**Crop Deal**

**Low Level Design (LLD)**

**Table of Content**

1. [Assumption](#Assumption_)…….…………………………………………………………..2
2. [Document Purpose](#Document_purpose) ………………………………………………………...2
3. [Scope](#Scope_)……………………………………………………………………….2
4. [Intended Audience](#Intended_Audience)…………………………………………………………2
5. [Project Background, Objective(s)](#Project_background)………………………………………….3
   1. [Project Background](#Project_background)…………………………………………...3
   2. [Project Objective](#Porject_Objective)……………………………………………...3
6. [Hardware and Software Requirements](#Hardware_and_software_requirments)……………………………………..3
7. [Design Pattern](#Design_pattern)………………………………………………………………4
8. [Solution Diagram](#Solution_diagram)…………………………………………………………...5
9. [Solution](#Solution_steps)…………………………………………………………………….5
10. [Classes/Function](#Classes_function) ………………………………………………………..9
11. [Database Schema](#Database_schema)…………………………………………………….....11
12. [HTTP Status Code](#HTTP_status_code)……………………………………………………...11
13. [Unit Testing](#Unit_testing)…………………………………………………………….12
14. [Request](#Request_)…………………………………………………………………15
15. [Response](#Response_)………………………………………………………………..16
16. [UML Diagram](#UML_Diagram)………………………………………………………….19
    1. [Architecture Diagram](#Architecture_Diagram)……………………………………..19
    2. [Use Case Diagram](#Use_case_diagram)………………………………………...20
    3. [ER Diagram](#ER_diagram)……………………………………………….21
    4. [Sequence Diagram](#Sequence_diagram)………………………………………...22

# Assumption

The assumption is that the Crop Deal application will be extended to support the proposed new features.

The existing architecture and system design will be used including all  
existing components and sub-systems.

It is certain that additional functionality will be added to the proposed solution.

# Document Purpose

This document describes the solution architecture for Crop Deal for Users.

# Scope

Mobile and Web Applications are trending and fastest growing technologies.   
By use of these technology there can be solved various problems of Users.  
The study scope in the field of agriculture sector is very important and effective.  
India stand in first five rank in every agriculture produce and impacts on the Indian economy. The main problem is to solve the productivity and utilization of existing resources effectively. Mobile application can make the revolutionary change in to the Users productivity.

# Intended Audience

This document is intended as a reference for the following roles and stakeholders who are interested in the Crop Deal technical architecture.

|  |  |
| --- | --- |
| Role | Nature of Engagement in WB Classics Portal Technical Architecture |
| Product Owners/SME | Key stakeholder to ensure that the architecture is aligned with business goals. |
| Business Analysts | Business analysts are one of the stakeholders who are informed with the key architectural decisions. |
| Enterprise Architects | To enforce Crop Deal management Platform Architecture is aligned to business goals and architecture, architectural guidelines. |
| Solution Architects | To ensure solution design and architecture is aligned to business requirements, architectural guidelines. |
| Developers | Use Technical Architecture Document as the guiding document for detail design and implantation approach to align with Crop Deal management. |

# Project Background, Objective(s)

## Project Background

Crop Deal leads to perform Management of Users and Dealers details where one can register themselves and perform various operations.

## Project Objective

The Crop Deal Application is being designed to solve the problem of struggle of the Users who are facing to sell their crop in the market for proper price and many more problems such as cost of transportation, wait time, negotiations for proper price, paying lot of intermittent charges as commissions, etc.

# Hardware and Software Requirements

* **Operating Systems**

Windows 10 or Windows 8

* **Hardware Environment**

**Processor:** x86 or x64

**RAM:** 512 MB (minimum), 1 GB (recommended)

**Hard Disc:** up to 3 GB of free space may be required

* **Development Environment**

Microsoft Visual Studio 2019 or 2022.

Visual Studio Code (Text-editor)

.NET Core 3.1 and above.

