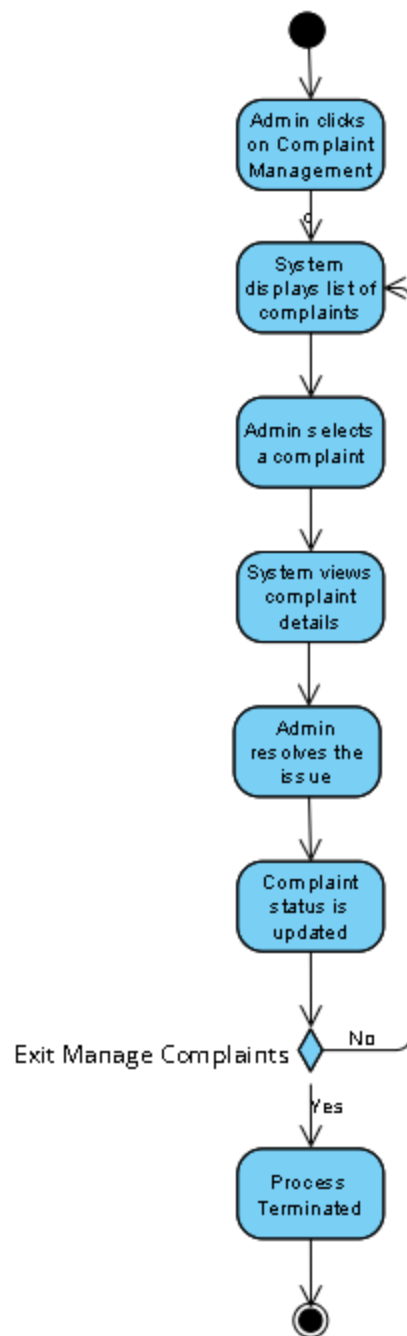


Figure 7: Manage Complaints Activity Diagram

4.1.6 Manage Reports

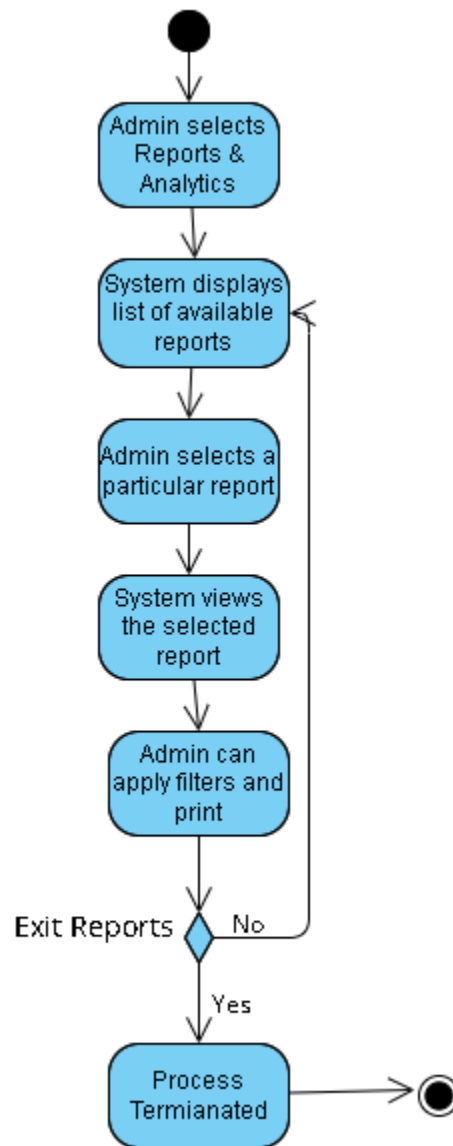


Figure 8: Manage Reports Activity Diagram

Farmer Functionalities

4.1.7 Analyze Soil

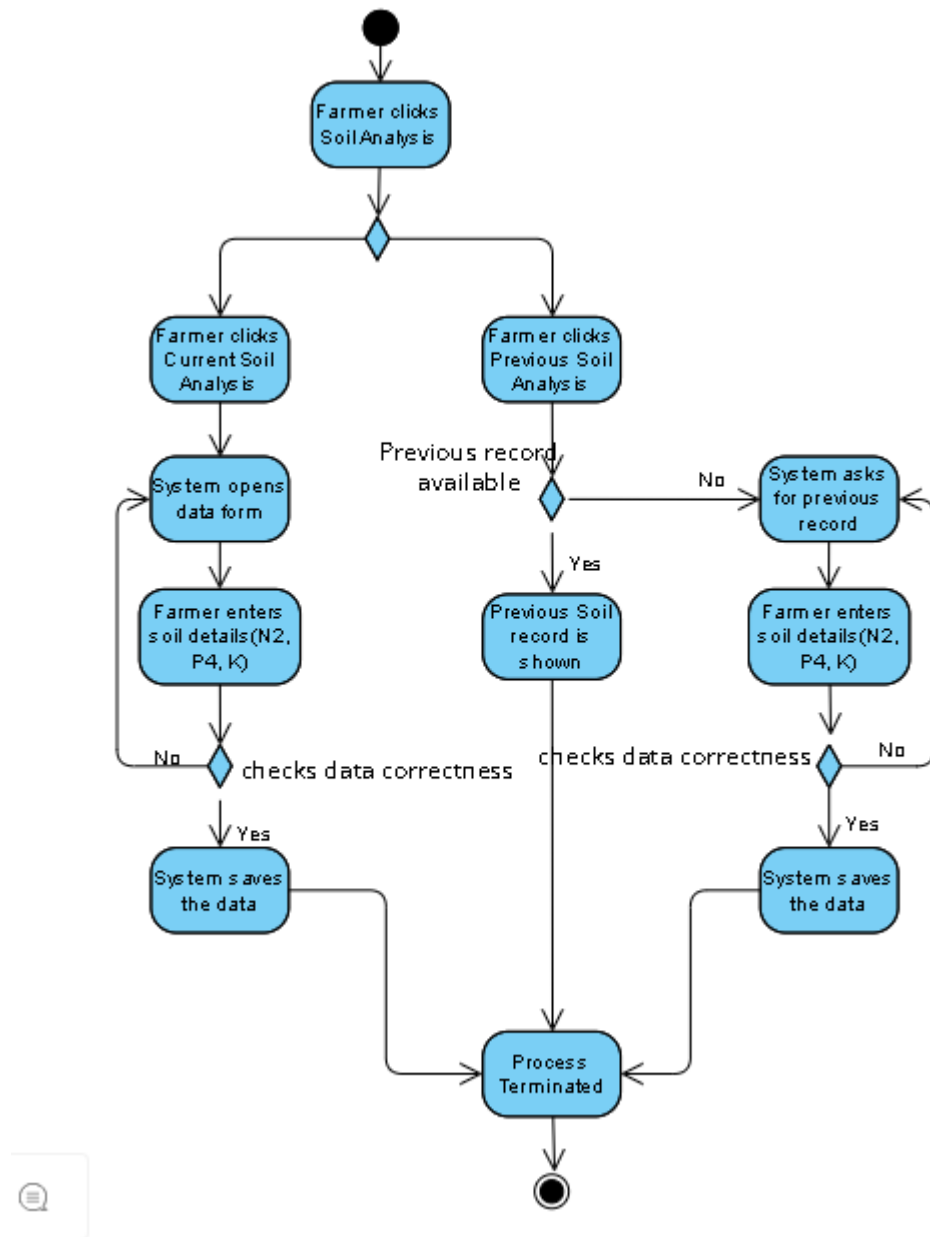


Figure 9: Soil Analysis Activity Diagram

4.1.8 Analyze Climate

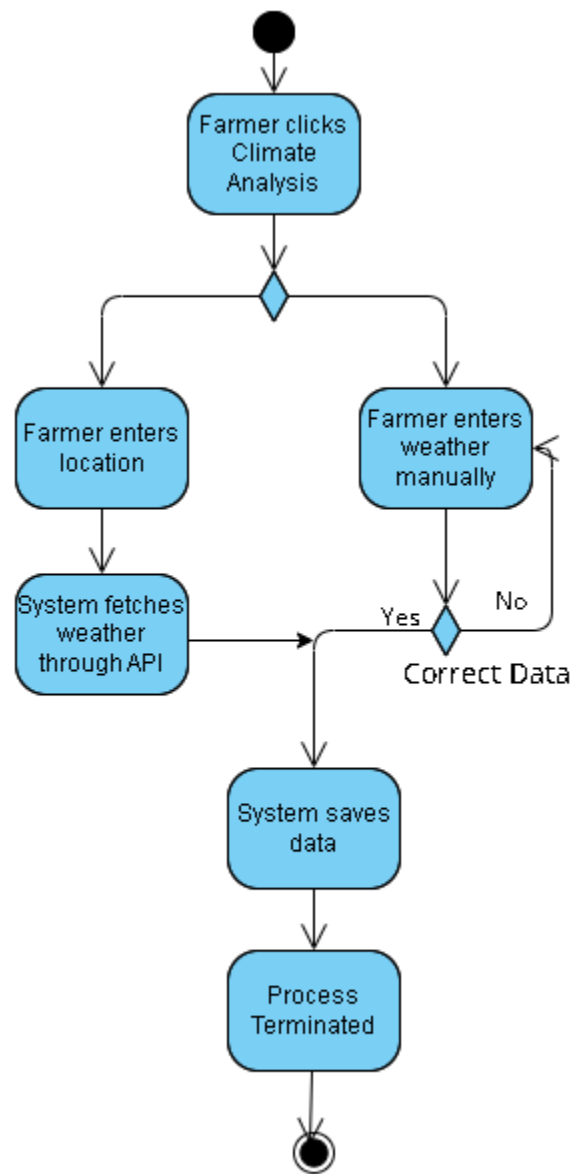


Figure 10: Climate Analysis Activity Diagram

4.1.9 Crop Recommendation

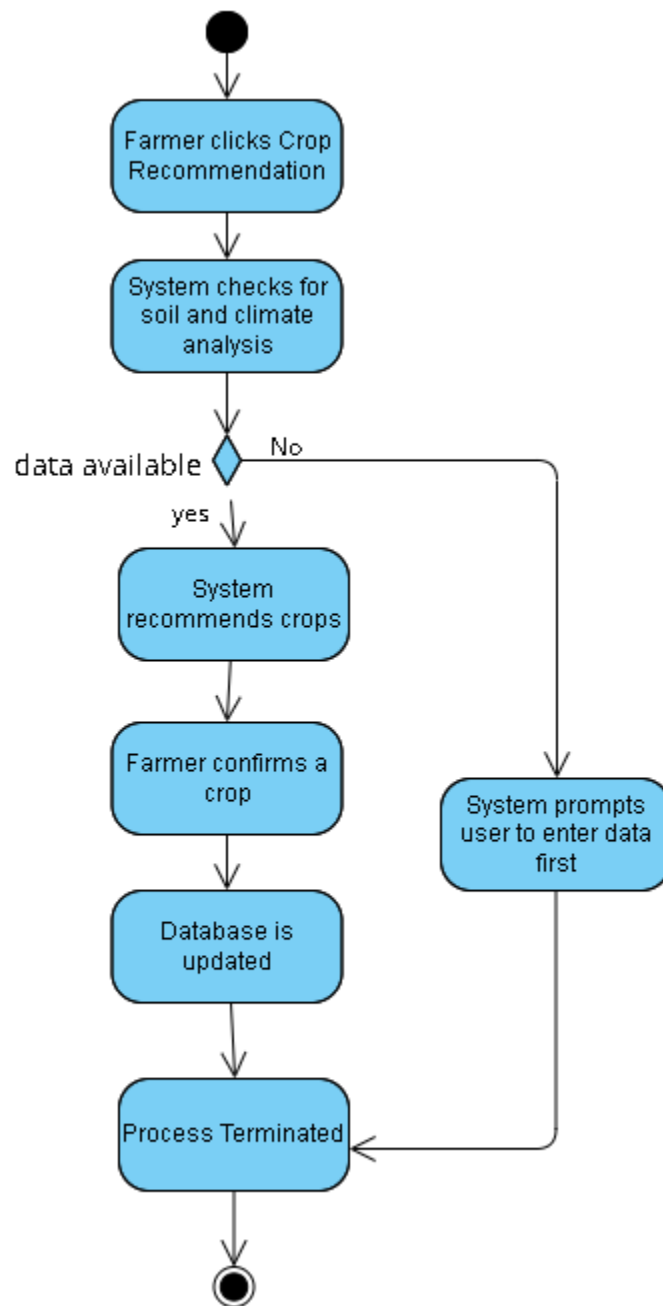


Figure 11: Crop Recommendation Activity Diagram

4.1.10 AI Chatbot

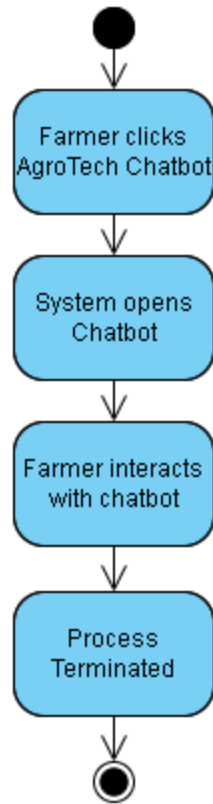


Figure 12: AI Chatbot Activity Diagram

4.1.11 Register Complaints

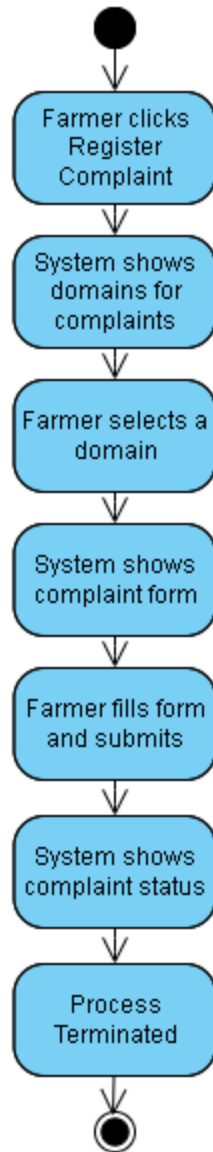


Figure 13: Register Complaints Activity Diagram

4.1.12 Identify Crop Stresses

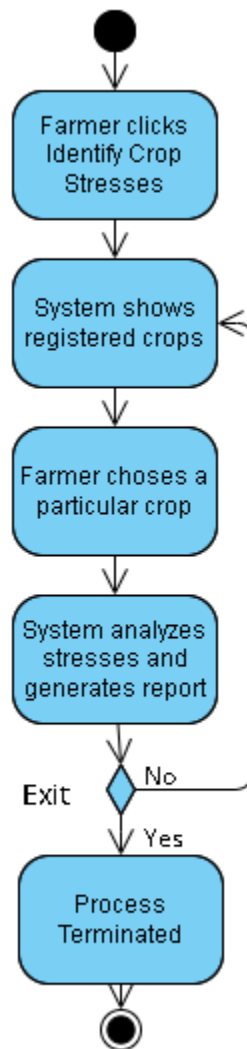


Figure 14: Identify Crop Stresses Activity Diagram

4.1.13 Identify Crop Diseases

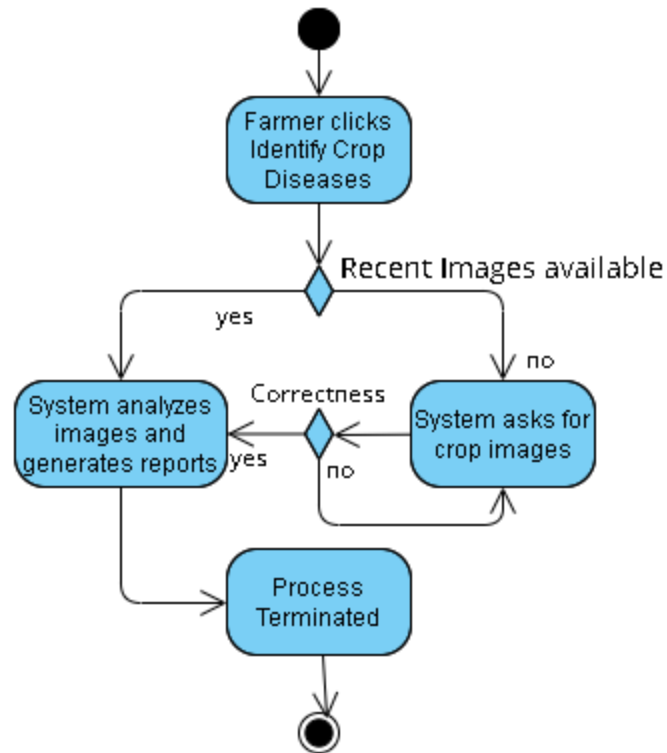


Figure 15: Identify Crop Diseases Activity Diagram

4.1.14 Estimate Crop Yield

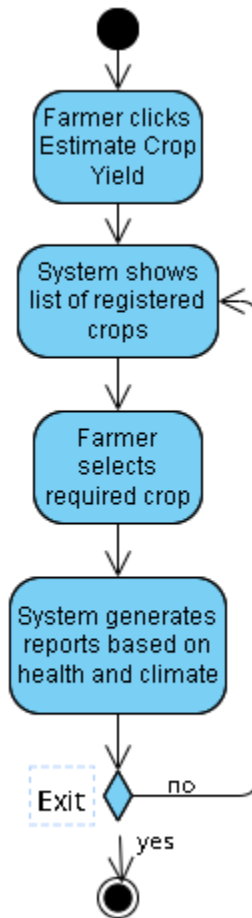


Figure 16: Estimate Crop Yield Activity Diagram

4.1.15 Crop Health Monitoring

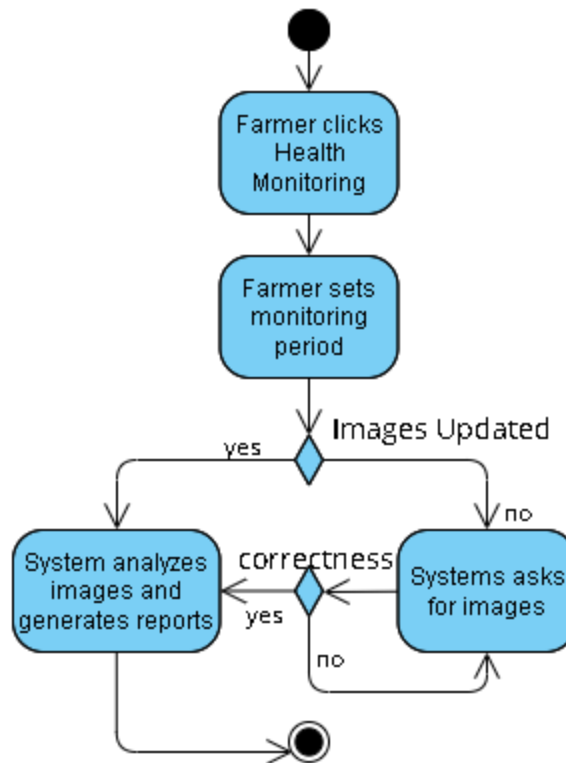


Figure 17: Crop Health Monitoring Activity Diagram

4.1.16 Crop Maturity Assessment

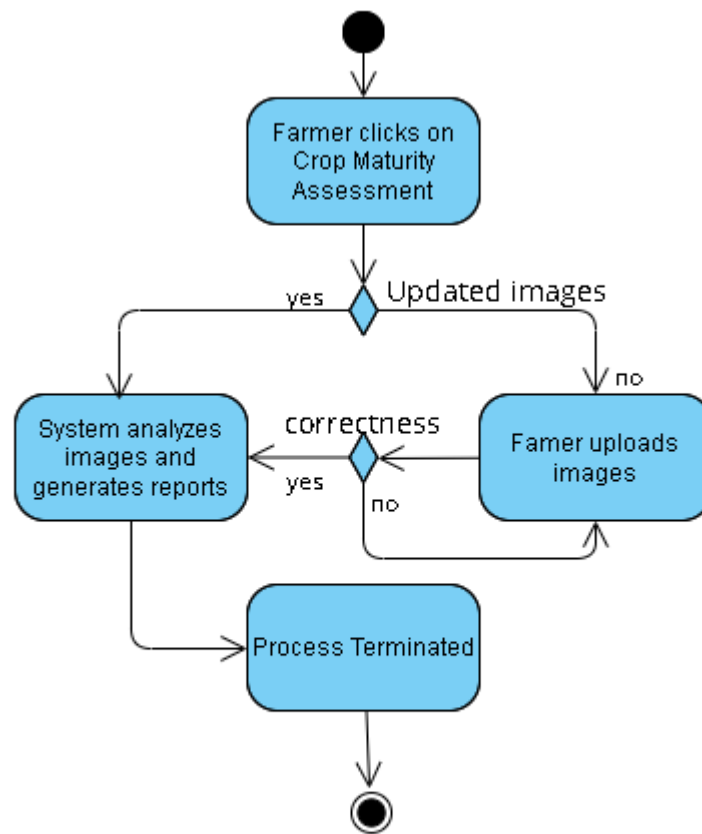


Figure 18: Crop Maturity Assessment Activity Diaram

4.1.17 Crop Quality Assessment

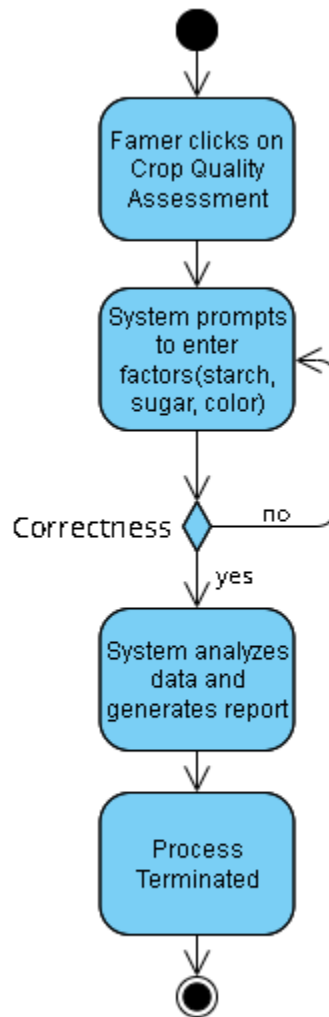


Figure 19: Crop Quality Assessment Activity Diagram

4.1.18 Calculate Optimal Harvest Time

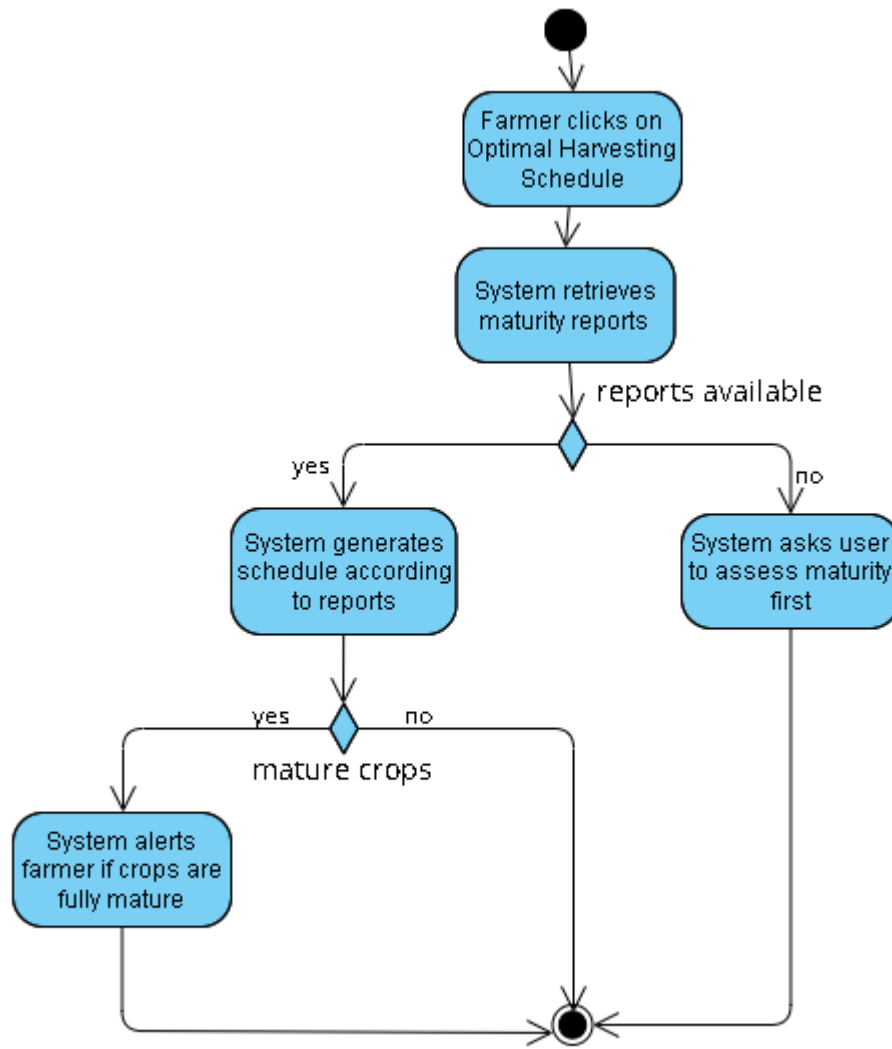


