1. INTRODUCTION

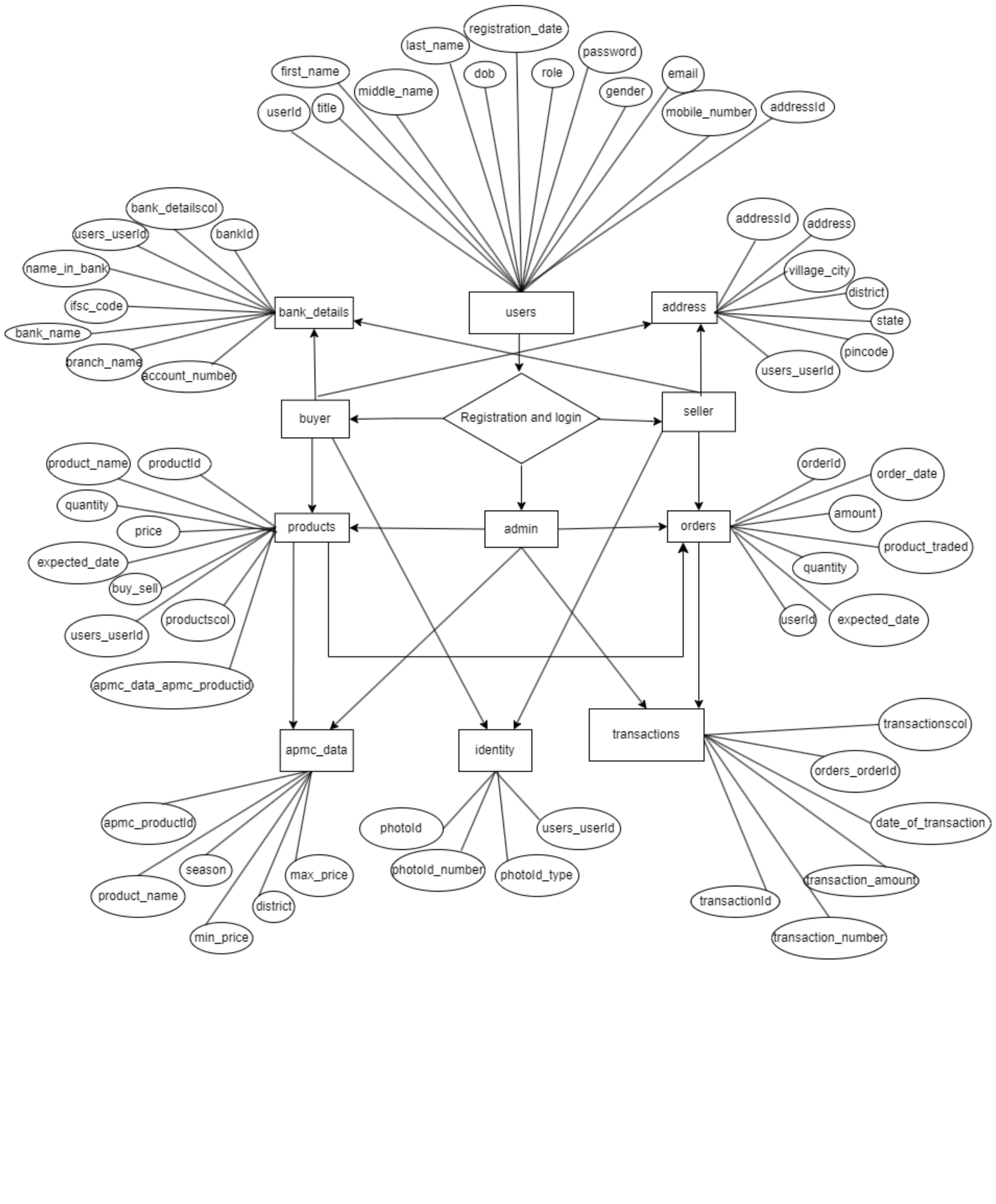
The web-based application titled “Agricultural Products Trading Platform” is an attempt to automate the trading of agricultural products across the country. The platform allows sellers to publish their agricultural products and can be found by genuine and bulk buyers across the country. The platform removes the hurdle of taking the produces to the market, or searching for buyers in the same village and allows sellers to sell their products across India, where he/she can expect the best prices and also removes the presence of middlemen who charges a hefty amount.

The platform also displays the current weather in the same area where the seller belongs to. Using this feature, seller can decide what to grow next. The platform also provides current market prices of their produces by comparing with APMC data. If the seller is not getting what he deserves, he/she can always opt for APMC.

The user(seller/buyer) first needs to create a free account on the platform. After selecting his appropriate role, the functionalities change accordingly. If the new user created is a seller, he/she can list his products, search for buyer. If the new user is a buyer, he/she can search for the required product, and if the product is not available, he/she can always put his/her requirements on the platform. And if the new user is Admin, he/she can inspect all the orders executed and their respective transactions on the platform, he/she can also see all the registered users and can remove fake profiles.