Command Line (Optional, Necessary for command line deployment)

Internet Information Services (IIS) 7.0+

Microsoft SQL Server Management studio 2018 or 2019

* **Browser support**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chrome** | **Firefox** | **Opera** | **Edge** | **IE** |
| Latest | Latest | Latest | 13 + | 11 + |

# Specification and other Technologies

Angular as front end and Web API as Backend.

**Frontend:** Angular, Bootstrap, HTML, CSS, JavaScript

**Middleware:** .Net Core with Entity frame work, Web API

**Backend:**  MySQL Server.

# Design Pattern

|  |  |  |
| --- | --- | --- |
| **#** | **Name** | **Description** |
| 1 | API | Using HTTP requests, we will use the respective action to trigger various operations |

# Solution Diagram

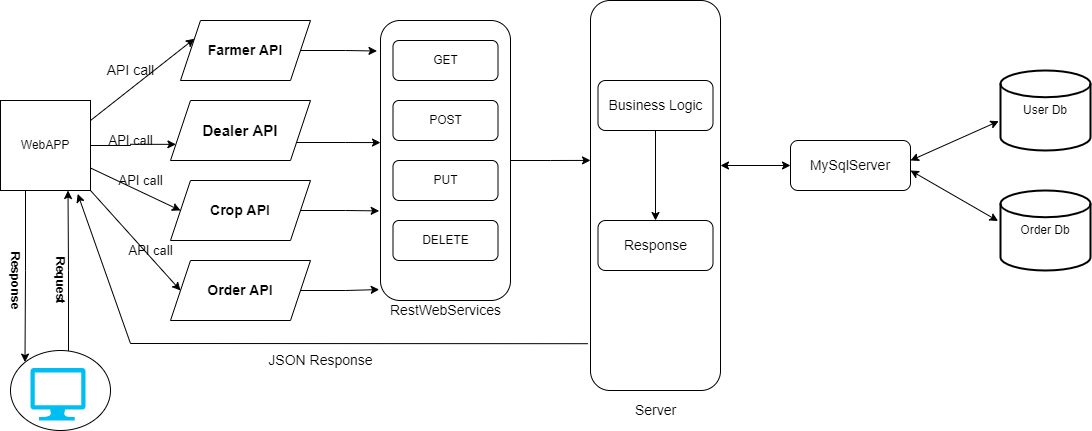
****

Figure 8.0 (Solution Diagram)

# Solution Steps

**UserProfile Registration**

1. User will enter the required details such as UserId, UserfirstName, UserlastName, email, UserMobileNumber, UserType(Farmer,Dealer), UserPassword and click submit button browser directs the request to User registration API
2. Call reaches the api gateway
3. API gateway does the routing and forwards the request to registerUserHandler.handle. And this handle function will call the doProcess ()
4. doProcess () will call the UserschemaValidator.dovalidate() function to do the input validation it will have the UserValidator.validateUserRegistration as argument to perform the validation
5. If validation fails, then it will return the error code and error description (User registration failed), with status code
6. If validation is successful, then the handler will call the registerUserService.registerUser() which will call the registerUserRepository.registerUser() to store the data in database
7. It sends a response body with HTTP Success response to registerUserHandler.
8. registerUserHandler returns JSON Response
9. Success JSON response and HTTP status code 200 with corresponding success message.

**User Listing**

1. User wants to get the User details. Enters the id in parameter for which User wants to see the details. Browser directs the request to User List API.
2. Call reaches the API gateway.
3. API gateway does the routing and forwards the request to listUserHandler.handle this handle function calls the doProcess ().
4. doProcess () will call the listUserService.listUser() which calls the listUserRepository.listUser () to fetch the data from database.
5. It sends response body with HTTP Success response code to listUserHandler.
6. listUserHandler returns JSON Response
7. Success JSON response and HTTP status code 200 with corresponding success message.