Figure 20: Harvest Scheduling Activity Diagram

4.1.19 Recommend Harvesting Tools

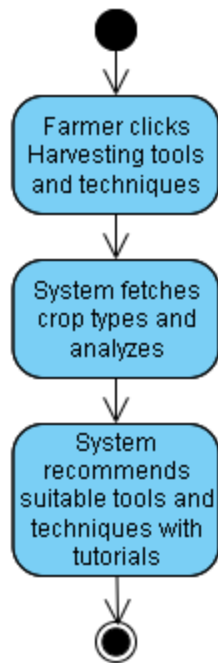


Figure 21: Recommend Harvesting Tools Activity Diagram

4.1.20 Recommend Transport

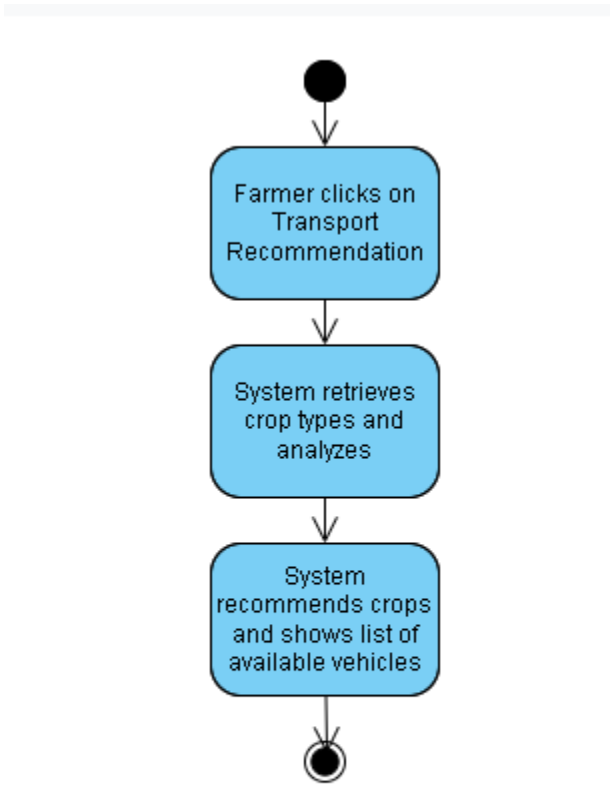


Figure 22: Transport Recommendation Activity Diagram

4.1.21 Packaging Assist

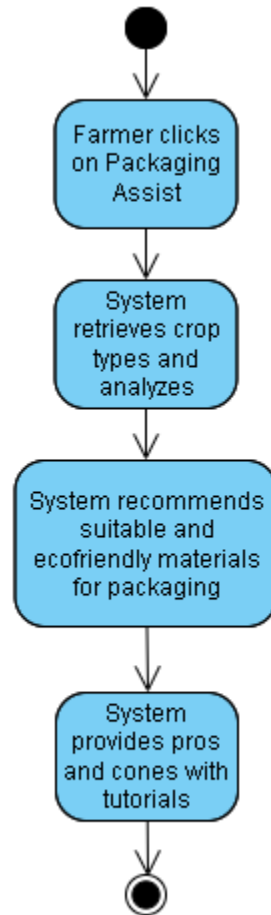


Figure 23: Packaging Assist Activity Diagram

4.1.22 Climate Controlled Storage Reservation

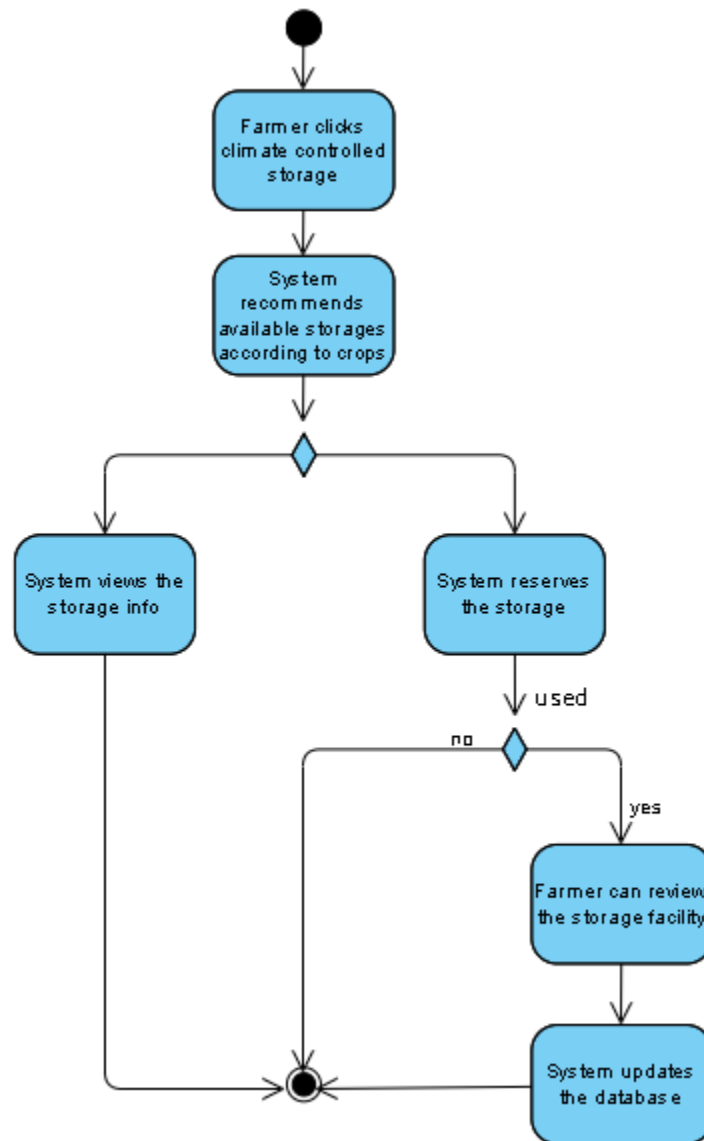


Figure 24: Storage Reservation Activity Diagram

4.1.23 Crop Quality Grading

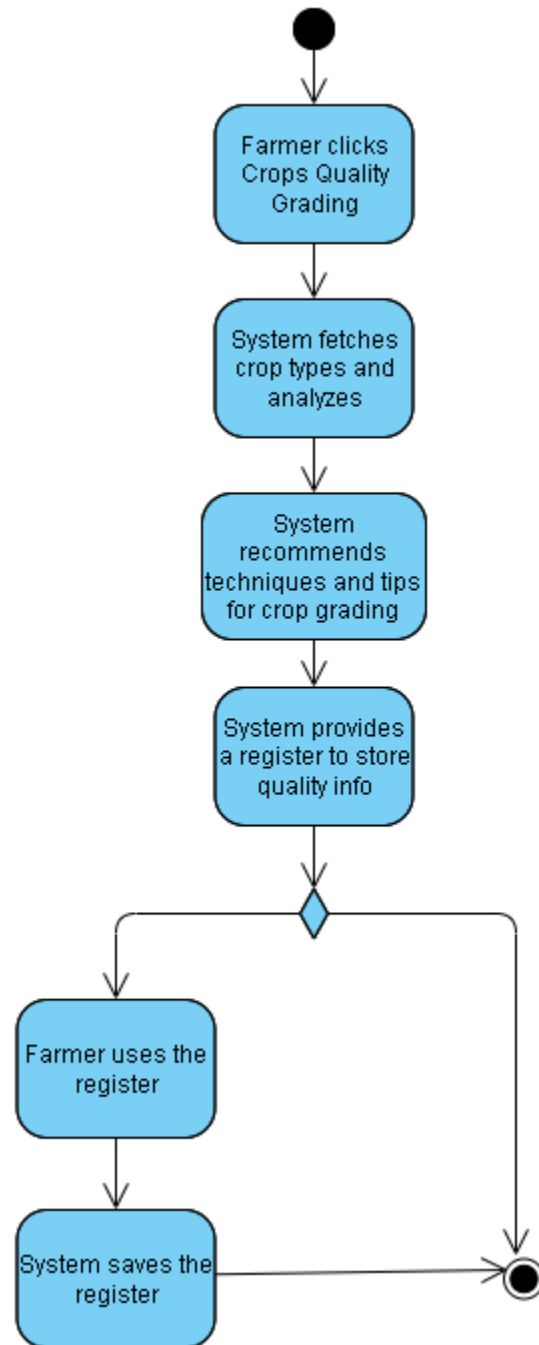


Figure 25: Crop Quality Grading Activity Diagram

Customer Functionalities

4.1.24 Bidding

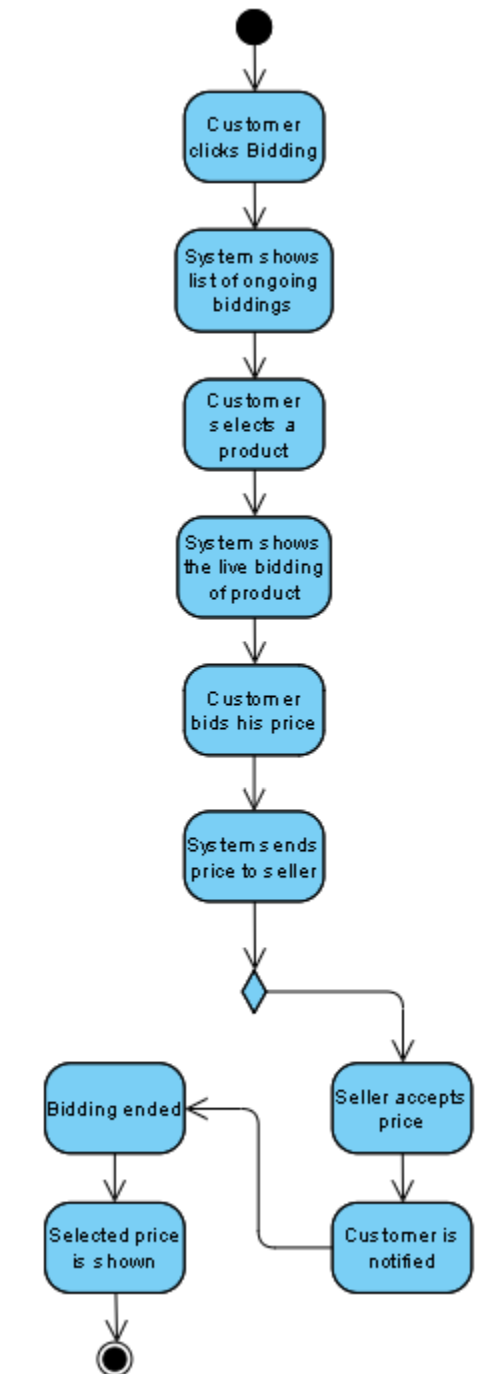


Figure 26: Bidding (Customer) Activity Diagram

4.1.25 Access Products

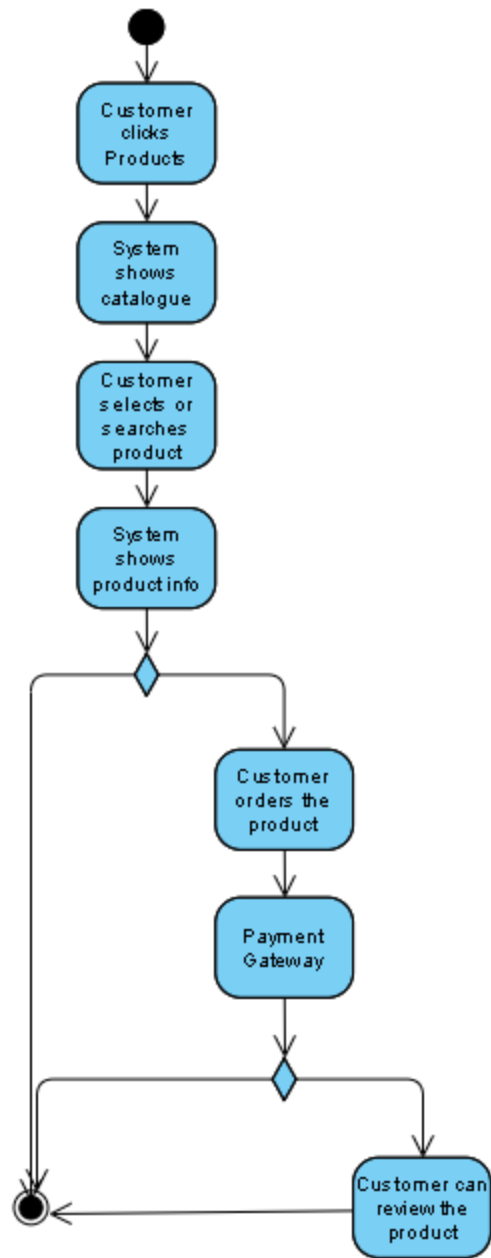


Figure 27: Access Products Activity Diagram

Seller Functionalities

4.1.26 Upload Product

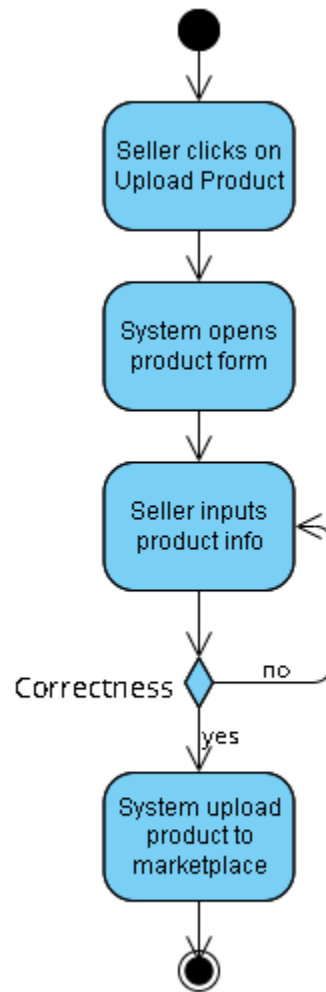


Figure 28: Upload Product Activity Diagram

4.1.27 Bidding

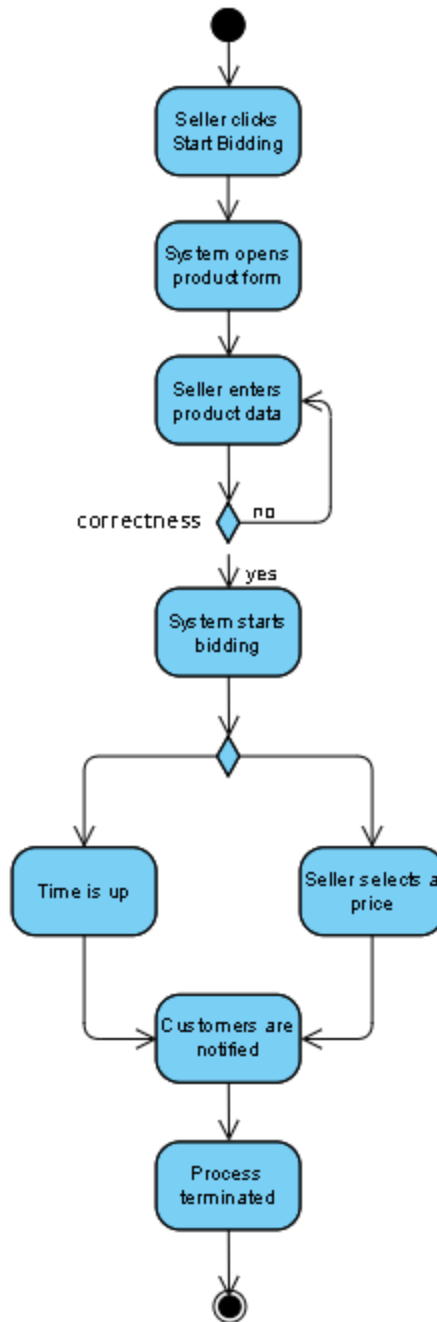


Figure 29: Bidding (Seller) Activity Diagram

4.1.28 Reports and Analytics



Figure 30: Reports and Analytics Activity Diagram

4.1.29 Feature Products



Figure 31: Feature Products Activity Diagram

4.2 Data Flow Diagrams

4.2.1 Level 0

Represents the high-level interaction between AgroTech and external entities such as users (farmers, customers, and sellers) and external systems (payment gateways and transportation services). It highlights the flow of data across the system boundaries.