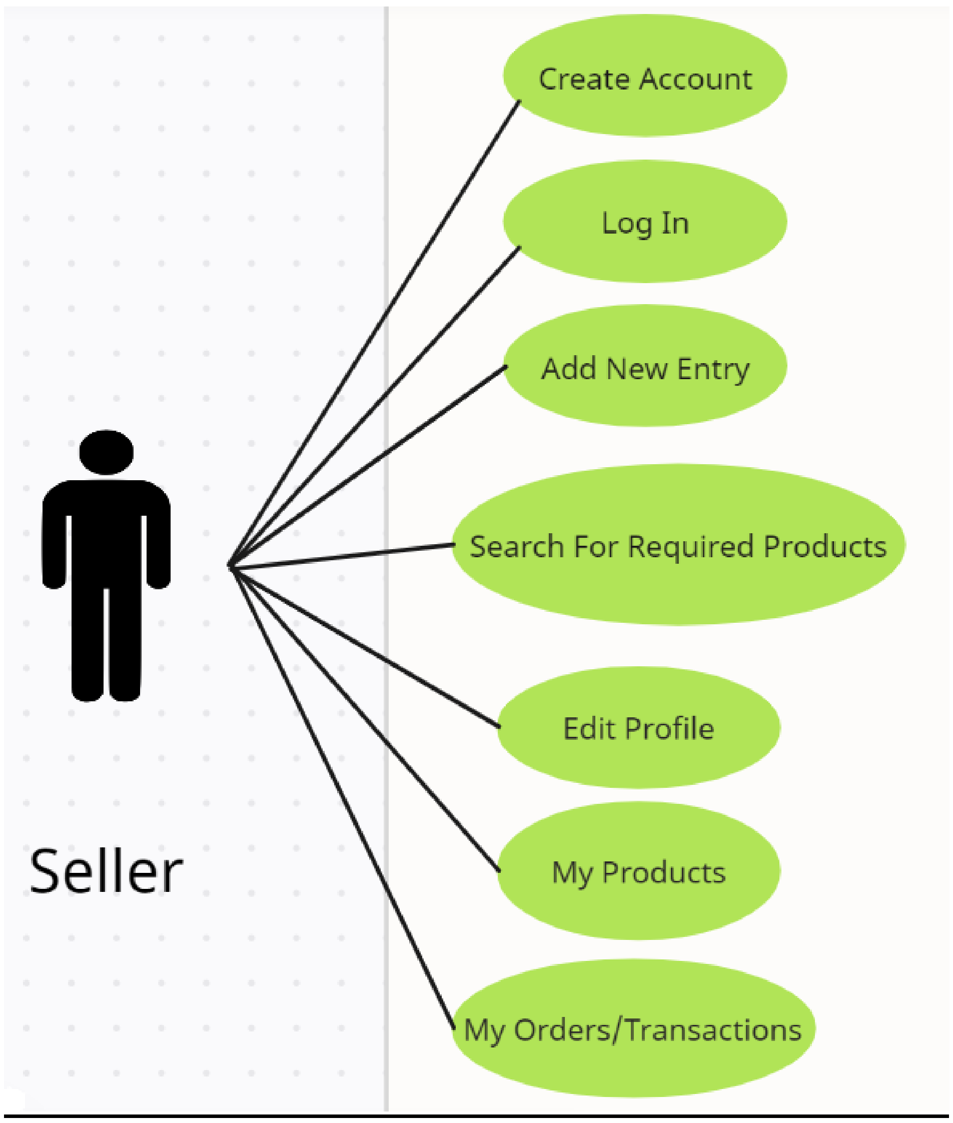
2. FUNCTIONAL REQUIREMENTS

- **CLASS-FLOW DIAGRAM:**



The person who visits the platform for the first time will be henceforth called as 'user', will be redirected to registration form, where he/she can register as either seller/buyer.

2.1: SELLER:



2.1.1: CREATE ACCOUNT:

- The user who selects 'Seller' as the role will be able to register himself/herself as seller.
- On this page, he/she can register on the platform. Registration is must to login and use the other functionalities.
- Every new user will be assigned a unique 'User-ID' on successful completion of the registration form.

- The seller has to select the appropriate title.
- The seller has to enter his/her first-name, middle-name and last-name.

- The seller has to enter his/her date of birth.
- The registration-date will be automatically added to the database.
- The role field will decide the role of the user. Here the role is seller.
- The seller has to enter a new password. Seller has to confirm the password.
- The seller has to select the gender.
- The seller has to enter his/her e-mail. E-mail will be used for login purpose in the future.
- The seller has to enter his/her mobile number for communication.
- The seller has to submit any government approved identity for verification. The id can be Aadhar-card, PAN-card. The seller has to select his/her id-type, id-number, etc.
- The seller has to enter his full address including village, district, state and pin-code. This address will help buyers to identify their nearest sellers.
- The seller has to enter his/her bank details like account number, IFSC, bank name, etc.

2.1.2: LOG IN:

- The seller can now log in to the platform using the credentials.
- The username field will require to enter the e-mail id, as e-mail id is the primary login credential.
- The seller has to enter the same password that was entered while registering.
- After clicking on 'Login' button, the credentials will be verified from the database and the seller will be able to login to his/her account, if the credentials are valid.
- The seller will be redirected to the dashboard, where he/she can see and use other functionalities.