**User Updation**

1. User wants to update the details enters the id and the details which User wants to update the details. Browser directs the request to User update API
2. Call reaches the API gateway.
3. API gateway does the routing and forwards the request to update

UserHandler.handle this handle function calls the doProcess ()

1. doProcess () will call the UserschemaValidator.dovalidate() function to do the input validation.
2. If validation fails, then it will return the error code and error description. with status code
3. If validation is successful, then the handler will call the update User Service.updateUser () which will call the updateUserRepository.updateUser() to update the data in database
4. It sends response body with HTTP Success response code to updateUserHandler.
5. updateUserHandler returns JSON Response
6. Success JSON response and status HTTP code 200 with corresponding success message.

**Dealer Listing**

1. Admin wants to get the dealer details. Enters the id in parameter for which dealer wants to see the details. Browser directs the request to Dealer List API.
2. Call reaches the API gateway.
3. API gateway does the routing and forwards the request to listDealerHandler.handle this handle function calls the processFunction ().
4. processFunction () will call the listDealerService.listDealer() which calls the listDealerRepository.listDealer () to fetch the data from database.
5. It sends response body with HTTP Success response code to listDealerHandler.
6. listDealerHandler returns JSON Response
7. Success JSON response and HTTP status code 200 with corresponding success message.

**Dealer Updation**

1. Dealer wants to update the details enters the id and the details which dealer wants to update the details. Browser directs the request to dealer update API
2. Call reaches the API gateway.
3. API gateway does the routing and forwards the request to updateDealerrHandler.handle this handle function calls the doProcess ()
4. processFunction () will call the DealerschemaValidator.dovalidate() function to do the input validation.
5. If validation fails, then it will return the error code and error description. with status code
6. If validation is successful, then the handler will call the update User Service.updateDealer () which will call the updateDealerRepository.updateDealer() to update the data in database
7. It sends response body with HTTP Success response code to updateDealerHandler.
8. updateDealerHandler returns JSON Response
9. Success JSON response and status HTTP code 200 with corresponding success message.

**Admin Login**

1. Admin will enter the details such as AdminId, Password, Email and click submit button browser directs the request to admin login API
2. call reaches the api gateway
3. API gateway does the routing and forwards the request to loginAdminHandler.handle. And this handle function will call the adminProcess ()
4. adminProcess () will call the adminschemaValidator.dovalidate() function to do the input validation it will have the adminValidator.validateadminLogin as argument to perform the validation
   1. If validation fails, then it will return the error code and error description (admin login failed), with status code
   2. If validation is successful, then the handler will call the adminLoginService.loginAdmin () which will call the adminLoginRepository.loginAdmin() to store the data in database
5. It sends a response body with HTTP Success response to loginAdminHandler.

**Admin Role**

1. After Login, Admin has authority to access the Users and dealers data such as profile and transaction details, active/inactive status.
2. Admin can manage all the lists of all the entities such as Crop, Users, Dealers, Order or Transactions. These Lists get add-ons and update after every time data added.
3. Admin can generate reports and advanced wash reports based on business, users and locations.

# Classes/function

|  |  |  |
| --- | --- | --- |
| **#** | **Class** | **Description** |
| 1 | UserProfile.cs | Model holds the Users schema details |
| 2 | registerUserHandler.cs | The handler to handle the registration of Users which calls the registerUserService class |
| 3 | registerUserService.cs | It contains the core business logic for the registration of Users. Which calls the registrationUserRepository class to create the User in database |
| 4 | registerUserRepository.cs | This class deals with the data accessibility for User registration |
| 5 | listUserHandler.cs | The handler to handle the listing of Users. which calls the listUserService class |
| 6 | listUserrService.cs | It contains the core business logic for the registration of Users. Which calls the listUserRepository class to list the User from database |
| 7 | listUserRepository.cs | This class deals with data accessibility for User list |
| 8 | removeUserHandler.cs | The handler to handle the deletion of Users. which calls the removeUserService class |
| 9 | removeUserService.cs | It contains the core business logic for the registration of Users. Which calls the removeUserRepository class to remove the User from database |
| 10 | removeUserRepository.cs | This class deals with data accessibility for User deletion |
| 11 | updateUserHandler.cs | The handler to handle the updation of User details in database which calls the updateUserService class |
| 12 | updateUserService.cs | It contains the core business logic for the registration of Users. Which calls the updateUserRepository class to update the User data in database |
| 13 | updateUserRepository.cs | This class deals with data accessibility for User deletion |
| 14 | UserValidator.cs | It deals with the validation of the inputs provided by the Users. |