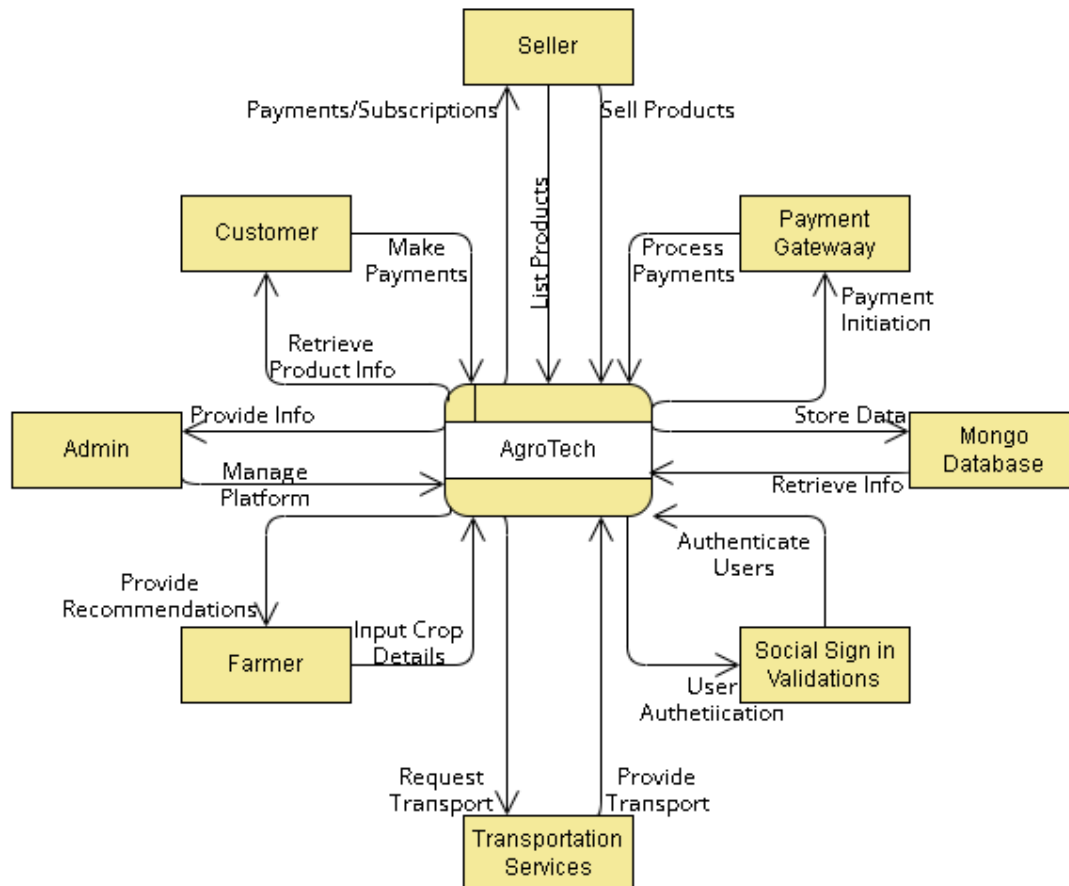


Figure 32: Level 0 Data Flow Diagram

4.2.2 Level 1

Depicts the decomposition of the AgroTech system into its major modules, including User Management, Crop Management, and Payment Services. It shows the detailed data flow between the system's internal components and external entities.