2.1.3: ADD NEW ENTRY:

- On this page, the seller can enter his/her product details that he/she wishes to sell.
- This functionality requires seller to enter various details like product name, the quantity available for selling, the expected price, and the date from which the product is available for buying.
- After successful completion, every product will be added to the database and will be assigned a unique 'Product-ID' and the 'User-ID' of the seller.
- Now, this product is live on the platform for selling.

2.1.4: SEARCH FOR REQUIRED PRODUCT:

- The seller can search for any pending request on the platform. On this page, seller will be able to search for any requirements, that a buyer has added.
- The seller then can directly contact the buyer, if he/she has the matching required product and matches with his/her expectations.

2.1.5: EDIT PROFILE:

- On his page, seller will be able to change his/her basic details like name, contact details, bank details, etc.
- The seller can update all the required fields except the primary e-mail id.

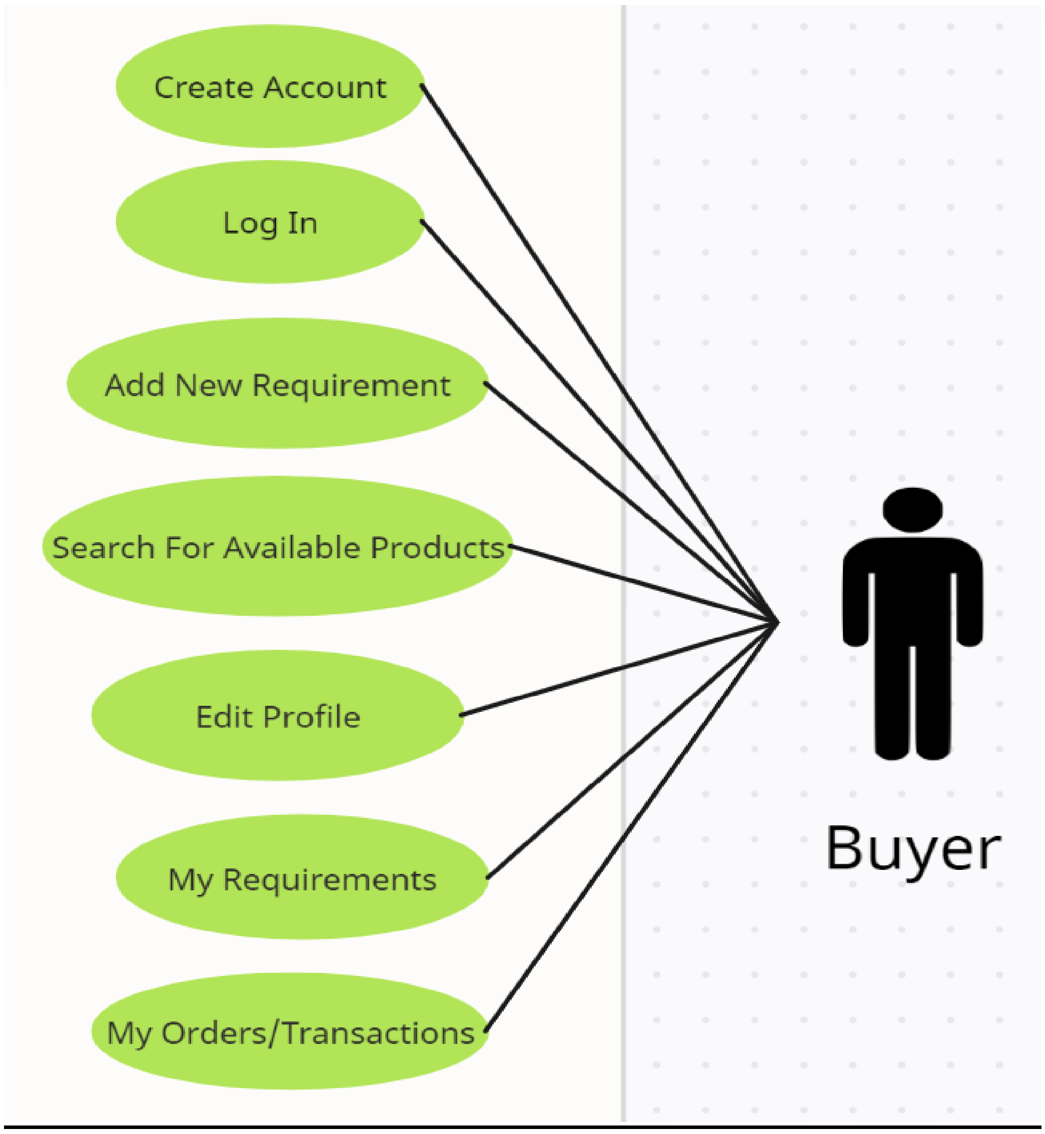
2.1.6: MY PRODUCTS:

- On this page, seller can check his/her listed products. All the products that are linked to a particular 'User-ID' will be displayed here.
- The seller can update the product details, if required.

2.1.7: MY ORDERS/TRANSACTIONS:

- On this page, seller will be able to review his/her previous orders. If there occurs a deal between buyer and seller, then only the particular executed order will be displayed here.
- Seller can check the order date, amount, the product traded, quantity of the product traded.
- Also, seller can check the transaction details for that particular order on the same page.