# **Database Schema**

**User Database:**

|  |  |  |
| --- | --- | --- |
|  | User | DATA\_TYPES |
| PK | UserId | int |
|  | UserType | nvarchar(200) |
|  | UserFirstName | nvarchar(200) |
|  | UserLastName | nvarchar(200) |
|  | UserMobileNumber | int |
|  | Email | nvarchar (200) |
|  | Password | nvarchar(200) |

User Profile Table

**Order Database:**

Order Transaction

|  |  |  |
| --- | --- | --- |
|  | **ORDER** | **DATA\_TYPES** |
| PK | OrderTrxnId | Int |
|  | CropDate | Date |
|  | OrderQuantity | Nvarchar(100) |
|  | DeliveryDate | date |

Crop List Table

|  |  |  |
| --- | --- | --- |
|  | CROP List | DATA\_TYPE |
| PK | CropListId | int |
|  | AvailQuantity | nvarchar (100) |
|  | Price | int |
|  | CropName | nvarchar (100) |

Payment Table

|  |  |  |
| --- | --- | --- |
|  | Payment | DATA\_TYPE |
| PK | InvoiceId | int |
|  | PaymentMethods | Nvarchar(100) |
|  | CardHolderNumber | Nvarchar(100) |
|  | CardNumber | Nvarchar(100) |
|  | ExpiryDate | Nvarchar(20) |
|  | PayAmoubtBill | Int |
|  | Cvv | int |

**Order Transaction Table**

|  |  |  |
| --- | --- | --- |
|  | **Oder** | **DATA\_TYPE** |
| PK | OrderTrxnId | int |
|  | OrderQuantity | Nvarchar(100) |
|  | OrderDate | date |
|  | UserCreatedBy | Nvarchar(100) |
|  | UserModifiedBy | Nvarchar(100) |
|  | DelivryDate | date |

# Invoice Transaction Table

|  |  |  |
| --- | --- | --- |
|  | **Invoice** | **DATA\_TYPE** |
| PK | InvoiceId | int |
|  | BillingTo | Nvarchar(100) |
|  | DeliverTo | Nvarchar(100) |
|  | UserCreateDate | date |
|  | UserModifiedBy | Nvarchar(100) |
|  | UserModifiedDate | Nvarchar(100) |

# HTTP Status Code

201 – User Registered

200 - Request succeeded

400 – Inputs are invalid

404 – User Not found

502 – Bad gateway

# Unit Testing

|  |  |
| --- | --- |
| **Project Name** | **Crop Deal** |
| Created by | Group 5 |
| Date of Creation |  |
| Date of review |  |

**For Registration of Users**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test CASE ID** | **TEST CASE SCENARIO** | **TEST CASE** | **PRE CONDITION** | **TEST STEPS** | **TEST DATA** | **EXPECTED RESULT** | **Actual Result** |
| TC\_o1 | User registration | Enter the valid data to get registered | User needs to enter all the valid details | 1) Enter  UserId: F001  FFirstName: John  FLastName: Doe  FEmail: User@gmail.com  FContactNo: 123456789  FAdresss: Delhi  Password:\*\*\*\*\*\*  2) Enter Submit | <Valid Details | Successful registration | Successful registration |
| TC\_o2 | User registration | Enter invalid data to get registered | User needs to enter the valid details with wrong phone number type | 1) Enter  UserId: F001  FFirstName: John  FLastName: Doe  FEmail: john@gmail.com  FContactNo: 012256789  FAdresss: Mumbai  Password:\*\*\*\*\*\*  2)Enter Submit | <invalid phone number> | Phone number should be of 10 digits | Phone number should be 10 digits |
| TC\_o3 | User registration | Enter all the required fields to get registered | User must enter all the required field as per specified in schema | 1) Enter  UserId: F001  FFirstName: John  FLastName: Doe  FEmail: User@gmail.com  FContactNo: 0123456789  FAdresss: Delhi  Password:\*\*\*\*\*\*  2) Enter Submit | <All the Required fields are available> | Successful registration | Successful registration |
| TC\_o4 | User registration | Enter all the required fields to get registered | If User misses one of the fields during registration which is marked as required in schema | 1) Enter  UserId: F001  FFirstName: John  FLastName: Doe  FEmail:  FContactNo: 0123456789  FAdresss: Delhi  Password:\*\*\*\*\*\*  2) Enter Submit | <email is missing> | You need to enter email | You need to enter email |