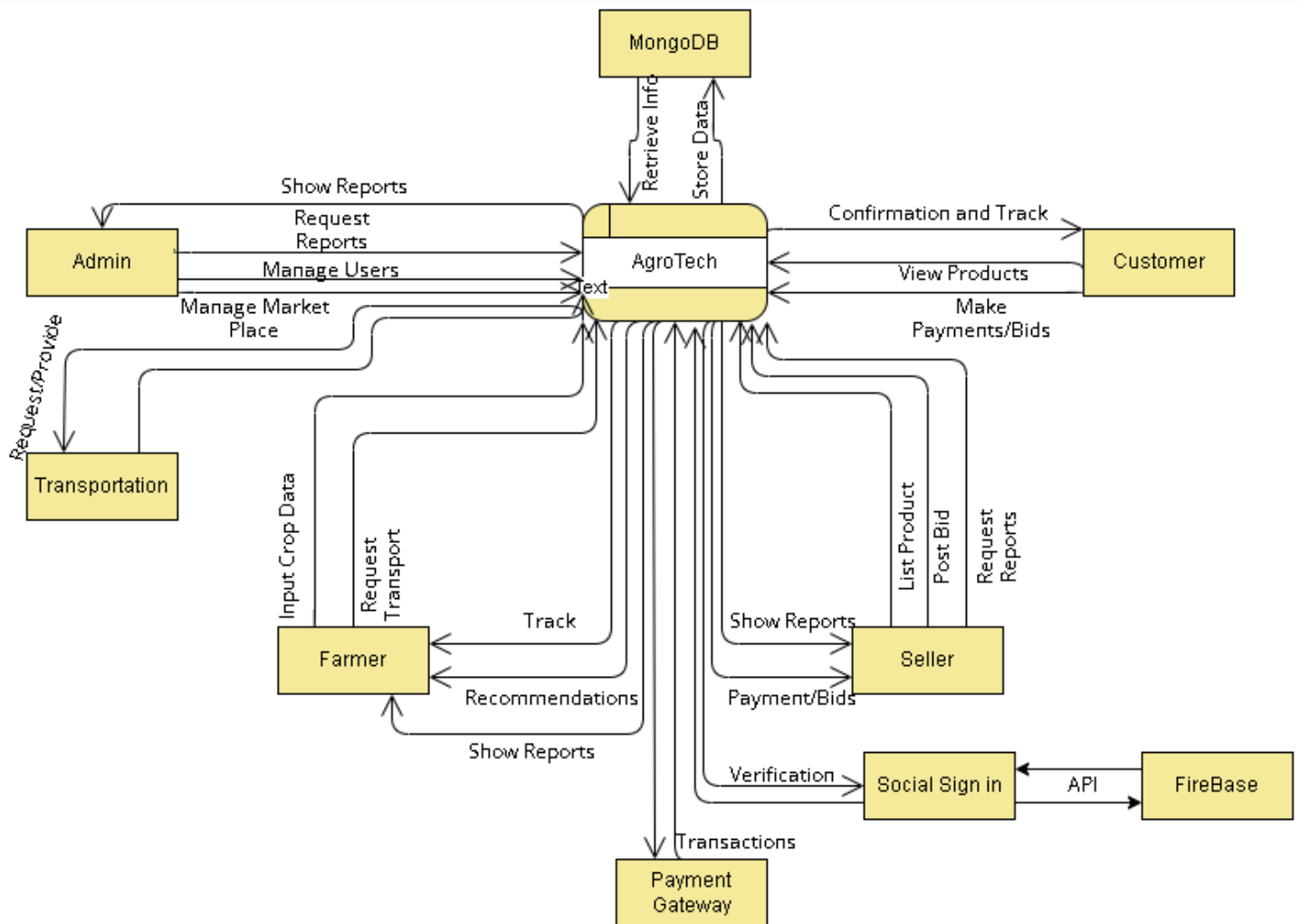


Figure 33: Level 1 Data Flow Diagram

4.2.3 Level 2

4.2.3.1 Module 1: User Profiling

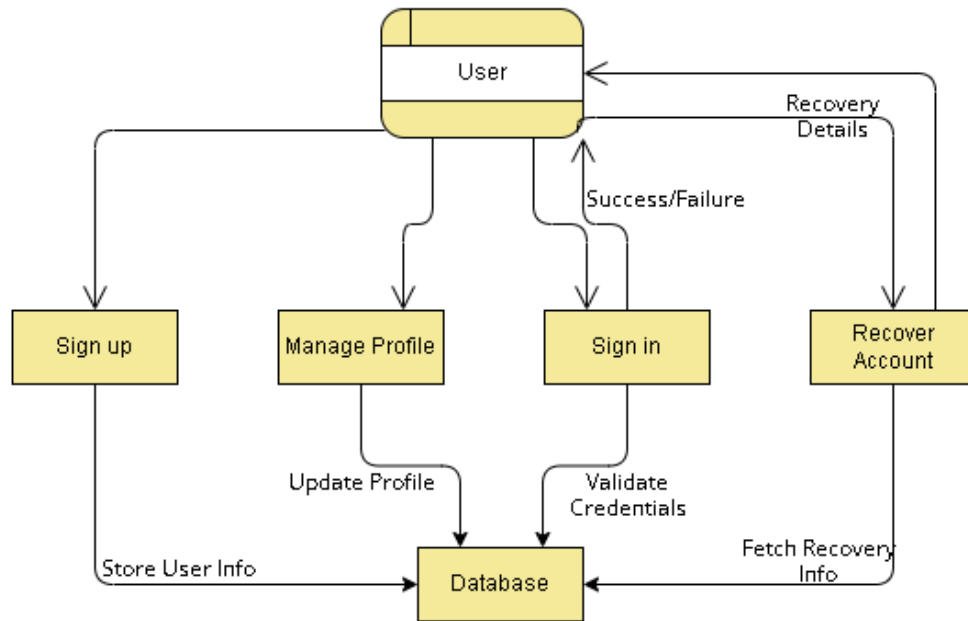


Figure 34: Module 1 Data Flow Diagram

4.2.3.2 Module 2: Soil Analysis

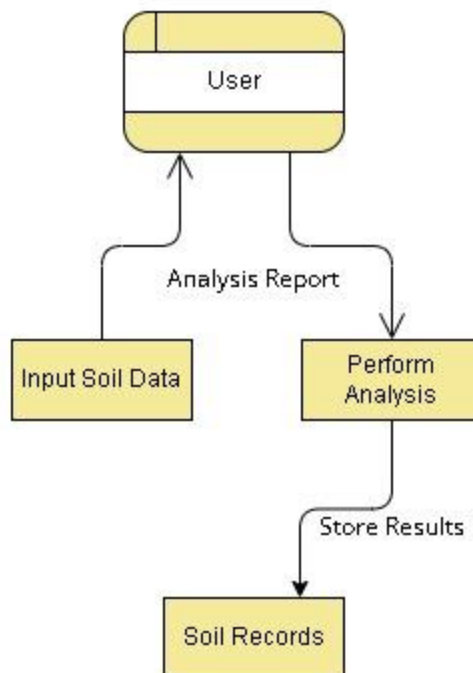


Figure 35: Module 2 Data Flow Diagram

4.2.3.3 Module 3: Climate Analysis

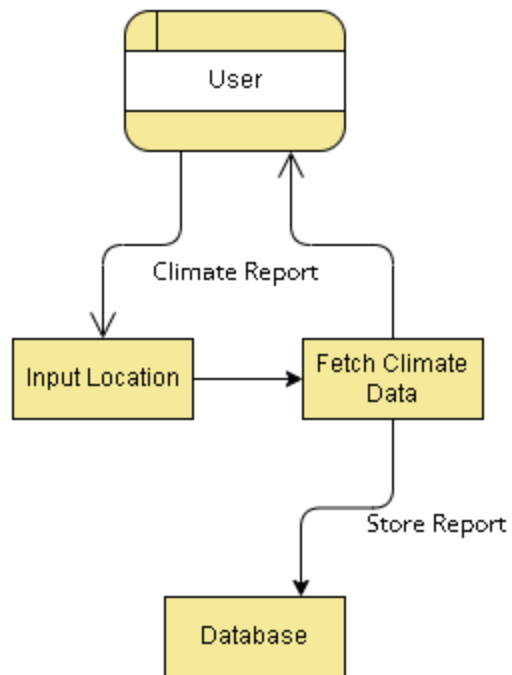


Figure 36: Module 3 Data Flow Diagram

4.2.3.4 Module 4: Crop Recommendation

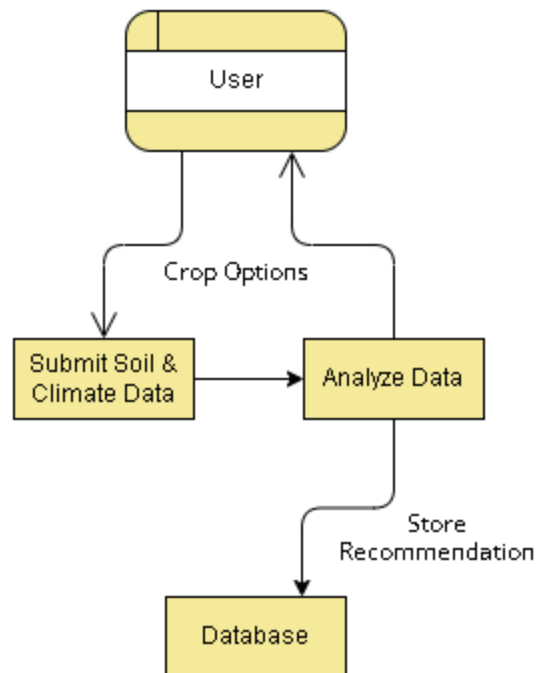


Figure 37: Module 4 Data Flow Diagram

4.2.3.5 Module 5: AI Chatbot

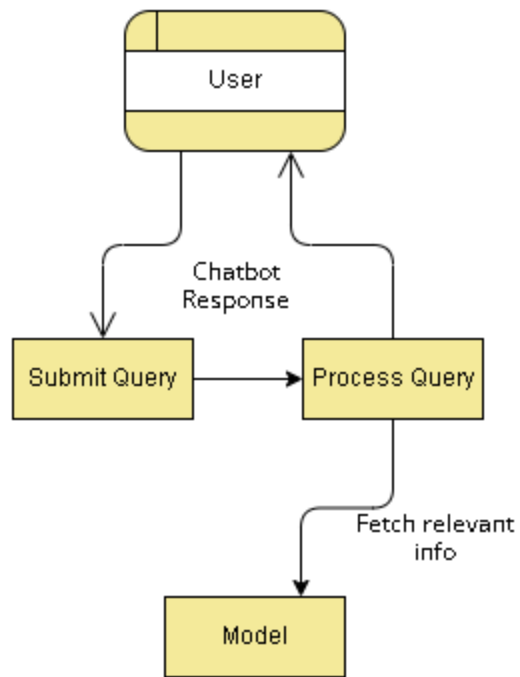


Figure 38: Module 5 Data Flow Diagram

4.2.3.6 Module 6: Crop Health Monitoring

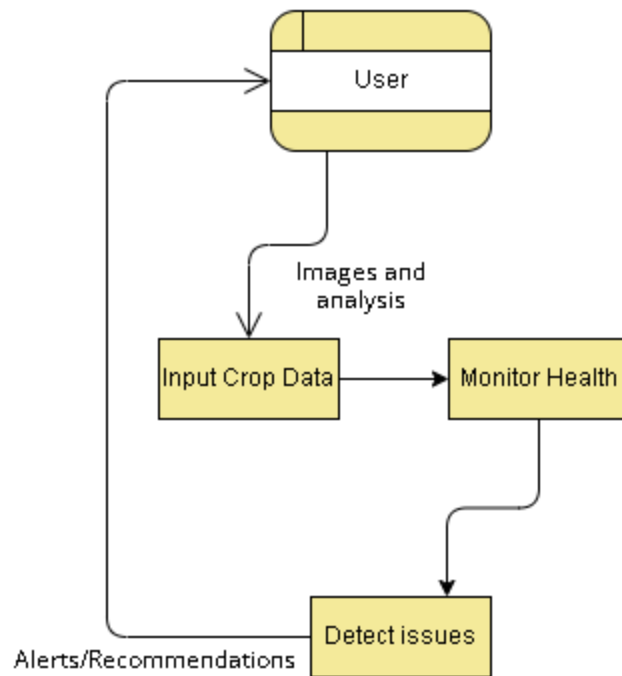


Figure 39: Module 6 Data Flow Diagram

4.2.3.7 Module 7: Yield Estimation and Prediction

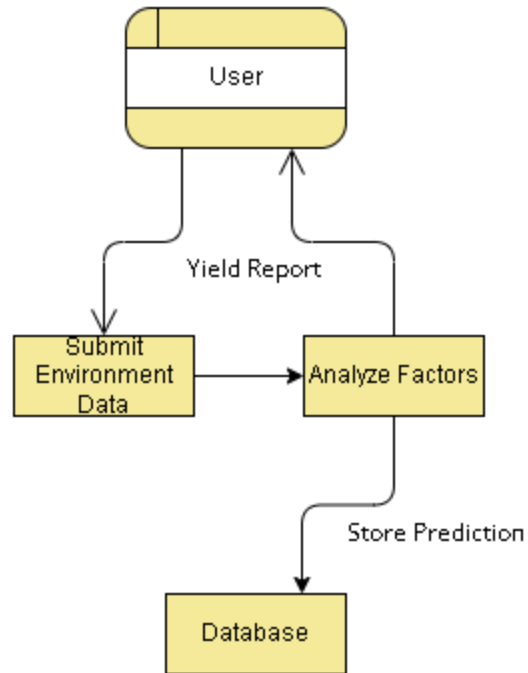


Figure 40: Module 7 Data Flow Diagram

4.2.3.8 Module 8: Crop Maturity Assessment

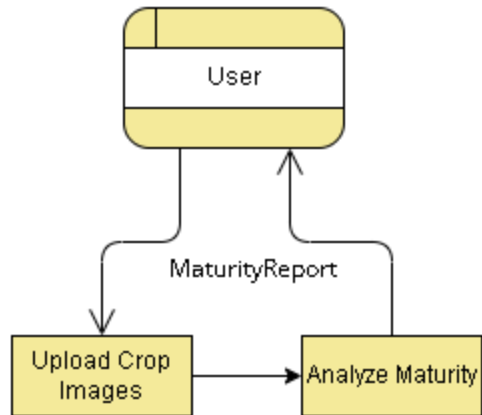
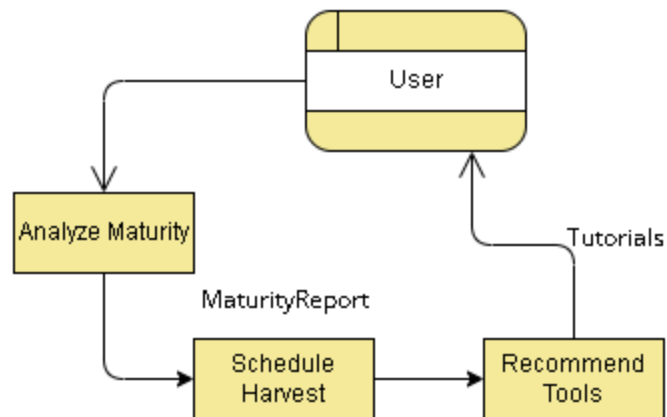
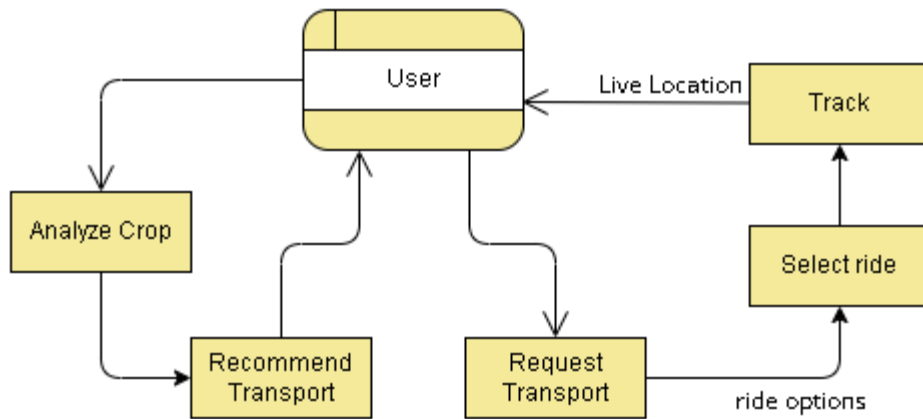
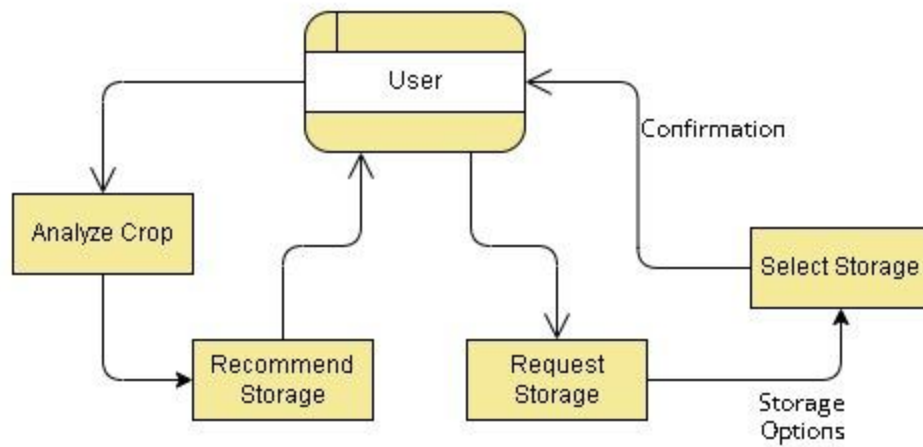
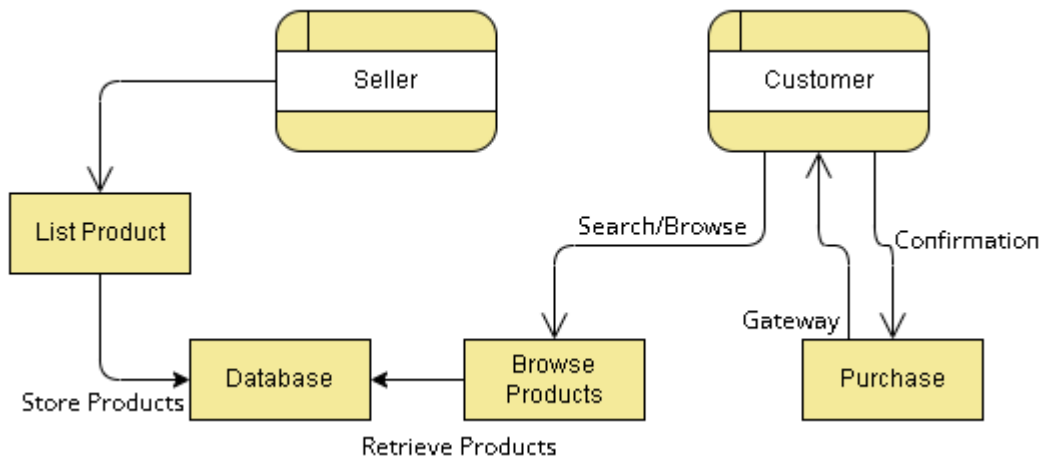


Figure 41: Module 8 Data Flow Diagram

4.2.3.9 Module 9: Harvesting Assist**Figure 42: Module 9 Data Flow Diagram****4.2.3.10 Module 10: Transport Assist****Figure 43: Module 10 Data Flow Diagram**

4.2.3.11 Module 11: Storage Assist**Figure 44: Module 11 Data Flow Diagram****4.2.3.12 Module 12: Online Marketplace****Figure 45: Module 12 Data Flow Diagram**