2.2: BUYER:



2.2.1: CREATE ACCOUNT:

- The user who selects 'Buyer' as the role will be able to register himself/herself as buyer.
- On this page, he/she can register on the platform. Registration is must to login and use the other functionalities.
- Every new user will be assigned a unique 'User-ID' on successful completion of the registration form.
- The buyer has to select the appropriate title.
- The buyer has to enter his/her first-name, middle-name and last-name.
- The buyer has to enter his/her date of birth.
- The registration-date will be automatically added to the database.
- The role field will decide the role of the user. Here the role is buyer.
- The buyer has to enter a new password. Seller has to confirm the password.
- The buyer has to select the gender.
- The buyer has to enter his/her e-mail. E-mail will be used for login purpose in the future.
- The buyer has to enter his/her mobile number for communication.
- The buyer has to submit any government approved identity for verification. The id can be Aadhar-card, PAN-card. The buyer has to select his/her id-type, id-number, etc.
- The buyer has to enter his full address including village, district, state and pin-code. This address will help sellers to identify their nearest buyers.
- The buyer has to enter his/her bank details like account number, IFSC, bank name, etc.

2.2.2: LOG IN:

- The buyer can now log in to the platform using the credentials.
- The username field will require to enter the e-mail id, as e-mail id is the primary login credential.
- The buyer has to enter the same password that was entered while registering.
- After clicking on 'Login' button, the credentials will be verified from the database and the buyer will be able to login to his/her account, if the credentials are valid.
- The buyer will be redirected to the dashboard, where he/she can see and use other functionalities.

2.2.3: ADD NEW REQUIREMENT:

- On this page, the buyer can enter his/her product details that he/she wishes to buy.
- This functionality requires buyer to enter various details like product name, the quantity required, the expected price, and the date on which the product is required.
- After successful completion, every requirement will be added to the database and will be assigned a unique 'Product-ID' and the 'User-ID' of the seller.
- Now, this requirement is live on the platform for sellers to see.

2.2.4: SEARCH FOR AVAILABLE PRODUCT:

- The buyer can search for required product on the platform. On this page, buyer will be able to search for any products, that a seller has added.

- The buyer then can directly contact the seller, if he/she has the required product and matches with his/her expectations.

2.2.5: EDIT PROFILE:

- On his page, buyer will be able to change his/her basic details like name, contact details, bank details, etc.
- The buyer can update all the required fields except the primary e-mail id.

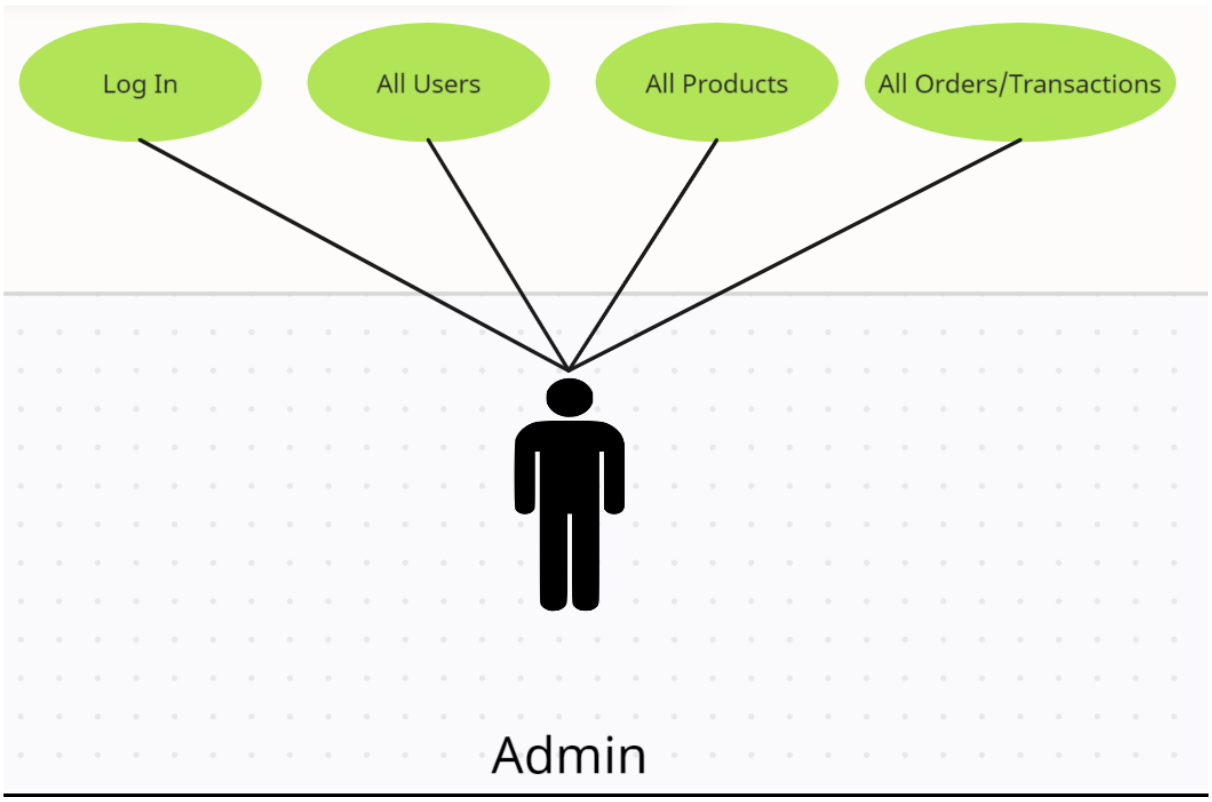
2.2.6: MY REQUIREMENTS:

- On this page, buyer can check his/her listed requirements. All the requirements that are linked to a particular 'User-ID' will be displayed here.
- The buyer can update the requirement, if required.