**For Users listing**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test CASE ID** | **TEST CASE SCENARIO** | **TEST CASE** | **PRE-CONDITION** | **TEST STEPS** | **TEST DATA** | **EXPECTED RESULT** | **ACTUAL RESULTS** |
| TC\_o1 | User List | enter the valid User Id of User in parameter to get details | That id needs to be present in database | 1) Enters Valid Id  2) Enter Submit | Valid Id | User Details | User Details |
| TC\_o2 | User List | enters the wrong User Id which is not there in database to get User details | That id needs to be present in database | 1) Enters invalid Id  2) Enter Submit | Invalid Id | User Details with this id is not present | User Details with this id is not present |

**For Users Deletion**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test CASE ID** | **TEST CASE SCENARIO** | **TEST CASE** | **PRE-CONDITION** | **TEST STEPS** | **TEST DATA** | **EXPECTED RESULT** | **ACTUAL RESULT** |
| TC\_o1 | User Deletion | enter the valid id of User in parameter to get details | That id needs to be present in database | 1) Enters Valid Id  2) Enter Submit | Valid Id | User Deleted successfully | User Deleted successfully |
| TC\_o2 | User deletion | enters the wrong id which is not there in database to get User details | That id needs to be present in database | 1) Enters invalid Id  2) Enter Submit | Invalid Id | User Details with this id you want to delete is not present | User Details with this id you want to delete is not present |

**For User Updation**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test CASE ID** | **TEST CASE SCENARIO** | **TEST CASE** | **PRE-CONDITION** | **TEST STEPS** | **TEST DATA** | **EXPECTED RESULT** | **ACTUAL RESULT** | **STATUS(PASS/FAIL)** |
| TC\_o1 | User Updation | enter the valid id of User in parameter to get details | That id needs to be present in database | 1) Enters Valid Id  2) Enter Submit | Valid Id | User Update  Successfully | Update  Successfully | pass |
| TC\_o2 | User Updation | enters the wrong id which is not there in database to get User details | That id needs to be present in database | 1) Enters invalid Id  2) Enter Submit | Invalid Id | User Details with this  User Id you want to update is not present | User Details with this id you want to update is not present | fail |

# Request

**Register User**

/Users

{

“UserId”: “F001”,

“FFirstName”: “John”,

“FLastName”: ”Doe”,

“FEmail”: ” john@gmail.com”

“FContactNo”: ” 0123456789

“FAdresss”: ” Delhi”

“Password”: ” \*\*\*\*\*\*

}

**Update User**

/Users/

{

“UserId”: “F001”,

“FFirstName”: “Rohan”,

“FLastName”: ”Sharma”,

“FEmail”: ” rohan@gmail.com”

“FContactNo”: ” 0123456789

“FAdresss”: ” Delhi”

“Password”: ” \*\*\*\*\*\*

}

1. Response

**Register User**

**If valid details**

{

"message": "User registered successfully"

}

status code: 201

**If invalid details**

{

“message”: “Inputs are not valid”

}, status code: 400

**If server encounters unexpected error**

{

“message” :” Internal server error ”

}, status code: 500

**List Users**

**If valid ID**

{

“UserId”: “F001”,

“FFirstName”: “Rohan”,

“FLastName”: ”Sharma”,

“FEmail”: ” rohan@gmail.com”

“FContactNo”: ” 0123456789

“FAdresss”: ” Delhi”

“Password”: ” \*\*\*\*\*\*

}

status code: 200

**If invalid ID**

{

“message”: “User not found”