4.2.3.13 Module 13: Reports and Analytics

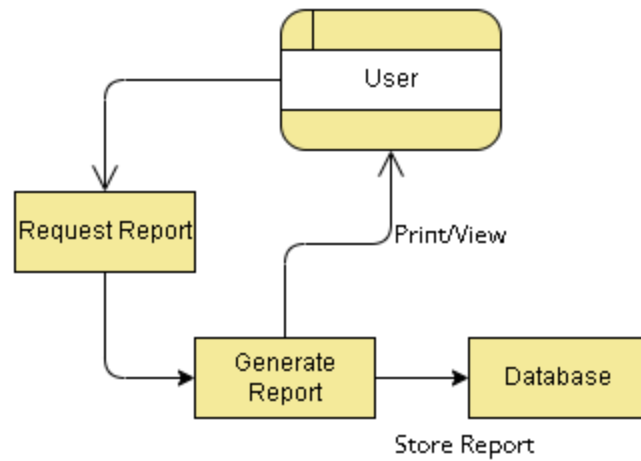


Figure 46: Module 13 Data Flow Diagram

4.3 State Transition Diagrams

4.3.1 Sign up

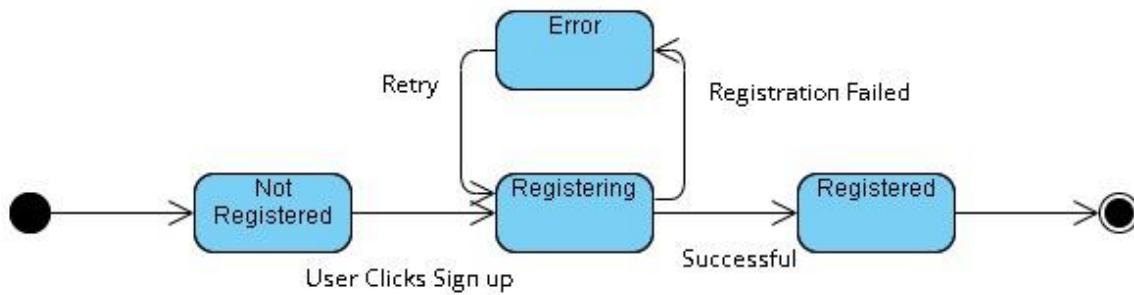


Figure 47: Sign up State Transition Diagram

4.3.2 Sign in

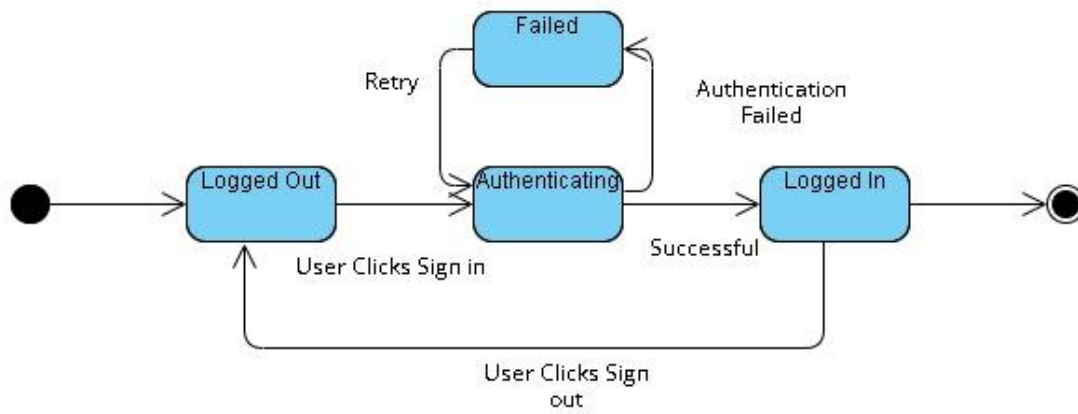


Figure 48: Sign in State Transition Diagram

4.3.3 Manage Profile

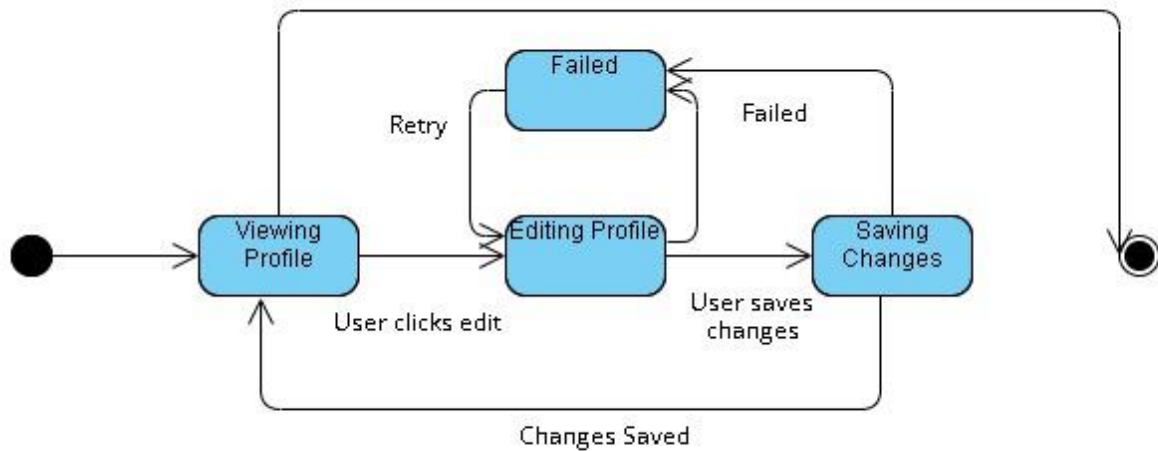


Figure 49: Manage Profile State Transition Diagram

4.3.4 Switch Profile

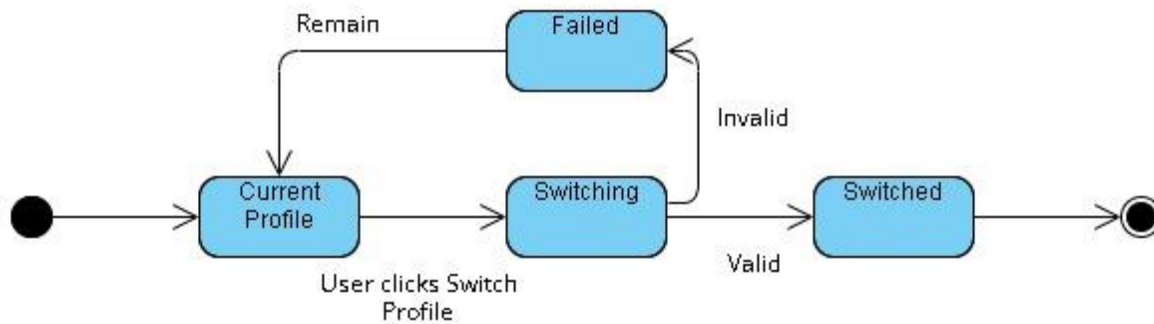


Figure 50: Switch Profile State Transition Diagram

4.3.5 Crop Maturity Assessment

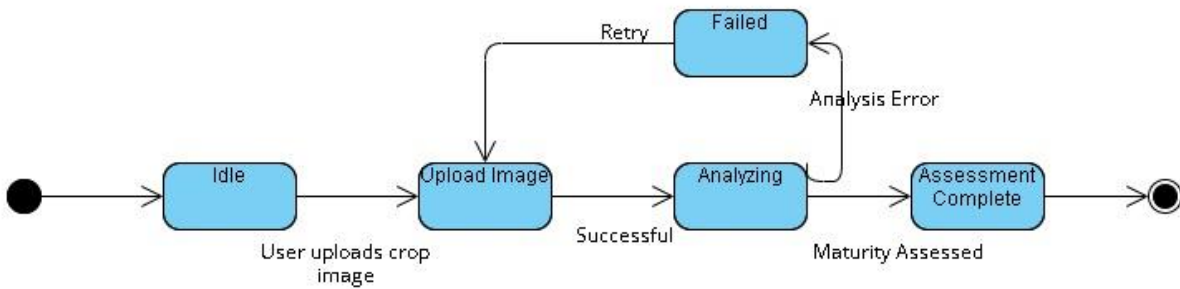


Figure 51: Maturity Assessment State Transition Diagram

4.3.6 Crop Health Monitoring

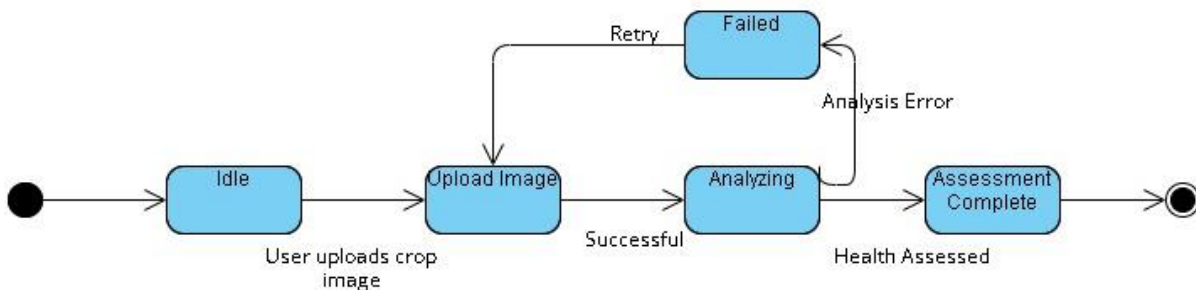


Figure 52: Health Monitoring State Transition Diagram

4.3.7 Soil Analysis

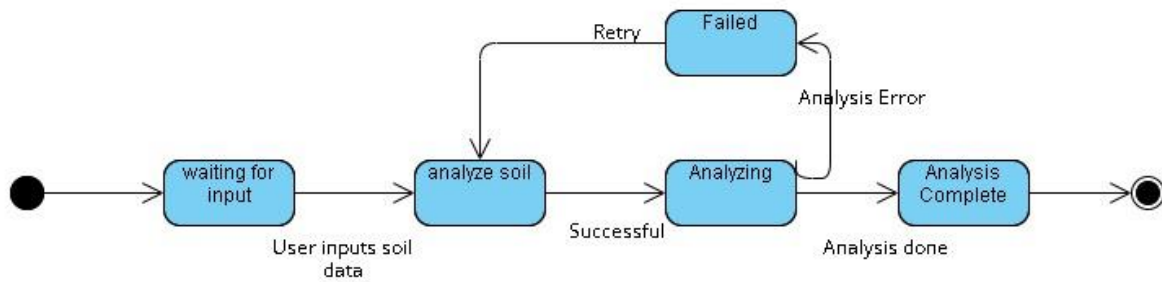


Figure 53: Soil Analysis State Transition Diagram

4.3.8 Yield Estimation and Prediction

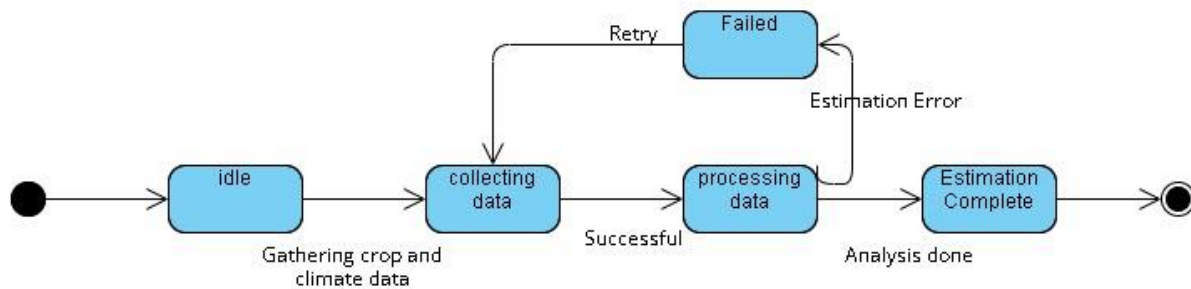


Figure 54: Yield Estimation and Prediction State Transition Diagram

4.3.9 Online Marketplace

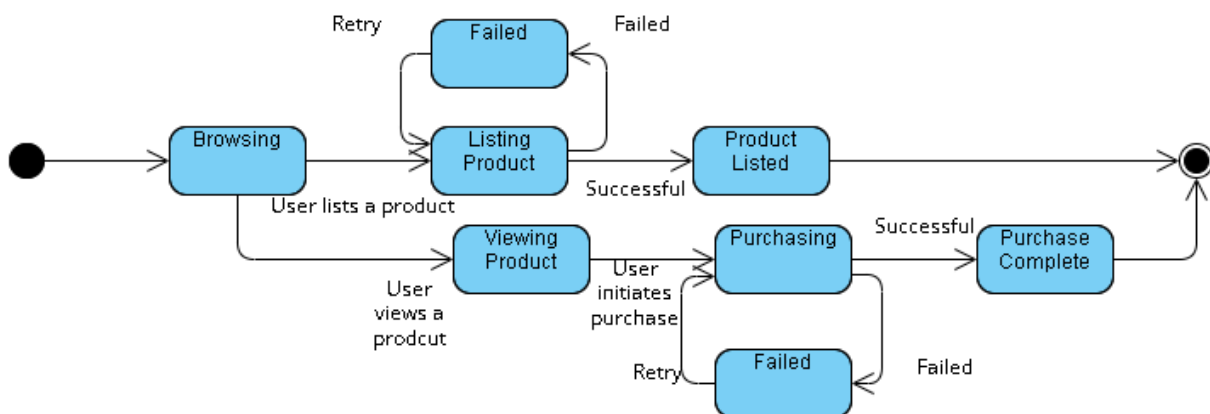


Figure 55: Online Marketplace State Transition Diagram

4.3.10 Bidding

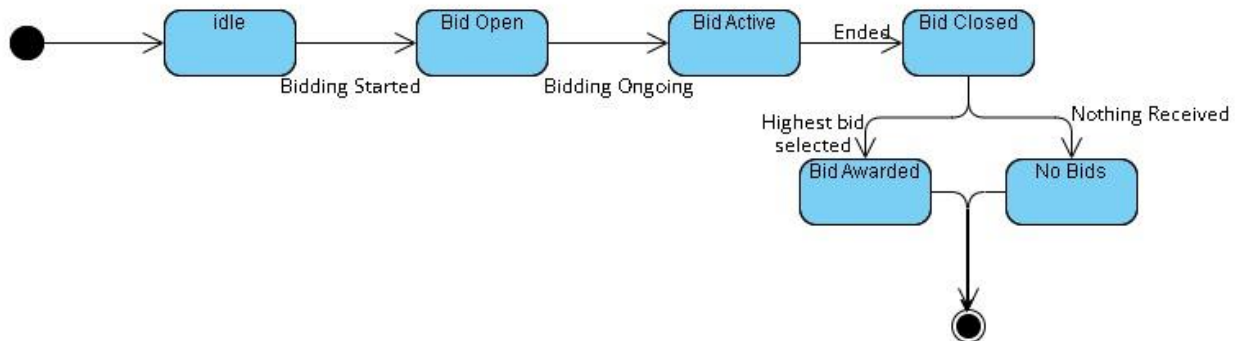


Figure 56: Bidding State Transition Diagram

4.3.11 Transportation Assist

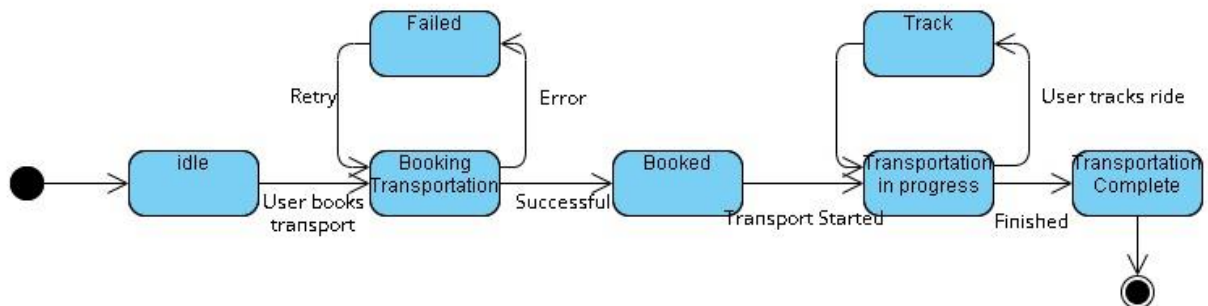


Figure 57: Transportation Assist State Transition Diagram

4.3.12 Manage User Complaints

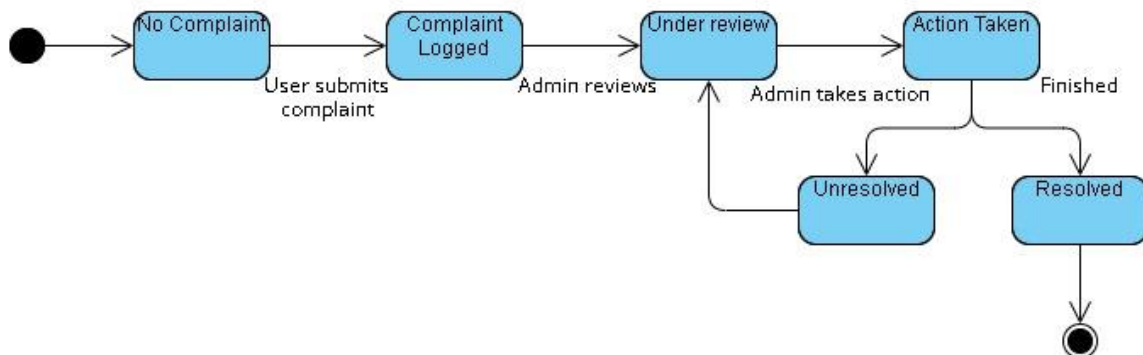


Figure 58: Manage User Complaints State Transition Diagram

4.3.13 Feedback and Reviews

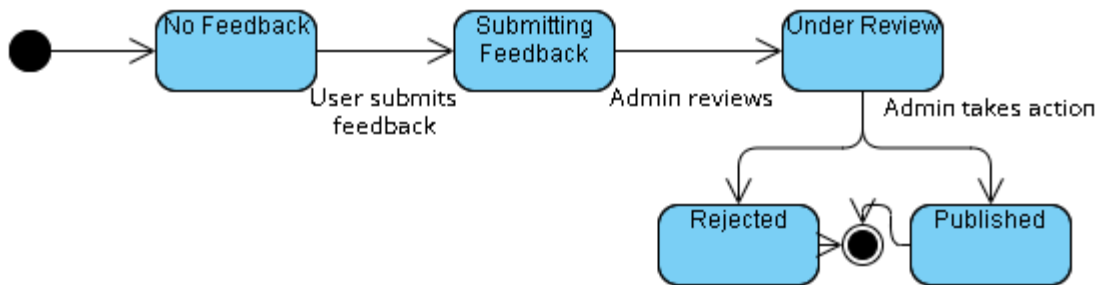


Figure 59: Feedback and Reviews State Transition Diagram

4.3.14 Storage Assist

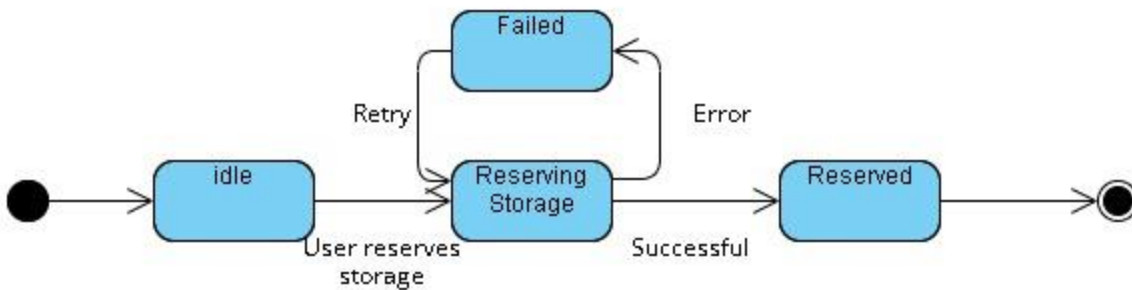


Figure 60: Storage Assist State Transition Diagram

4.3.15 Reports and Analytics

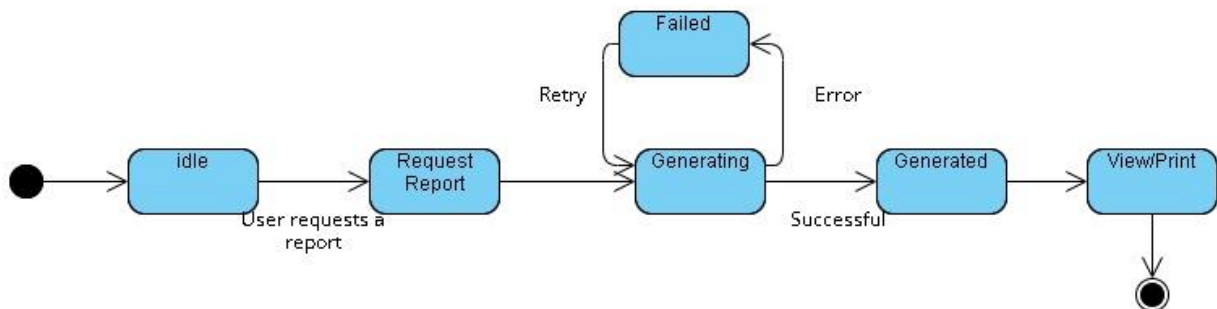


Figure 61: Reports and Analytics State Transition Diagram

5. Data Design

5.1 Data Dictionary

Collection	Property	Data Type	Description
User	_id	Schema.Types.ObjectId	The auto generated primary key for User
User	username	String	The unique username for the user.
User	firstName	String	The first name of the user.
User	lastName	String	The last name of the user.
User	email	String	The unique email address for the user.
User	password	String	The user's password.
User	userType	String	The type of user (Admin, Farmer, Customer, Seller).
User	googleId	String	The Google account ID, used for Google login
User	profilePicture	String	URL of the user's profile picture.
User	phoneNumber	String	The phone number of the user.
User	isActive	Boolean	A flag indicating if the user's account is active.
User	isLoggedIn	Boolean	A flag indicating if the user is currently logged in.
User	verificationToken	String	A token for email verification.
User	verificationTokenExpire	Date	Expiration date for the verification token.
Report	reportType	String	The type of report (e.g., userActivity, salesData, systemMetrics)
Report	generatedBy	ObjectId	Reference to the User who generated the report.
Report	reportData	Object	Contains the actual data of the report
Report	generatedAt	Date	The date and time the report was generated (default: current date/time).

Report	Filters	Object	Filters applied to generate the report.
CropRecommendations	userId	ObjectId	Reference to the User who requested the crop recommendation.
CropRecommendations	Nitrogen	Number	The Nitrogen content in the soil.
CropRecommendations	Phosphorus	Number	The Phosphorus content in the soil.
CropRecommendations	Potassium	Number	The Potassium content in the soil.
CropRecommendations	temperature	Number	The current temperature in the region.
CropRecommendations	humidity	Number	The current humidity in the region.
CropRecommendations	pH	Number	The pH level of the soil (required).
CropRecommendations	rainfall	Number	The annual rainfall in the region.
CropRecommendations	recommendedCrop	String	The recommended crop based on the provided data.
CropRecommendations	CreatedAt	Date	The date and time when the recommendation was generated.
Complaint	userId	ObjectId	Reference to the User who raised the complaint.
Complaint	complaintText	String	The content of the complaint.
Complaint	status	String	The current status of the complaint (e.g., pending, resolved, closed). Default: pending.

Complaint	createdAt	Date	The date and time when the complaint was created.
Complaint	resolvedBy	ObjectId	Reference to the User who resolved the complaint.
Complaint	resolutionText	String	The text explaining the resolution of the complaint.
ChatBot	userId	ObjectId	Reference to the User who initiated the conversation with Chatbot.
ChatBot	userMessage	String	The content of the query of the user.
ChatBot	botResponse	String	The response of the chatbot against user's query
ChatBot	timeStamp	String	Exact time when the message is logged
ChatBot	messageType	String	Indicates whether the message is from the user or the bot.
ChatBot	isResolved	Boolean	Flag to indicate whether the query has been resolved or not.
ChatBot	context	Object	Store additional context if needed (e.g., user's previous questions, bot intent)

Table 1: AgroTech Data Dictionary

5.2 Data Schemas:

5.2.1 User.js

```
import mongoose from "mongoose";
```

```
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true, select: false },
  userType: { type: String, enum: ['Admin', 'Farmer', 'Customer', 'Seller'], required: true },
```

```
googleId: { type: String, unique: true, sparse: true },
profilePicture: { type: String },
phoneNumber: { type: String, required: false }, // New field for phone number
isActive: { type: Boolean, default: true },
isLoggedIn: { type: Boolean, default: false },
verificationToken: { type: String },
verificationTokenExpire: { type: Date },
createdAt: { type: Date, default: Date.now },
});
```

```
const User = mongoose.model('User', userSchema);
```

```
export default User;
```

5.2.2 CropRecommendation.js

```
import mongoose from "mongoose";
```

```
const CropRecommendationSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  nitrogen: { type: Number, required: true },
  phosphorus: { type: Number, required: true },
  potassium: { type: Number, required: true },
  temperature: { type: Number, required: true },
  humidity: { type: Number, required: true },
  pH: { type: Number, required: true },
  rainfall: { type: Number, required: true },
  recommendedCrop: { type: String, required: true },
  createdAt: { type: Date, default: Date.now },
});
```

```
const CropRecommendations = mongoose.model("CropRecommendations",
CropRecommendationSchema);
```

```
export default CropRecommendations;
```

5.2.3 Complaint.js

```
// models/Complaint.js
import mongoose from 'mongoose';
const complaintSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  complaintText: {
    type: String,
    required: true,
  },
});
```

```
    trim: true,
  },
  status: {
    type: String,
    enum: ['pending', 'resolved', 'closed'],
    default: 'pending',
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
  resolvedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: false,
  },
  resolutionText: {
    type: String,
    required: false,
  },
});
const Complaint = mongoose.model('Complaint', complaintSchema);
export default Complaint;
```

5.2.4 Report.js

```
// models/Report.js
import mongoose from 'mongoose';

const reportSchema = new mongoose.Schema({
  reportType: {
    type: String,
    enum: ['userActivity', 'salesData', 'systemMetrics'],
    required: true,
  },
  generatedBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  reportData: {
    type: Object,
    required: true,
  },
  generatedAt: {
    type: Date,
    default: Date.now,
  },
  filters: {
    type: Object,
    required: false,
  },
});
```

```
const Report = mongoose.model('Report', reportSchema);  
export default Report;
```

5.2.5 ChatBot.js

```
import mongoose from 'mongoose';  
  
const chatbotSchema = new mongoose.Schema({  
  userId: {  
    type: mongoose.Schema.Types.ObjectId,  
    ref: 'User',  
    required: true,  
  },  
  userMessage: {  
    type: String,  
    required: true,  
    trim: true,  
  },  
  botResponse: {  
    type: String,  
    required: true,  
    trim: true,  
  },  
  timestamp: {  
    type: Date,  
    default: Date.now,  
  },  
  conversationId: {  
    type: String,  
    required: true,  
  },  
  messageType: {  
    type: String,  
    enum: ['user', 'bot'],  
    required: true,  
  },  
  isResolved: {  
    type: Boolean,  
    default: false,  
  },  
  context: {  
    type: Object,  
    default: {},  
  },  
});  
  
const Chatbot = mongoose.model('Chatbot', chatbotSchema);  
export default Chatbot;
```