2.2.7: MY ORDERS/TRANSACTIONS:

- On this page, buyer will be able to review his/her previous orders. If there occurs a deal between buyer and seller, then only the particular executed order will be displayed here.
- Buyer can check the order date, amount, the product traded, quantity of the product traded.
- Also, buyer can check the transaction details for that particular order on the same page.

2.3: ADMIN:



2.3.1: LOG IN:

- The admin can now log in to the platform using the credentials.
- The admin credentials will be added to the database directly.
- The admin has to enter the same e-mail id and password that was entered in the database.
- After clicking on ‘Login’ button, the credentials will be verified from the database and the admin will be able to login to his/her account, if the credentials are valid.
- The admin will be redirected to the dashboard, where he/she can see and use other functionalities.

2.3.2: ALL USERS:

- The admin can check all the registered users on the platform. Admin can inspect all the details of sellers, buyers. Admin has a filter to categorise users on the basis of their roles viz. buyer, seller, admin.
- Admin can delete any suspicious or inactive user from the database.

2.3.3: ALL PRODUCTS:

- On this page, admin can check all the products listed by sellers.
- Also, admin can check all the pending requirements from the buyers.

2.3.4: ALL ORDERS/TRANSACTIONS:

- On this page, admin will be able to review all the previous orders. If there occurs a deal between a buyer and a seller, then only the particular executed order will be displayed here.
- Admin can check the order date, amount, the product traded, quantity of the product traded.
- Also, admin can check the transaction details for that particular order on the same page.

3. PROJECT DESIGN

The project design has been divided into 3 stages, viz.

1) Database design.
2) Back-end development.
3) Front-end development.
4) Security.
5) Integration.
6) Deployment.

3.1: DATABASE DESIGN:

3.1.1: TABLE STRUCTURE:

The following table structure depict the database design.

- Table 1: Users

FIELD	TYPE	LENGTH	KEY	DEFAULT	EXTRA
User_id	bigint	-	PRI	NULL	auto_increment
Email	varchar	40	-	NULL	-
Title	varchar	10	-	NULL	-
First Name	varchar	20	-	NULL	-
Middle Name	varchar	20	-	NULL	-
Last Name	varchar	20	-	NULL	-
DOB	date	-	-	NULL	-
Gender	varchar	20	-	NULL	-
Password	varchar	255	-	NULL	-
Mobile Number	varchar	14	-	NULL	-
Registration Date	date	-	-	NULL	-
Role	varchar	25	-	NULL	-
Address_id	bigint	-	MUL	NULL	-
Bank_Details_id	bigint	-	MUL	NULL	-
Identity_id	bigint	-	MUL	NULL	-

- Table 2: Address

FIELD	TYPE	LENGTH	KEY	DEFAULT	EXTRA
Address_id	bigint	-	PRI	NULL	auto_increment
Address	varchar	150	-	NULL	-
Village	varchar	30	-	NULL	-
District	varchar	30	-	NULL	-
Pincode	varchar	12	-	NULL	-
State	varchar	30	-	NULL	-

- Table 3: Bank_details

FIELD	TYPE	LENGTH	KEY	DEFAULT	EXTRA
Bank_details_id	bigint	-	PRI	NULL	auto_increment
Account Holder Name	varchar	45	-	NULL	-
Account Number	varchar	30	UNI	NULL	-
Bank Name	varchar	40	-	NULL	-
Branch	varchar	22	-	NULL	-
IFSC	varchar	30	-	NULL	-

- Table 4: Identity

FIELD	TYPE	LENGTH	KEY	DEFAULT	EXTRA
Identity_id	bigint	-	PRI	NULL	auto_increment
Identity Type	varchar	20	-	NULL	-
Identity Number	varchar	30	-	NULL	-

- Table 5: APMC Data

FIELD	TYPE	LENGTH	KEY	DEFAULT	EXTRA
Id	bigint	-	PRI	NULL	auto_increment
District	varchar	30	-	NULL	-
Maximum Price	double	-	-	NULL	-
Minimum Price	double	-	-	NULL	-
Product	varchar	30	UNI	NULL	-
Season	varchar	20	-	NULL	-

- Table 6: Products

FIELD	TYPE	LENGTH	KEY	DEFAULT	EXTRA
Product_id	bigint	-	PRI	NULL	auto_increment
Buy_Or_Sell	varchar	15	-	NULL	-
Expected_Date	date	-	-	NULL	-
Price	double	-	-	NULL	-
Product_Name	varchar	60	-	NULL	-
Quantity	double	-	-	NULL	-
User_Id	bigint	-	MUL	NULL	-