}, status code:404

**If server encounters unexpected error**

{

“message” :” Internal server error ”

}, status code: 500

**Update User**

**If valid ID**

{

“User updated successfully”

}

status code: 200

**If invalid ID**

{

“message”: “User not found”

}, status code:404

**If server encounters unexpected error**

{

“message” :” Internal server error”

}, status code: 500

**Delete User**

**If valid ID**

{

“User Deleted successfully”

}

status code: 200

**If invalid ID**

{

“message”: “User not found”

}, status code:404

**If server encounters unexpected error**

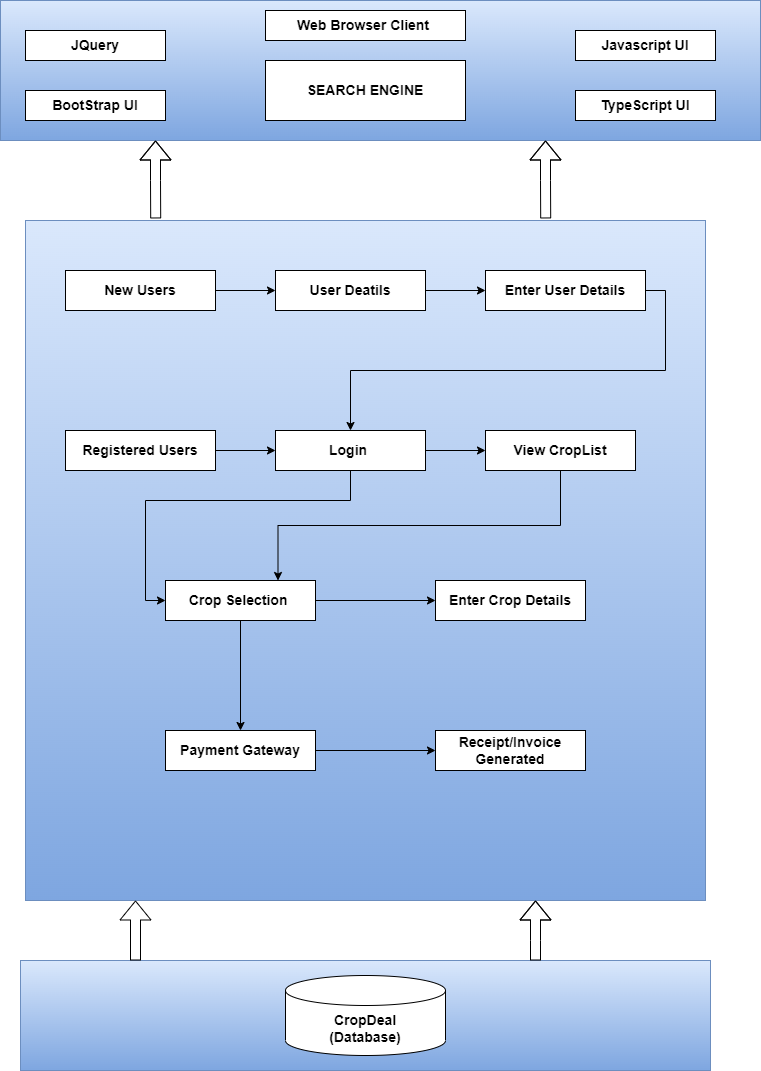
{

“message” :” Internal server error ”