6. Human Interface Design

6.1 Screen Images



Figure 62: Home Page



Figure 63: Products

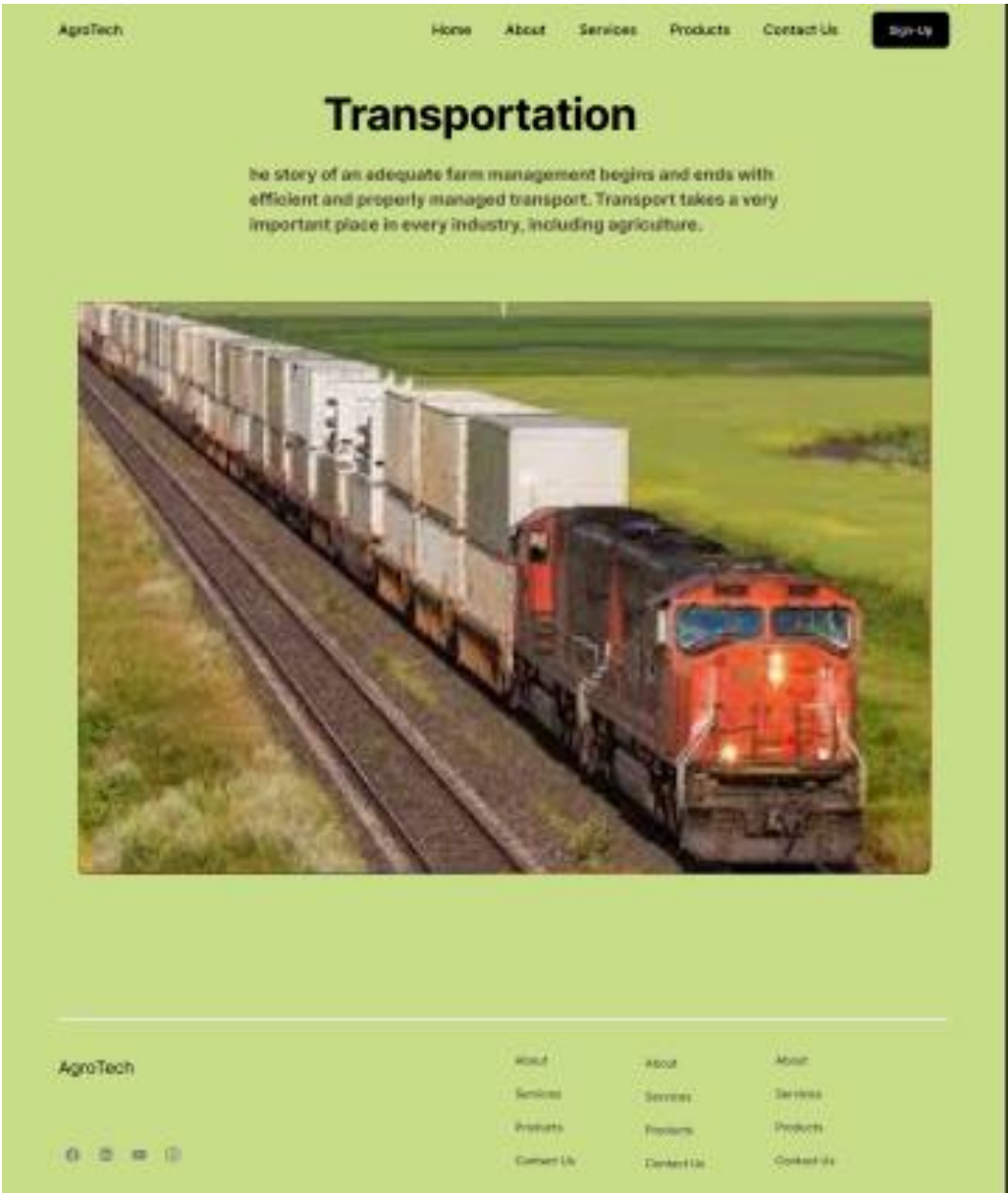


Figure 64: Transportation

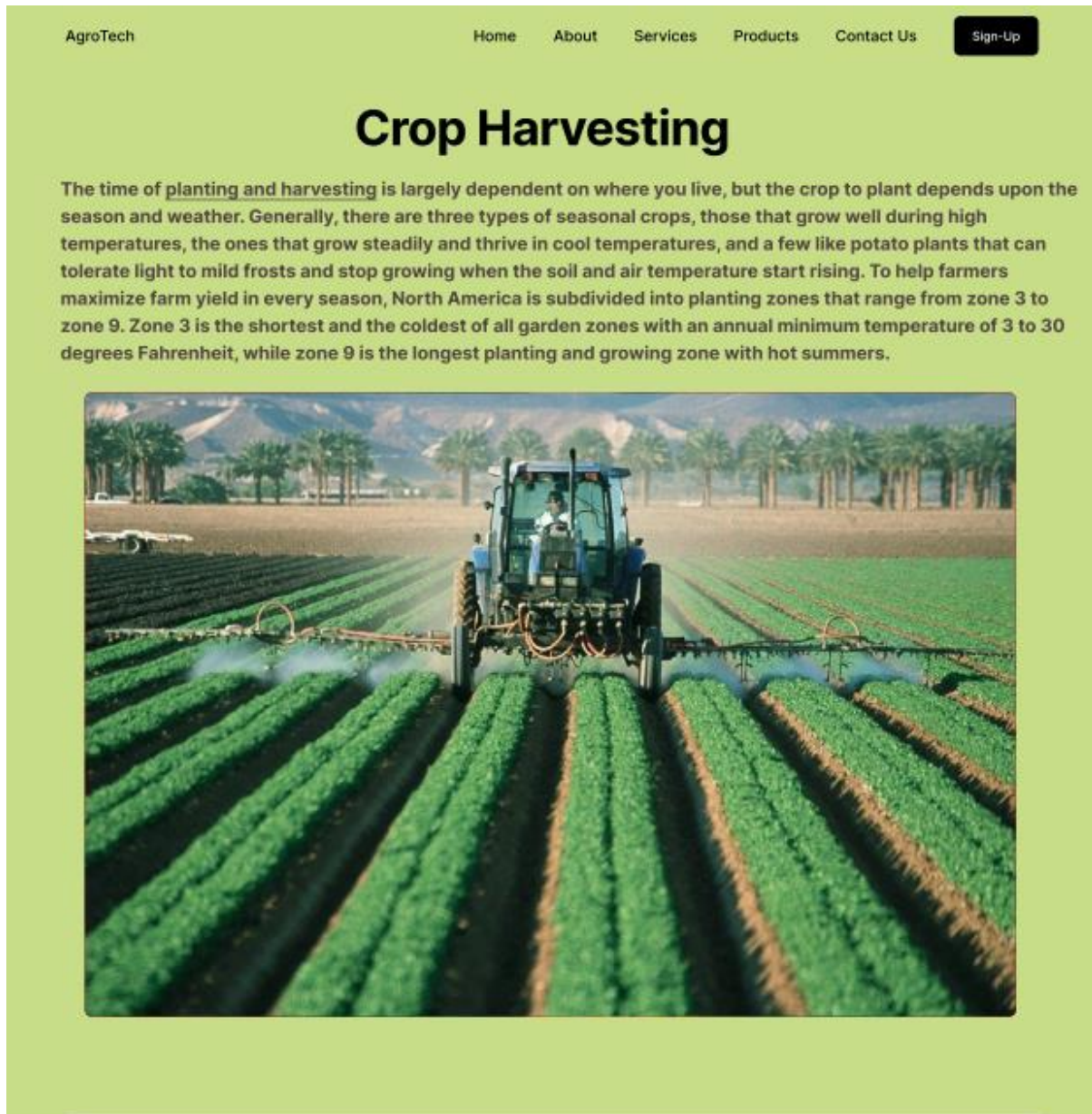


Figure 65: Harvesting Assist

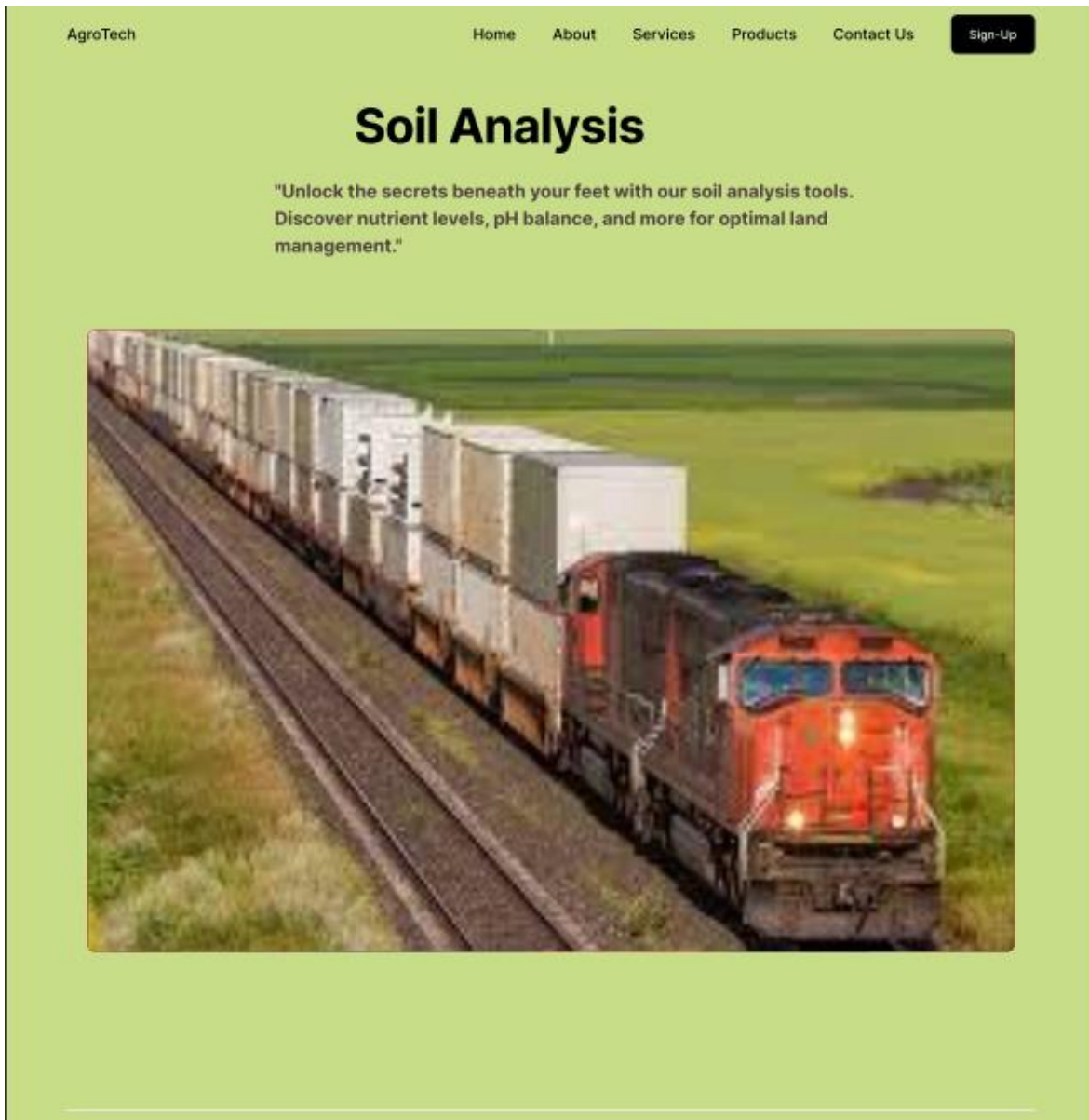


Figure 66: Soil Analysis

6.2 Screen Objects and Actions

Screen Object	Actions
Button	Buttons are used for various purposes in the application, those application may include clicking on sign in button etc.
Radio Button	To select option from multiple options given. E.g., selecting userType
Field	Fields to provide input
RouterLink	Link to navigate to other pages
NavBar	Navlinks provided to navigate to different sections of the application
Icons	Used for various purposes like viewing information, navigation to certain page etc.
Header	To provide user with navigation links
DashBoard Title	Displays the title (e.g., "Farmer Dashboard," "Admin Dashboard")

Table 2: Screen Objects and Actions

7. Implementation

7.1 Algorithm

7.1.1 Register User

Input: username, firstName, lastName, email, password, userType, phoneNumber, otp

Output: JSON response with success or error message

Try:

 If OTP is provided:

 user = FindUserByEmail(email)

 If user does not exist:

 Return 404: 'User not found'

 If verificationTokenExpire < current time:

 DeleteUserById(user.id)

 Return 400: 'OTP expired. Please register again.'

 hashedOtp = HashOTP(otp)

 If verificationToken does not match hashedOtp:

 Return 400: 'Invalid OTP. Please try again.'

 ActivateUser(user)

 SendWelcomeEmail(user.email, user.username)

 Return 200: 'Account activated successfully. Welcome email sent.'

Else:

 If UserExists(email):

 Return 400: 'User already exists'

 If userType not in ['Admin', 'Farmer', 'Customer', 'Seller']:

 Return 400: 'Invalid user type'

 If userType == 'Admin' and AdminExists():

 Return 400: 'Only one admin is allowed.'

 (isValid, message) = ValidatePassword(password)

 If isValid is False:

 Return 400: message

 hashedPassword = HashPassword(password)

 otpCode = GenerateOTP()

 hashedOtp = HashOTP(otpCode)

 newUser = CreateUser(username, firstName, lastName, email, hashedPassword,
phoneNumber, userType, hashedOtp, OTPExpireTime, isActive=False)

 SaveUser(newUser)

 SendOTPEmail(email, username, otpCode)

 Return 201: 'OTP sent to your email for account activation'

```
Catch error:
  If ValidationError in error:
    Return 400: error.message
  LogError(error)
  Return 500: 'Error registering user'
End Function
```

7.1.2 Login User

Input: email, password, rememberMe
Output: JSON response with success or error message

```
Try:
  user = FindUserByEmail(email, includePassword=True)
  If user does not exist:
    Return 404: 'User not found'

  If user.isActive is False:
    Return 403: 'Account is not active. Please verify your email.'

  If user.isLoggedIn is True:
    Return 400: 'User is already logged in. Please log out first.'

  If PasswordsDoNotMatch(password, user.password):
    Return 400: 'Invalid credentials'

  expiresIn = '7d' if rememberMe else '1h'
  token = GenerateToken(user.id, expiresIn)

  MarkUserAsLoggedIn(user)
  Return 200: 'Login successful', token, expiresIn

Catch error:
  LogError(error)
  Return 500: 'Error logging in'
End Function
```

7.1.3 Google Login

Input: email, googleId, name, profilePicture
Output: JSON response with success or error message

```
Try:
  If any of email, googleId, or name is missing:
    Return 400: 'Missing required fields'

  user = FindUserByEmail(email)
  If user does not exist:
    user = CreateUser(username=name, email=email, googleId=googleId,
profilePicture=profilePicture, userType='Customer', isActive=True)
    SaveUser(user)
  Else:
```

```

    If user.googleId is empty and googleId is provided:
        UpdateGoogleId(user, googleId)

    token = GenerateToken(user.id, '7d')
    Return 200: 'Google login successful', user, token

Catch error:
    LogError(error)
    Return 500: 'Error logging in with Google'
End Function

```

7.1.4 Logout User

```

Input: token from Authorization header
Output: JSON response with success or error message

If token is missing:
    Return 401: 'Token not provided'

Try:
    user = FindUserById(request.user.id)
    If user does not exist:
        Return 404: 'User not found. Please check the token.'

    If user.isLoggedIn is False:
        Return 400: 'User is already logged out. Please log in first.'

    MarkUserAsLoggedOut(user)
    Return 200: 'Logout successful'

Catch error:
    LogError(error)
    Return 500: 'Error logging out'
End Function

```

7.1.5 Recommend Crop Using Random Forest Classifier:

STEP 1:

```

Function SetUpFlaskAppAndLoadModels():
    app = InitializeFlaskApp()
    EnableCORS(app)

    model = LoadModelFromPickle("model.pkl")
    sc = LoadScalerFromPickle("standscaler.pkl")
    ms = LoadScalerFromPickle("minmaxscaler.pkl")

    Return app, model, sc, ms

```

End Function

STEP 2:

```

Function ConnectToMongoDB():
    Try:
        // Connect to MongoDB database using the URI

```

```

    client
ConnectToMongoDBWithURI("mongodb+srv://Moiz:Moiz123@agrotech.mqmqge.mongodb.net/")
    db = GetMongoDBDatabase(client, "agrotech")
    crop_recommendations_collection = GetMongoDBCollection(db, "croprecommendations")
    Print("Connected to MongoDB")
Catch Exception as e:
    Print("Error connecting to MongoDB: " + e)

```

```

    Return crop_recommendations_collection
End Function

```

STEP 3:

```
Function DefinePredictRoute(app, model, sc, ms, crop_recommendations_collection):
```

```

    // Define the API route for crop prediction
    app.Route("/api/predict", method="POST", Function predict):
        // Parse input JSON data from the request
        data = ParseJSONRequest(request)

        N = ConvertToFloat(data["Nitrogen"])
        P = ConvertToFloat(data["Phosphorus"])
        K = ConvertToFloat(data["Potassium"])
        temp = ConvertToFloat(data["Temperature"])
        humidity = ConvertToFloat(data["Humidity"])
        ph = ConvertToFloat(data["Ph"])
        rainfall = ConvertToFloat(data["Rainfall"])

        feature_list = [N, P, K, temp, humidity, ph, rainfall]
        single_pred = ReshapeToArray(feature_list)

        scaled_features = ms.Transform(single_pred)
        final_features = sc.Transform(scaled_features)

        // Predict the crop using the model
        prediction = model.Predict(final_features)

        crop_dict = MapPredictionToCrop(prediction)
        crop = crop_dict[prediction[0]]

        recommendation_data = CreateRecommendationData(data, crop)

        // Save the recommendation data to MongoDB
        SaveToMongoDB(crop_recommendations_collection, recommendation_data)

        // Return a success response with the recommendation data
        Return CreateJSONResponse(True, "Crop recommendation saved successfully",
recommendation_data)
    End Function

```

STEP 4:

```
Function Main():
```

```

    // Set up Flask app, model, and scalers
    app, model, sc, ms = SetUpFlaskAppAndLoadModels()

    crop_recommendations_collection = ConnectToMongoDB()

    // Define the API route for crop prediction

```

```
DefinePredictRoute(app, model, sc, ms, crop_recommendations_collection)
```

```
RunFlaskApp(app, debug=True, port=5001)
```

```
End Function
```

Function that is using Random Forest Classifier:

```
def recommendation(N,P,k,temperature,humidity,ph,rainfal):  
    features = np.array([[N,P,k,temperature,humidity,ph,rainfal]])  
    transformed_features = ms.fit_transform(features)  
    prediction = rfc.predict(transformed_features)  
    print(prediction)  
    return prediction[0]
```

```
End Function
```

7.2 External APIs/SDKs

Describe the third-party APIs/SDKs used in the project implementation in the following table. Few examples of APIs are provided in the table.

Name of API and version	Description of API	Purpose of usage	List down the API endpoint/function/class in which it is used
Stripe (version 2020-08-27)	Credit Card payment integration	Sandbox used for the orders payment	stripe.paymentMethods.create
JamAI	ChatBot Text Generation	Ai Based Text Generation for Chatting with users	https://api.jamai.com/v1/generate
Open-Mateo	Open-Source Weather Api	Fetches weather and humidity data for crop recommendation	fetch_weather_data(location)
Firebase	Backend-as-a-Service platform	Google Authentication for user login	firebase.auth().signInWithPopup(), firebase.auth().signOut()

Table 3: External APIs/SDKs

7.3 User Interface

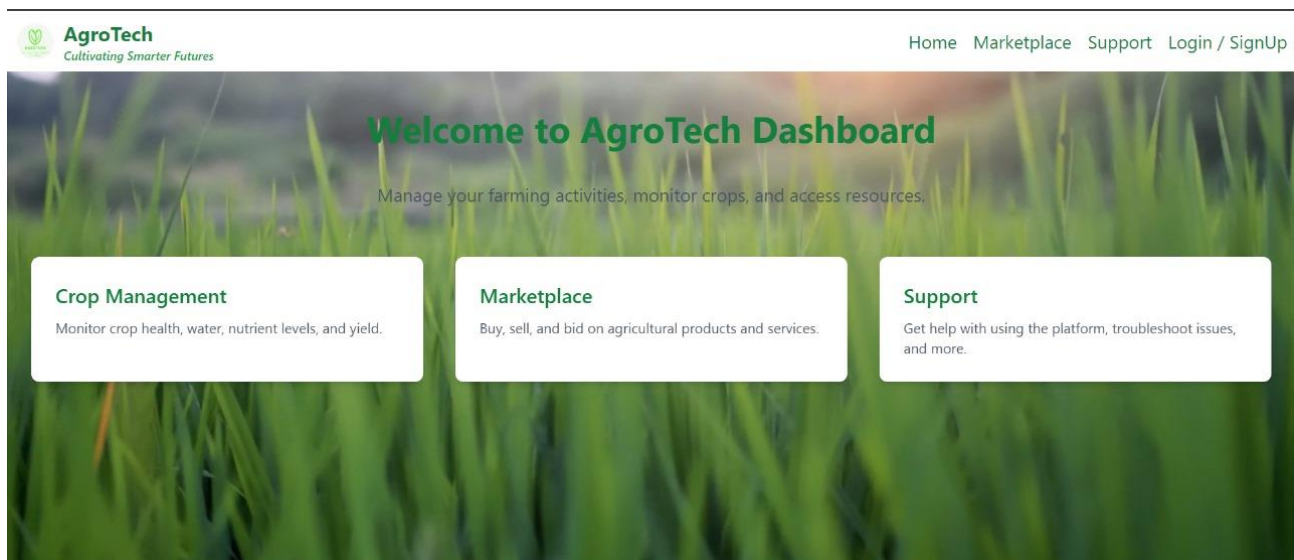
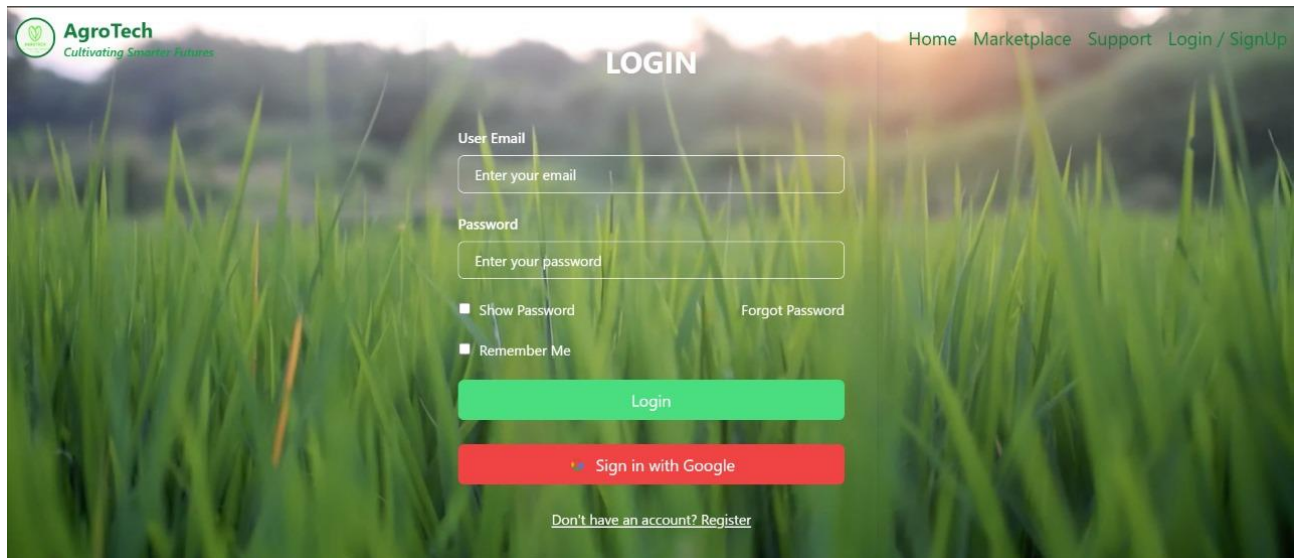


Figure 67: Home page



The login page features a green header with the AgroTech logo and tagline 'Cultivating Smarter Futures'. The background is a lush green field. The main heading is 'LOGIN'. The form includes fields for 'User Email' and 'Password', both with placeholder text 'Enter your email' and 'Enter your password' respectively. There are checkboxes for 'Show Password' and 'Remember Me', and a link for 'Forgot Password'. A green 'Login' button and a red 'Sign in with Google' button are present. At the bottom, a link says 'Don't have an account? Register'.