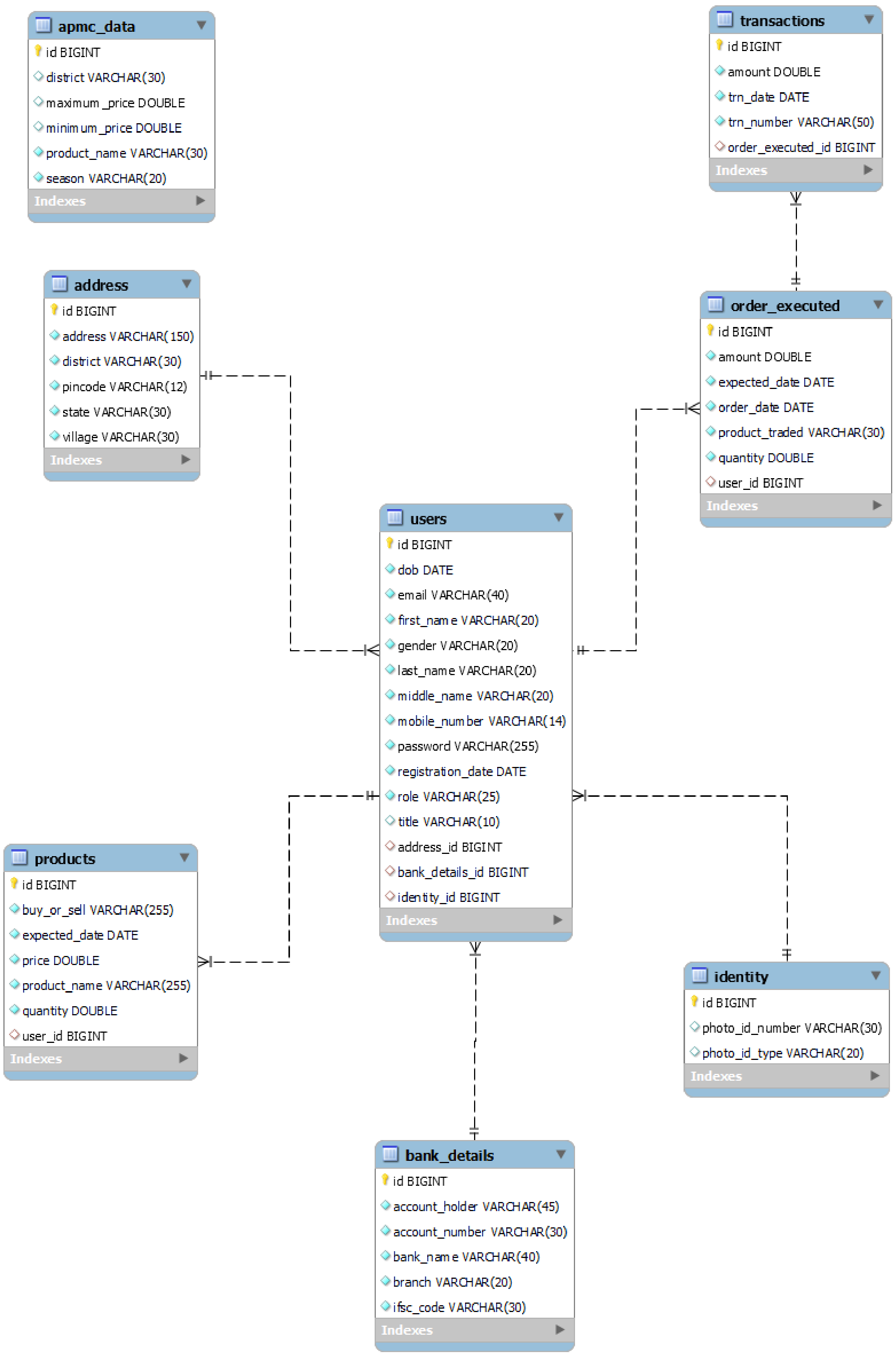
- Table 7: Orders Executed

FIELD	TYPE	LENGTH	KEY	DEFAULT	EXTRA
Order_Executed_Id	bigint	-	PRI	NULL	auto_increment
Amount	double	-	-	NULL	-
Expected Date	date	-	-	NULL	-
Order Date	date	-	-	NULL	-
Product Traded	varchar	30	-	NULL	-
Quantity	double	-	-	NULL	-
User_Id	bigint	-	MUL	NULL	-

- Table 8: Transactions

FIELD	TYPE	LENGTH	KEY	DEFAULT	EXTRA
Transaction_Id	bigint	-	PRI	NULL	auto_increment
Amount	double	-	-	NULL	-
Transaction Date	date	-	-	NULL	-
Transaction Number	varchar	50	-	NULL	-
Order_Executed_ID	bigint	-	MUL	NULL	-

3.1.2: E-R DIAGRAM:



3.2: BACK-END DEVELOPMENT:

‘J2EE’ i.e ‘Jave Enterprise Edition’ has been used to implement the back-end i.e. server-side programming.