}, status code: 500

# UML Diagram

## **Architecture Diagram**

 Figure 15.1 (Architecture Diagram

## Use Case Diagram

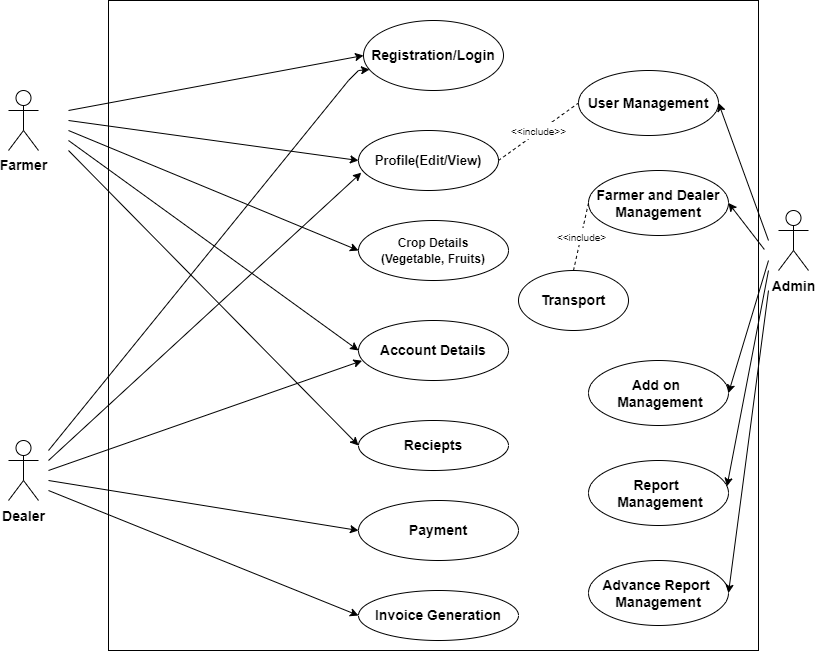


Figure 15.2 (Use Case Diagram)

## ER Diagram

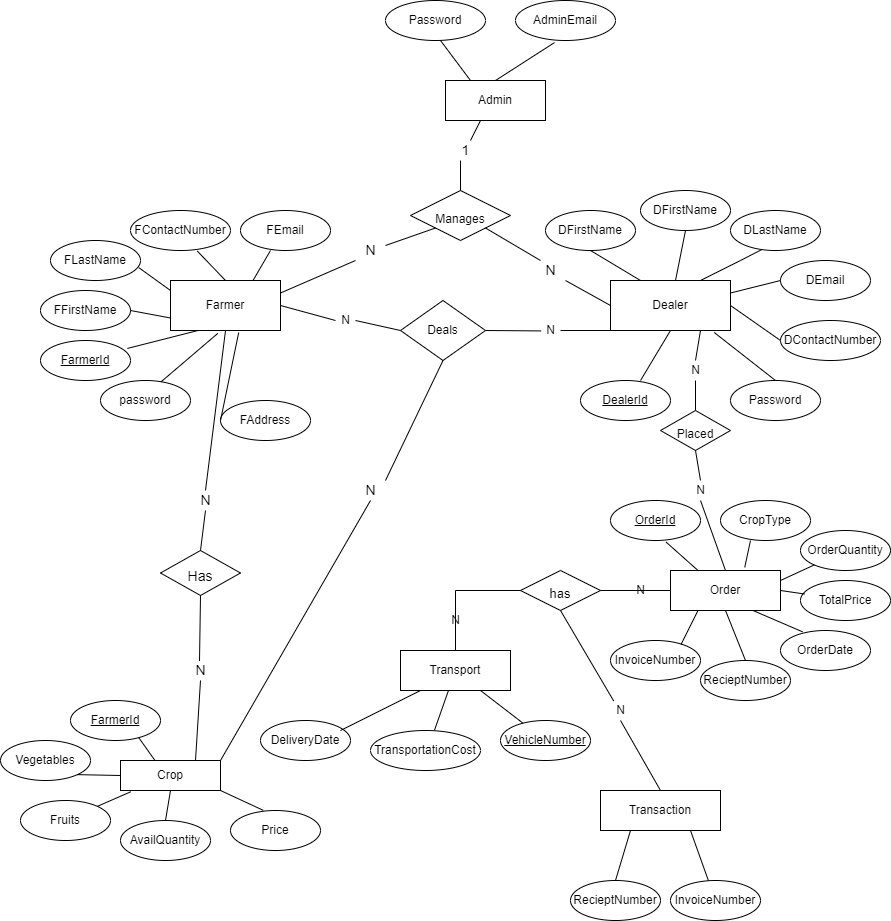


Figure 15.3 (ER Diagram)