AgroTech
Cultivating Smarter Futures

Home Marketplace Support Login / SignUp

LOGIN

User Email
Enter your email

Password
Enter your password

☐ Show Password [Forgot Password](#)

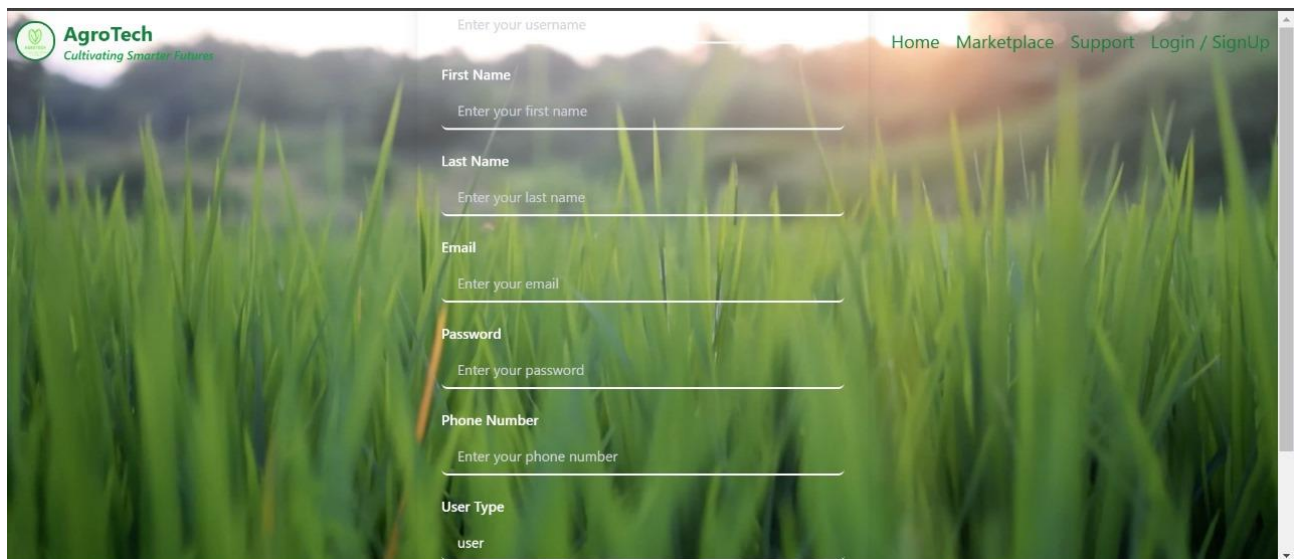
☐ Remember Me

Login

[Sign in with Google](#)

[Don't have an account? Register](#)

Figure 69: Login



The sign up page features a green header with the AgroTech logo and tagline 'Cultivating Smarter Futures'. The background is a lush green field. The main heading is 'SIGN UP'. The form includes fields for 'First Name', 'Last Name', 'Email', 'Password', 'Phone Number', and 'User Type'. Each field has a placeholder text. The 'User Type' field has a dropdown menu with 'user' selected. There are links for 'Home', 'Marketplace', 'Support', and 'Login / SignUp' in the top right corner.

AgroTech
Cultivating Smarter Futures

Home Marketplace Support Login / SignUp

SIGN UP

Enter your username

First Name
Enter your first name

Last Name
Enter your last name

Email
Enter your email

Password
Enter your password

Phone Number
Enter your phone number

User Type
user

Figure 68: Sign up

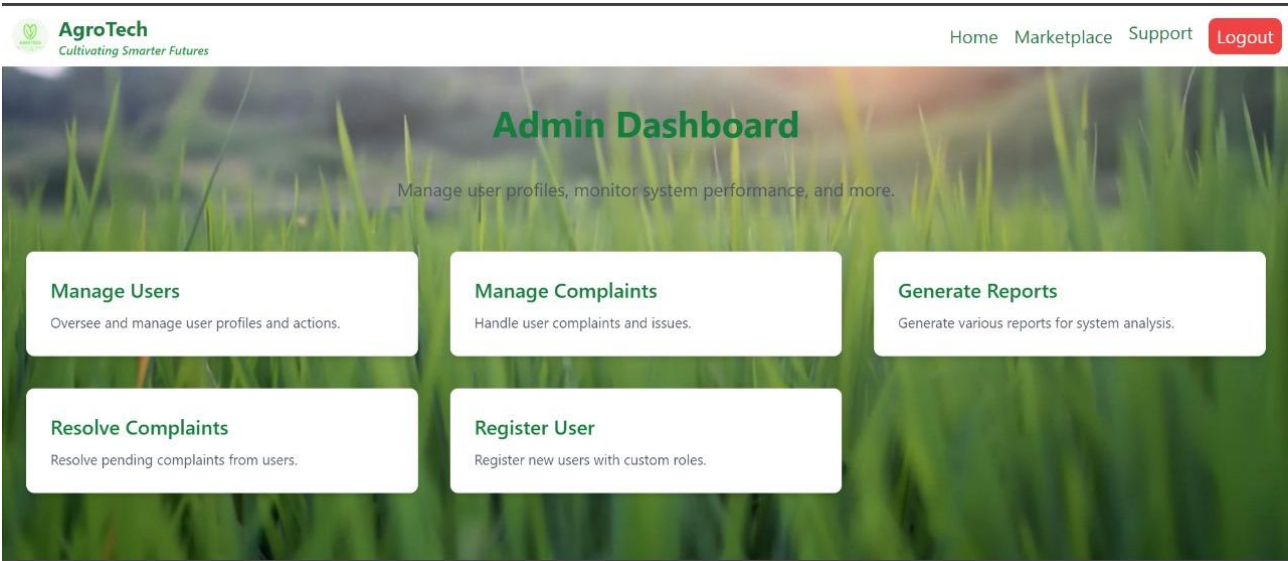


Figure 70: Admin Dashboard

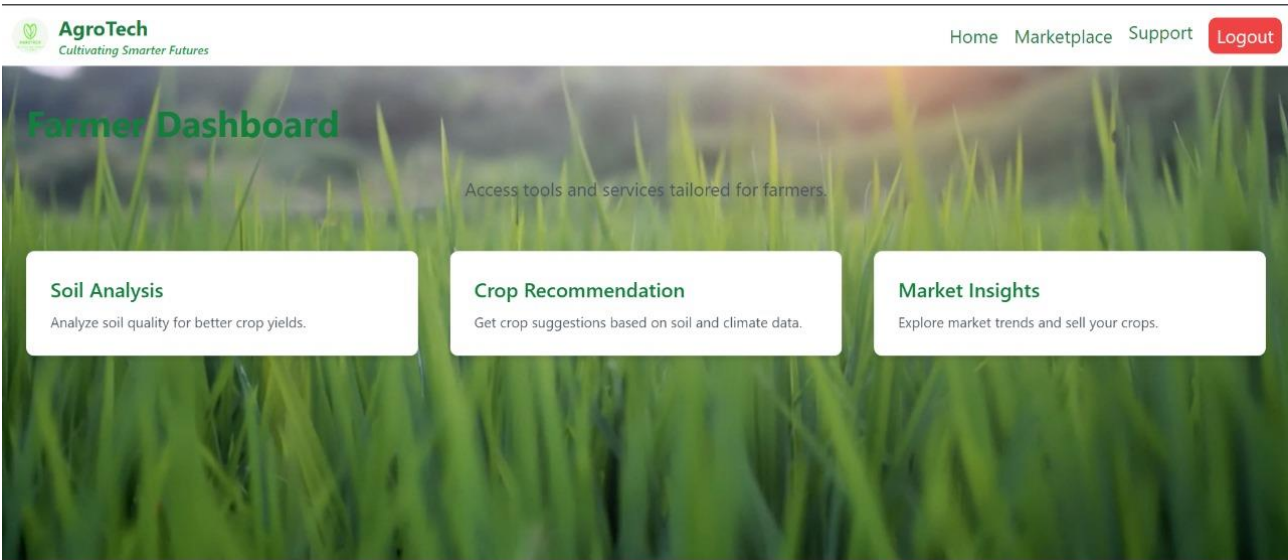
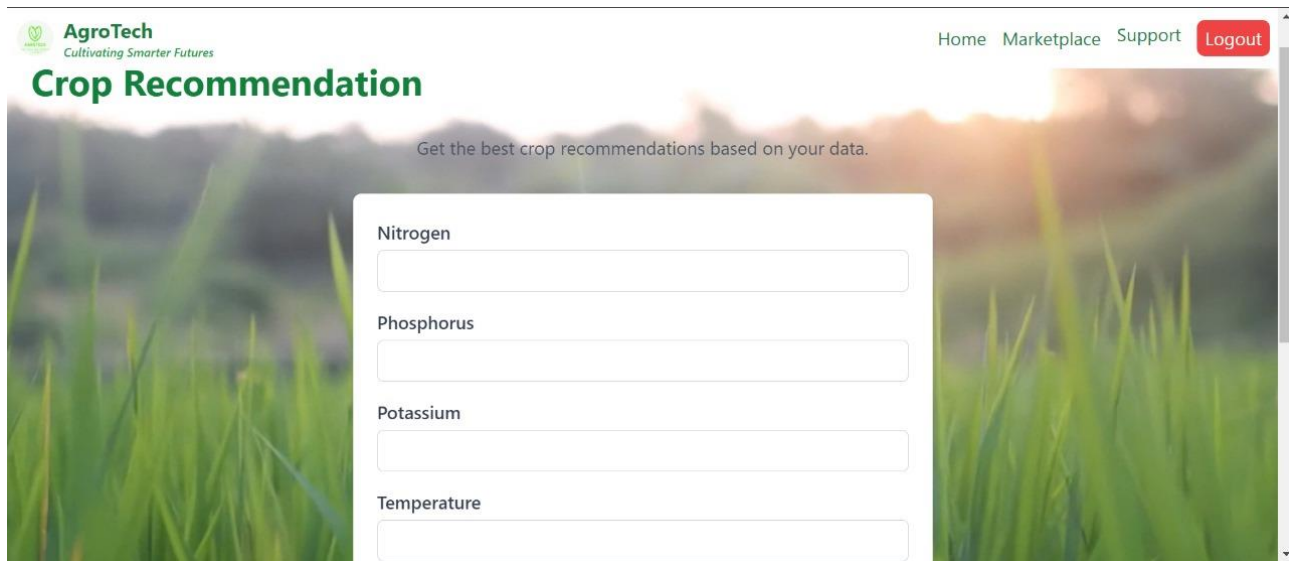


Figure 71: Farmer Dashboard



AgroTech
Cultivating Smarter Futures

Home Marketplace Support Logout

Crop Recommendation

Get the best crop recommendations based on your data.

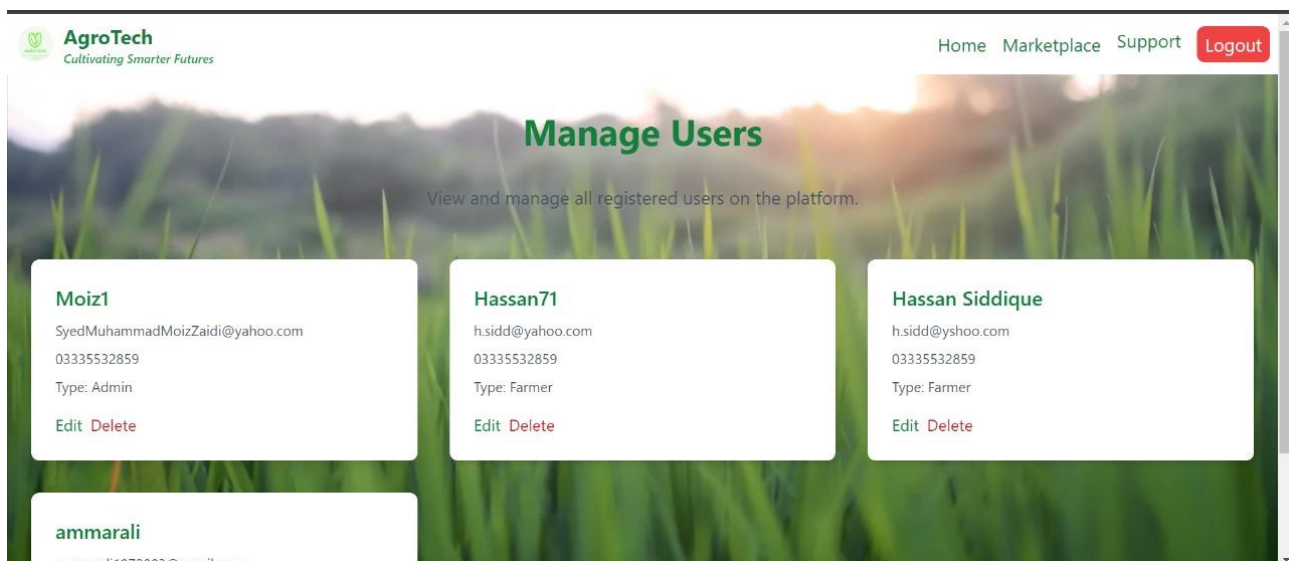
Nitrogen

Phosphorus

Potassium

Temperature

Figure 73: Crop Recommendation



AgroTech
Cultivating Smarter Futures

Home Marketplace Support Logout

Manage Users

View and manage all registered users on the platform.

Moiz1 SyedMuhammadMoizZaidi@yahoo.com 03335532859 Type: Admin Edit Delete	Hassan71 h.sidd@yahoo.com 03335532859 Type: Farmer Edit Delete	Hassan Siddique h.sidd@yshoo.com 03335532859 Type: Farmer Edit Delete
ammarali ammarali1973002@gmail.com		

Figure 72: Manage Users

7.4 Deployment

The AgroTech application's frontend, developed using the React.js framework, will be hosted on **Vercel**, a cloud platform optimized for frontend frameworks and static site generation. Vercel ensures seamless deployment, optimized performance, and scalability, leveraging its Jamstack architecture. The current version of Vercel being utilized is **34.2.0**, providing cutting-edge features for frontend deployment.

For the backend services, we plan to utilize **Heroku**, **Microsoft Azure**, or **Amazon Web Services (AWS)**, depending on the application's scalability, reliability, and infrastructure needs. This combination of Vercel for frontend hosting and one of these leading cloud platforms for backend services ensures a robust, scalable, and efficient deployment solution for the AgroTech application.

8. Testing and Evaluation

8.1 Unit Testing

Unit Testing 1: Admin Controller

Testing Objective: To ensure the controller is working correctly with valid and invalid credentials/inputs.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Result
1	Fetch all registered users and verify they are returned without passwords.	Request: GET /users	Responds with a list of users without their passwords.	Pass
2	Update a user's details successfully.	Request: PUT /users/:userIdBody: { "username": "NewName", "email": "newemail@example.com", "userType": "Farmer" }	Responds with status 200 and updated user details.	Pass
3	Attempt to update details for a non-existent user.	Request: PUT /users/:invalidUserId	Responds with status 404 and message "User not found".	Pass
4	Delete a user successfully and send a deletion email.	Request: DELETE /users/:userId	Responds with status 200 and message "User deleted successfully". Email is sent to the user.	Pass
5	Attempt to delete a non-existent	Request: DELETE /users/:invalidUserId	Responds with status 404 and	Pass

	user.		message "User not found".	
6	Fetch all complaints successfully.	Request: GET /complaints	Responds with a list of all complaints.	Pass
7	Resolve a complaint and mark it as "resolved".	Request: PUT /complaints/resolveBody: { "complaintId": "12345", "action": "resolved", "resolutionText": "Resolved explanation" }	Responds with status 200 and updated complaint details.	Pass
8	Attempt to resolve a non-existent complaint.	Request: PUT /complaints/resolveBody: { "complaintId": "invalidId", "action": "resolved" }	Responds with status 404 and message "Complaint not found".	Pass
9	Generate a report successfully.	Request: POST /reportsBody: { "reportType": "Sales", "filters": { "year": 2023 } }	Responds with status 200 and new report details.	Pass
10	Fetch all reports and verify the "generatedBy" field is populated.	Request: GET /reports	Responds with a list of reports with "generatedBy" field populated with user details.	Pass
11	Register a user as an admin with valid details.	Request: POST /users/registerByAdminBody: { "username": "JohnDoe", "email": "johndoe@example.com", "password": "Secure123!", "userType": "Farmer" }	Responds with status 201 and message "User created successfully and welcome email sent".	Pass
12	Attempt to register a user with an invalid user type.	Request: POST /users/registerByAdminBody: { "username": "JaneDoe", "email": "janedoe@example.com", "password": "Secure123!", "userType": "InvalidType" }	Responds with status 400 and message "Invalid user type".	Pass
13	Attempt to register a user with an existing email.	Request: POST /users/registerByAdminBody: { "username": "JaneDoe", "email": "existingemail@example.com", "password": "Secure123!", "userType": "Farmer" }	Responds with status 400 and message "User already exists".	Pass

Table 4: Admin Controller Unit Test Cases

Unit Testing 2: AUTH Controller

Testing Objective: To ensure the controller is working correctly with valid and invalid credentials/inputs.