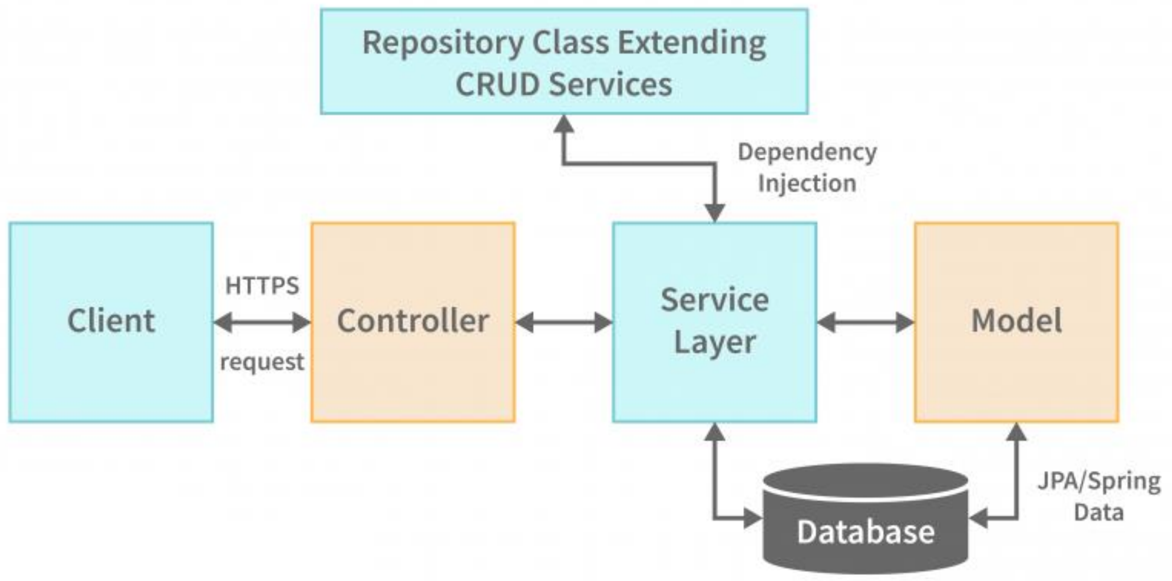
Spring-Boot framework is the core part for the back-end development. All the implementation has been done on ‘STS-IDE’, which is a ‘Eclipse-based’ IDE that has in-built plugins for Spring-Boot Application development.

Spring Boot is an open-source Java-based framework that is designed to simplify the development of web applications. It provides a set of tools and features that enable developers to quickly create and deploy production-ready applications with minimal configuration. Spring Boot builds on top of the Spring framework.

Spring Boot provides several benefits for back-end development. Firstly, it simplifies the development process by reducing the amount of boilerplate code required for configuring the application. Secondly, it provides a wide range of pre-built components and modules, such as RESTful APIs, database connectivity, security features, in-built ‘Apache Tomcat Server’ and more, which developers can easily integrate into their applications. Thirdly, it allows developers to package and deploy their applications as standalone executable JAR files, making deployment and maintenance easier.

Using Spring Boot for back-end development enables developers to focus on the core functionality of their applications rather than spending time on infrastructure and configuration. This results in faster development cycles and more robust, maintainable code. Additionally, Spring Boot has a large and active community of developers who contribute to the framework, providing regular updates and support.

Following is the basic architecture of a Spring-Boot project:



3.2.1: CLIENT:

- Client is a person who wishes to use the functionalities of the application. The client uses a ‘URL’ specified by the application developer.
- When the client enter this url in the browser, he/she sends an ‘HTTP-Request’ to the server of the application. This http request can be stateless or statefull.

3.2.2: CONTROLLER:

- Controller is a special type of class that receives the request from the client.
- When any type of http request is made by the client, it first goes to the controller class. ‘Home-Controller’ is the class that receives the request.
- On the basis of http request, the controller class decides where to forward the request. E.g. if the request received contains functionalities of ‘Admin’, the home-controller will forward the request to the ‘Admin-Controller’
- The code-snippet for the ‘Home-Controller’ looks like follow:

```
package com.example.demo.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.pojos.User;
import com.example.demo.service.UserService;

@RestController
@RequestMapping("/")
public class HomeController {

        @Autowired
        UserService userservice;

        @PostMapping("/register")
        public String createUser(@RequestBody User user)
        {
                return userservice.registerUser(user);
        }
}
```

3.2.3: SERVICE LAYER:

- Service layer contains the actual business logic of the project. In service layer, all the logic and methods are declared and defined.
- Depending on the request received from the controller, the appropriate method inside the class gets called.
- The service layer is further divided into 2 categories:

  1) Interfaces: The interfaces contain the declaration of the methods.
  2) Classes: The classes contain the actual definition of the methods. These classes implement the particular interfaces.

  This technique is used to achieve loose-coupling i.e. all the code should be independent of each other.

- Hence, all the required business logic could be found here. If there is a need of data from the database, the service layer communicates with the 'Model'
- The code-snippet for the service layer looks like follow:
- UserService Interface:

```
package com.example.demo.service;

import java.util.List;

import com.example.demo.pojos.User;

public interface UserService {

        public String registerUser(User user);

        public User authenticateUser(String email,String password);

        public String deleteUser(long id);

        public String updateUser(long id,User newUser);

        public List<User> getAllUser();

}
```