## Sequence Diagram

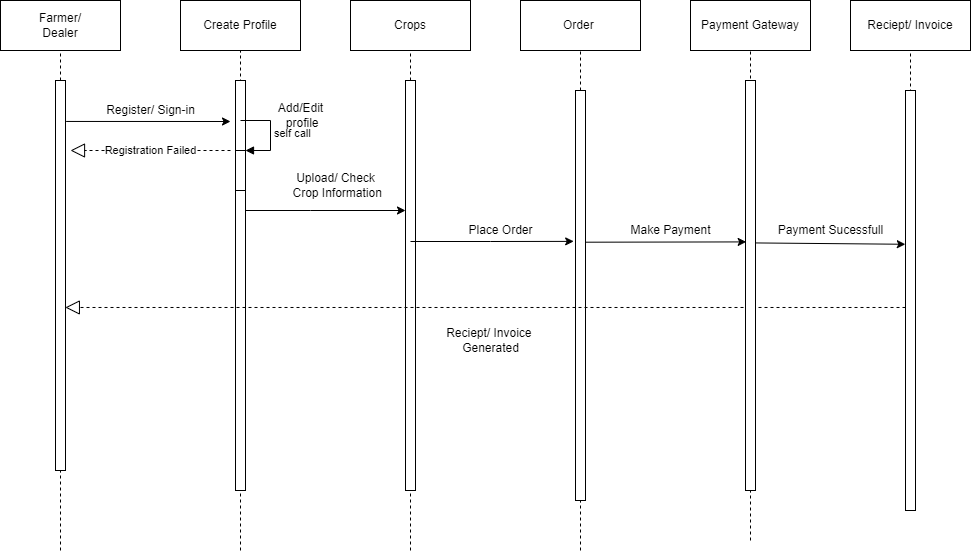


Figure 15.4(a) (Sequence Diagram)

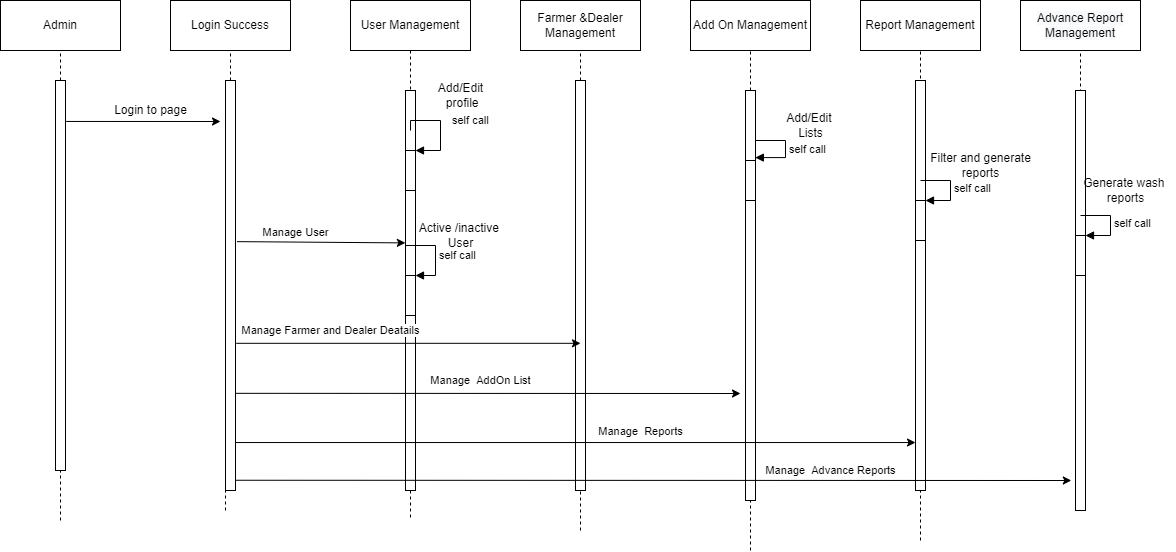


Figure 15.4(b) (Sequence Diagram)