No.	Function	Test Case/Test Script	Input Example	Expected Result	Result
1	registerUser	Register a new user with valid details.	Body: { "username": "John", "firstName": "John", "lastName": "Doe", "email": "john@example.com", "password": "Password123!", "userType": "Farmer", "phoneNumber": "1234567890" }	Responds with status 201 and message "OTP sent to your email for account activation".	Pass
2	registerUser	Register a user with an existing email.	Body: { "email": "existingemail@example.com", ... }	Responds with status 400 and message "User already exists".	Pass
3	registerUser	Register a user with an invalid userType.	Body: { "userType": "InvalidType", ... }	Responds with status 400 and message "Invalid user type".	Pass
4	registerUser	Register another admin when one already exists.	Body: { "userType": "Admin", ... }	Responds with status 400 and message "An admin is already registered. Only one admin is allowed".	Pass
5	registerUser	Submit an invalid OTP during account activation.	Body: { "email": "john@example.com", "otp": "123456" }	Responds with status 400 and message "Invalid OTP. Please try again".	Pass
6	registerUser	Submit a valid OTP for account activation.	Body: { "email": "john@example.com", "otp": "validOTP" }	Responds with status 200 and message "Account activated successfully. Welcome email".	Pass

				sent".	
7	registerUser	Submit an OTP after expiration.	Body: { "email": "john@example.com", "otp": "expiredOTP" }	Responds with status 400 and message "OTP expired. Please register again".	Pass
8	loginUser	Login with valid credentials.	Body: { "email": "john@example.com", "password": "Password123!" }	Responds with status 200 and message "Login successful" along with user details and token.	Pass
9	loginUser	Login with incorrect password.	Body: { "email": "john@example.com", "password": "WrongPassword" }	Responds with status 400 and message "Invalid credentials".	Pass
10	loginUser	Login for a user who has not activated their account.	Body: { "email": "inactive@example.com", "password": "Password123!" }	Responds with status 403 and message "Account is not active. Please verify your email".	Pass
11	loginUser	Attempt to login a user who is already logged in.	Body: { "email": "loggedin@example.com", "password": "Password123!" }	Responds with status 400 and message "User is already logged in. Please log out first".	Pass
12	googleLogin	Perform Google login for a new user.	Body: { "email": "googleuser@example.com", "googleId": "Google123", "name": "Google User", "profilePicture": "http://picture.com/image.png" }	Responds with status 200 and message "Google login successful" with user details and token.	Pass
13	googleLogin	Perform Google login for an existing user without a	Body: { "email": "existinguser@example.com", "googleId": "Google123", ... }	Updates the user with Google ID and responds with status 200 and	Pass

		Google ID.		message "Google login successful".	
14	googleLogin	Perform Google login with missing required fields.	Body: { "email": "user@example.com", "googleId": "" }	Responds with status 400 and message "Missing required fields".	Pass
15	logoutUser	Logout a logged-in user with a valid token.	Header: Authorization: Bearer <token>	Responds with status 200 and message "Logout successful".	Pass
16	logoutUser	Attempt to logout without a token.	Header: Authorization not provided.	Responds with status 401 and message "Token not provided".	Pass
17	logoutUser	Attempt to logout a user who is already logged out.	Header: Authorization: Bearer <token>	Responds with status 400 and message "User is already logged out. Please log in first".	Pass
18	logoutUser	Logout with an invalid token.	Header: Authorization: Bearer <invalidToken>	Responds with status 404 and message "User not found. Please check the token".	Pass

Table 5: AUTH Controller Unit Test Cases

Unit Testing 3: User Controller

Testing Objective: To ensure the controller is working correctly with valid and invalid credentials/inputs.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Result
1	Check the email field of forgotPassword to validate that it takes a proper email.	Email: abc@gmail.com	Validates email address and proceeds to generate a reset token.	Pass
2	Check the email field of forgotPassword to validate that it displays an error for	Email: abc.gmail.com	Highlights field and returns { message: 'User not found' }.	Pass

	invalid email.			
3	Check the forgotPassword function with a valid email to ensure email is sent successfully.	Email: validuser@example.com	Returns { message: 'Reset email sent successfully' } and saves token and expiration in the database.	Pass
4	Check the forgotPassword function with an invalid email to validate error handling.	Email: invaliduser@example.com	Returns { message: 'User not found' }.	Pass
5	Check the resetPassword function with a valid token and new password.	Token: validToken, NewPassword: NewPass123	Returns { message: 'Password reset successfully' }, clears token and expiration in the database.	Pass
6	Check the resetPassword function with an expired or invalid token.	Token: expiredToken	Returns { message: 'Invalid or expired token' }.	Pass
7	Check the updateProfile function to ensure all valid fields update successfully.	Valid fields for firstName, lastName, etc.	Returns { message: 'Profile updated successfully', user: updatedUser }.	Pass
8	Check the updateProfile function with missing fields to ensure partial updates work correctly.	Only firstName or email	Updates only provided fields and keeps others unchanged.	Pass
9	Check the getUserProfile function to fetch user profile successfully.	User ID: validUserId	Returns { user: userDetails }.	Pass
10	Check the getUserProfile function with an invalid user ID.	User ID: invalidUserId	Returns { message: 'User not found' }.	Pass
11	Check the changePassword function with correct current password and new password.	CurrentPassword: correctPass, NewPassword: 123	Returns { message: 'Password Updated Successfully' }.	Pass
12	Check the changePassword function with incorrect current password.	CurrentPassword: wrongPass	Returns { message: 'Current Password is Invalid' }.	Pass
13	Check the deleteUserAccount function to ensure account deletion works.	User ID: validUserId	Returns { message: 'User account deleted successfully' }.	Pass
14	Check the deleteUserAccount	User ID: invalidUserId	Returns { message: 'Error deleting	Pass

	function with an invalid user ID to ensure proper error handling.		account', error }.	
--	---	--	--------------------	--

Table 6: User Controller Unit Test Cases

8.2 Functional Testing

Functional Testing 1: Admin Controller

Objective: To ensure that the controller is working properly.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Login as an Admin to view registered users.	Admin credentials: Username: admin Password: 1234	Successfully logged in as Admin. List of all registered users displayed (excluding passwords).	List of users displayed correctly.	Pass
2	Attempt to fetch registered users as non-admin.	Non-admin credentials: Username: customer123 Password: password	Access denied with a 403 Forbidden response.	Correct error message returned.	Pass
3	Update user details by Admin.	User ID: 12345 Username: newUsername Email: newemail@example.com	User details updated successfully with a confirmation message.	User details updated in the database.	Pass
4	Attempt to update user details with invalid ID.	User ID: invalidID Email: invalidemail	"User not found" error returned with a 404 response.	Correct error message displayed.	Pass
5	Delete a user by Admin.	User ID: 67890	User deleted successfully. Email notification sent to the deleted user.	User deleted and email sent successfully.	Pass
6	Attempt to delete a non-existent user.	User ID: invalidID	"User not found" error returned with a 404 response.	Correct error message displayed.	Pass
7	Fetch all complaints raised in the	None	List of complaints displayed successfully.	Complaints fetched and displayed.	Pass

	system.				
8	Resolve a complaint.	Complaint ID: 98765 Action: "resolved" Resolution Text: "Issue addressed."	Complaint status updated to "resolved" with the resolution text saved.	Complaint resolved and status updated.	Pass
9	Attempt to resolve a non-existent complaint.	Complaint ID: invalidID	"Complaint not found" error returned with a 404 response.	Correct error message displayed.	Pass
10	Generate a report.	Report Type: "User Statistics" Filters: None	Report created successfully with data logged in the system.	Report generated and saved successfully.	Pass
11	Fetch all generated reports.	None	List of all reports displayed successfully.	Reports fetched and displayed correctly.	Pass
12	Register a user by Admin.	Username: newUser Email: user@example.com Password: pass123 User Type: Farmer	User created successfully. Welcome email sent to the user.	User created and email sent successfully.	Pass
13	Attempt to register a user with an invalid type.	Username: invalidUser Email: invalid@example.com Password: pass123 User Type: InvalidType	"Invalid user type" error returned with a 400 response.	Correct error message displayed.	Pass
14	Attempt to register a user with an existing email.	Username: newUser Email: existing@example.com Password: pass123 User Type: Customer	"User already exists" error returned with a 400 response.	Correct error message displayed.	Pass

Table 7: Admin Controller Functional Test Cases

Functional Testing 2: AUTH Controller**Objective:** To ensure that the controller is working properly.

No .	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Register a new user with valid data.	Username: JohnDoeEmail: johndoe@example.comPassword: Pass@1234UserType: Farmer	OTP is sent to the registered email, and a message confirms successful registration.	OTP email received, and confirmation message displayed.	Pass
2	Register	Email: johndoe@example.com	Registration	Registration	Pass

	with an existing email.		fails with a message: "User already exists."	failed with expected message.	
3	Register with invalid password.	Password: pass123	Registration fails with a message: "Password must contain at least 8 characters, one uppercase, one lowercase, one number, and one special character."	Registration failed with expected message.	Pass
4	Register as Admin when an admin already exists.	UserType: Admin	Registration fails with a message: "An admin is already registered. Only one admin is allowed."	Registration failed with expected message.	Pass
5	Activate account using valid OTP.	OTP: (Valid OTP sent to the email during registration)	Account is activated successfully, and a welcome email is sent to the registered email.	Account activated and welcome email received.	Pass
6	Activate account using expired OTP.	OTP: 123456 (expired)	Activation fails with a message: "OTP expired. Please register again."	Activation failed with expected message.	Pass
7	Activate account using invalid OTP.	OTP: 999999 (invalid)	Activation fails with a message: "Invalid OTP. Please try again."	Activation failed with expected message.	Pass
8	Login with	Email: johndoe@example.comPassword:	Login	Logged in	Pass

	valid credentials.	Pass@1234	successful, JWT token is generated, and the user is redirected to their dashboard.	and redirected successfully.	
9	Login with invalid credentials.	Email: johndoe@example.comPassword: wrongpass	Login fails with a message: "Invalid credentials."	Login failed with expected message.	Pass
10	Login for an inactive account.	Email: janedoe@example.comPassword: Pass@1234	Login fails with a message: "Account is not active. Please verify your email."	Login failed with expected message.	Pass
11	Login for a logged-in user.	Email: johndoe@example.comPassword: Pass@1234 (while the user is logged in)	Login fails with a message: "User is already logged in. Please log out first."	Login failed with expected message.	Pass
12	Google login with new account.	Email: newgoogleuser@example.comGoogleID: 12345Name: Google User	User is created, marked as active, and redirected with a valid token.	Account created, token generated successfully.	Pass
13	Google login for existing account.	Email: existinguser@example.comGoogleID: 67890Name: Existing User	User details are updated with the Google ID, and the user is logged in successfully.	Logged in successfully.	Pass
14	Logout for an active user.	Token: Valid JWT token	User is logged out, and isLoggedIn is set to false.	Logged out successfully.	Pass
15	Logout without providing a token.	No token provided	Logout fails with a message: "Token not provided."	Logout failed with expected message.	Pass

16	Logout for an already logged-out user.	Email: johndoe@example.com (while isLoggedIn is false)	Logout fails with a message: "User is already logged out. Please log in first."	Logout failed with expected message.	Pass
----	--	--	---	--------------------------------------	------

Table 8: AUTH Controller Functional Test Cases

Functional Testing 3: User Controller**Objective:** To ensure that the controller is working properly.

No.	Test Case/Test Script	Attribute and Value	Expected Result	Actual Result	Result
1	Forgot Password with valid email.	Email: johndoe@example.com	A password reset email is sent successfully to the registered email address with a reset URL.	Password reset email sent successfully.	Pass
2	Forgot Password with non-existing email.	Email: notfound@example.com	A message is returned: "User not found."	Message: "User not found."	Pass
3	Forgot Password and failure to send email.	Email: johndoe@example.com	Failure message returned: "Error sending email. Please try again later."	Error sending email, message returned as expected.	Pass
4	Reset Password with valid token.	Token: valid-token NewPassword: NewPass@1234	The password is updated successfully, and the user is able to log in with the new password.	Password updated successfully.	Pass
5	Reset Password with expired token.	Token: expired-token NewPassword: NewPass@1234	The reset fails with the message: "Invalid or expired token."	Password reset failed with expected message.	Pass

6	Reset Password with invalid token.	Token: invalid-tokenNewPassword: NewPass@1234	The reset fails with the message: "Invalid or expired token."	Password reset failed with expected message.	Pass
7	Update Profile with valid data.	FirstName: JohnLastName: DoeEmail: newemail@example.comPhoneNumber: 1234567890	Profile updated successfully with new data.	Profile updated with new data.	Pass
8	Update Profile with missing data.	Email: newemail@example.comPhoneNumber: 1234567890	Only provided fields are updated (first name, last name, user type remain unchanged).	Profile updated with partial data.	Pass
9	Update Profile for non-existing user.	User not found	The system returns: "User not found."	User not found message displayed.	Pass
10	Get User Profile for logged-in user.	Valid JWT token	The user profile is returned successfully.	User profile returned successfully.	Pass
11	Get User Profile for non-existing user.	Invalid JWT token	A message is returned: "User not found."	User not found message displayed.	Pass
12	Change Password with correct current password.	CurrentPassword: OldPass@1234NewPassword: NewPass@1234	The password is successfully updated, and the user can log in with the new password.	Password updated successfully.	Pass
13	Change Password with incorrect current password.	CurrentPassword: WrongPass@1234NewPassword: NewPass@1234	Error message returned: "Current password is invalid."	Error message returned as expected.	Pass
14	Change Password with missing current password.	NewPassword: NewPass@1234	Error message returned: "Current password is required."	Error message returned as expected.	Pass

15	Delete User Account for logged-in user.	Valid JWT token	The user account is deleted successfully, and a success message is returned.	User account deleted successfully.	Pass
16	Delete User Account for non-existing user.	Invalid JWT token	Error message returned: "User not found."	Error message returned as expected.	Pass

Table 9: User Controller Functional Test Cases

8.3 Business Rules Testing

8.3.1 Admin Controller

Condition	R1	R2	R3	R4	R5	R6	R7	R8
Username exists	T	T	F	F	T	T	F	F
Password is correct	T	F	T	F	T	F	T	F
Account is active	T	T	T	T	F	F	F	F
Actions								
Login successful	T	F	F	F	F	F	F	F
Display "Invalid username" error	F	F	T	T	F	F	T	T
Display "Invalid password" error	F	T	F	T	F	T	F	T

Table 10: Admin Controller Business Rules Testing

8.3.2 AUTH Controller

Condition	R1	R2	R3	R4	R5	R6	R7	R8
Username exists	T	T	F	F	T	T	F	F
Password is correct	T	F	T	F	T	F	T	F
Account is active	T	T	T	T	F	F	F	F
Actions								
Login successful	T	F	F	F	F	F	F	F
Display "Invalid username" error	F	F	T	T	F	F	T	T
Display "Invalid password" error	F	T	F	T	F	T	F	T

Table 11: AUTH Controller Business Rules Testing

8.3.3 User Controller

Condition	R1	R2	R3	R4	R5	R6	R7	R8
Username exists	T	T	F	F	T	T	F	F
Password is correct	T	F	T	F	T	F	T	F
Account is active	T	T	T	T	F	F	F	F
Actions								

Login successful	T	F	F	F	F	F	F	F
Display "Invalid username" error	F	F	T	T	F	F	T	T
Display "Invalid password" error	F	T	F	T	F	T	F	T
Display "Account is not active" error	F	F	F	F	T	T	T	T
Display "Token expired" error	F	F	F	F	F	T	T	T
Display "Password reset successfully"	F	F	F	T	T	F	F	F
Display "Error updating password"	F	F	F	F	F	F	T	F
Display "Account deleted successfully"	F	F	F	F	F	F	F	T

Table 12: User Controller Business Rules Testing

8.4 Integration Testing

Integration Testing 1: Register User

Testing Objective: To ensure that user is registered after proper process.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Register User (Valid)	username: "johnDoe", email: " john@example.com ", password: "Password123", userType: "Farmer"	Successfully register user and send OTP to email for verification.	OTP sent to user email for account activation.	Pass
2.	Register User (User Exists)	email: " john@example.com " (Already exists)	Return error "User already exists".	Error returned: "User already exists".	Pass
3.	Register User (Invalid User Type)	userType: "Admin" (Admin already registered)	Return error "An admin is already registered".	Error returned: "An admin is already registered".	Pass
4.	Register User (Invalid Password)	password: "123"	Return error "Password is too weak".	Error returned: "Password is too weak".	Pass
5.	Register User (OTP Verification)	otp: "123456"	Successfully verify OTP and activate the user account.	Account activated, welcome email sent.	Pass

Table 13: Register User Integration Testing

Integration Testing 2: Login User**Testing Objective:** To ensure that login is properly verified.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Login User (Valid)	email: " john@example.com ", password: "Password123", rememberMe: true	Successfully login user and return token.	User logged in, token generated.	Pass
2.	Login User (Invalid Credentials)	email: " john@example.com ", password: "WrongPassword"	Return error "Invalid credentials".	Error returned: "Invalid credentials".	Pass
3.	Login User (Inactive Account)	email: " john@example.com ", password: "Password123", isActive: false	Return error "Account is not active. Please verify your email".	Error returned: "Account is not active. Please verify your email".	Pass
4.	Login User (Already Logged In)	email: " john@example.com ", password: "Password123", isLoggedIn: true	Return error "User is already logged in. Please log out first".	Error returned: "User is already logged in. Please log out first".	Pass

Table 14: Login User Integration Testing**Integration Testing 3: Forgot Password****Testing Objective:** To ensure that email is sent for forgotten password.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Forgot Password (Valid)	email: " john@example.com "	Successfully send reset password email.	Reset email sent.	Pass
2.	Forgot Password (User Not Found)	email: " notfound@example.com "	Return error "User not found".	Error returned: "User not found".	Pass
3.	Forgot Password (Email Sending Failure)	email: " john@example.com " (Simulate email sending failure)	Return error "Error sending email".	Error returned: "Error sending email".	Pass

Table 15: Forgot Password Integration Testing

Integration Testing 4: Reset Password**Testing Objective:** To ensure that password is reset securely and properly.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Reset Password (Valid Token)	token: "validToken", newPassword: "NewPassword123"	Successfully reset the password.	Password reset successfully.	Pass
2.	Reset Password (Invalid or Expired Token)	token: "expiredToken", newPassword: "NewPassword123"	Return error "Invalid or expired token".	Error returned: "Invalid or expired token".	Pass
3.	Reset Password (Invalid Password)	token: "validToken", newPassword: "123"	Return error "Password is too weak".	Error returned: "Password is too weak".	Pass

Table 16: Reset Password Integration Testing**Integration Testing 5: Update Profile****Testing Objective:** To ensure that profile is updating the details.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Update Profile (Valid Update)	firstName: "John", lastName: "Doe", email: " john@example.com ", userType: "Farmer", phoneNumber: "1234567890"	Successfully update profile with new details.	Profile updated with new details.	Pass
2.	Update Profile (User Not Found)	firstName: "John" (User does not exist)	Return error "User not found".	Error returned: "User not found".	Pass

Table 17: Update Profile Integration Testing**Integration Testing 6: Change Password****Testing Objective:** To ensure that change password is securely and properly done.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Change Password (Valid Current Password)	currentPassword: "Password123", newPassword: "NewPassword123"	Successfully change the password.	Password changed successfully.	Pass
2.	Change Password (Invalid Current Password)	currentPassword: "WrongPassword", newPassword: "NewPassword123"	Return error "Current password is invalid".	Error returned: "Current password is invalid".	Pass

Table 18: Change Password Integration Testing

Integration Testing 7: Delete User**Testing Objective:** To ensure user is deleted properly.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Delete User Account (Valid)	userId: "validUserId"	Successfully delete user account.	Account deleted successfully.	Pass
2.	Delete User Account (User Not Found)	userId: "nonExistentUserId"	Return error "User not found".	Error returned: "User not found".	Pass

Table 19: Delete User Integration Testing

9. Plagiarism Report

Document Viewer

Turnitin Originality Report

Processed on: 17-Dec-2024 12:03 AM PST
ID: 2554461197
Word Count: 1158
Submitted: 1

Report By Muaaz Bin Mukhtar .

Similarity Index

0%

Similarity by Source

Internet Sources:	0%
Publications:	0%
Student Papers:	0%

exclude quoted

exclude bibliography

exclude small matches

mode:

quickview (classic) report

print

refresh

download

Figure 74: Plagiarism Report