- UserServiceImpl Class:

```
package com.example.demo.service;

import java.util.List;
import javax.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.demo.dao.UserDao;
import com.example.demo.pojos.User;

@Transactional
@Service
public class UserServiceImpl implements UserService {

	@Autowired
	UserDao userDao;

	@Override
	public String registerUser(User user) {
		userDao.save(user);
		return "User added successfully";
	}

	@Override
	public User authenticateUser(String email,String password) {
		User user= userDao.AuthenticateUser(email, password);
		return user;
	}

	@Override
	public String deleteUser(long id) {
		userDao.deleteById(id);
		return "User deleted Successfully";
	}

	@Override
	public String updateUser(long id,User newUser) {
		User oldUser=userDao.findById(id).orElseThrow(()->new RuntimeException("Id not found"));
		oldUser.setDob(newUser.getDob());
		oldUser.setEmail(newUser.getEmail());
		oldUser.setFirstName(newUser.getFirstName());
		oldUser.setGender(newUser.getGender());
		oldUser.setLastName(newUser.getLastName());
		oldUser.setMiddleName(newUser.getMiddleName());
		oldUser.setMobileNumber(newUser.getMobileNumber());
		oldUser.setPassword(newUser.getPassword());
		userDao.save(oldUser);
		return "user Updates Successfully";
	}

	@Override
	public List<User> getAllUser() {
		 List<User>allUsers=userDao.findAll();
		return allUsers;
	}


}
```

3.2.4: MODEL:

- Model, also called as DAO(Data Access Object) are the classes that actually communicates with the database.
- If service layer requests for data, then DAO communicates with the database and returns the same.
- DAO layer also contributes in adding data to the database.

3.2: FRONT-END DEVELOPMENT:

React-JS library has been used for development of the front-end i.e. client-side programming.

ReactJS is a popular JavaScript library that is primarily used for building user interfaces (UIs) for web applications. It was developed by Facebook and has gained significant popularity in recent years due to its simplicity, flexibility, and ability to efficiently handle large and complex applications.

ReactJS works by creating reusable components that can be used to build a UI. These components are small, modular pieces of code that represent different parts of the UI, such as a button, input field, or image. Components can be nested within each other to create more complex UI elements.

One of the key features of ReactJS is its use of a virtual DOM (Document Object Model), which is a lightweight representation of the actual DOM. When a user interacts with a React-based application, the virtual DOM is updated, and React compares the virtual DOM with the actual DOM to determine which parts of the UI need to be updated. This makes React very fast and efficient, as it only updates the parts of the UI that need to be changed, rather than re-rendering the entire UI.

ReactJS is also highly customizable and can be used with other libraries and frameworks, making it a versatile choice for building web applications. It is often used in combination with other front-end technologies, such as Redux for state management, React Router for routing, and Axios for handling HTTP requests.

Request forwarding flow:

- When a client enters the url in the browser, he gets redirected to the homepage of the application.
- The first page that client see is the ‘index.html’.
- Index.html is the first page that loads in the browser. In this page, there is a single ‘<div></div>’.
- This ‘div’ receives data from index.js, which is a javascript file, which has ‘javascript and html’ code.
- This index.js file has a function which returns a component called ‘App’.
- Now, this component is written in another ‘App.js’ file. You need to import the dependent files in your depending files.

- Now, this ‘App,js’ has all the code that renders the homepage. In this file, there is a function that returns all the components that user wants to display on the webpage.

- Thus, using this technique, we can achieve modularity and loose-coupling.
- All these pages have their own ‘.css’ file, which provides styling to the webpage.
- Following is the code-snippet that describes a basic ‘index.js’ file:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

3.4: SECURITY:

In this project, Spring-Security has been used to implement the security.

Spring Security is a powerful and highly customizable framework for managing security in Java applications. It is built on top of the Spring Framework and provides a variety of features for securing web applications, RESTful services, and microservices.

The primary goal of Spring Security is to provide a flexible and robust security architecture that can be easily integrated into existing Spring-based applications. It includes a number of features to help developers implement authentication, authorization, and other security-related functionality.

One of the key features of Spring Security is its support for a wide range of authentication mechanisms, including username/password authentication, token-based authentication, and OAuth 2.0 authentication. Spring Security also provides support for multi-factor authentication and integration with third-party identity providers, such as Google and Facebook.

In addition to authentication, Spring Security also provides robust authorization functionality. Developers can easily define authorization rules using expressions, annotations, or custom security filters. This allows them to control access to specific resources or functionalities based on a user's role or other criteria.

Spring Security also includes features for handling common security-related tasks, such as password hashing, CSRF protection, and session management. It can also

integrate with other Spring frameworks, such as Spring Data and Spring MVC, to provide a comprehensive security solution for Java applications.

Overall, Spring Security is a powerful and flexible framework for managing security in Java applications. Its broad range of features and easy integration with other Spring frameworks make it a popular choice for developers looking to add robust security to their applications.

The Authorization Controller class code looks like following:

```
package com.example.demo.security;

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.pojos.Credentials;


@RestController
@CrossOrigin
public class AuthControler {

	@Autowired
	AuthenticationManager authenticationManager;

	@Autowired
	JwtUtils jwtUtils;

	@PostMapping("/authenticate")
	public ResponseEntity<String> authenticate(@RequestBody Credentials cred) {
		System.out.println("Authenticating: " + cred);
		try {
			Authentication auth = new UsernamePasswordAuthenticationToken(cred.getEmail(), cred.getPassword());
			auth = authenticationManager.authenticate(auth);
			User user = (User)auth.getPrincipal();

			String role = null;
			Collection<GrantedAuthority> authorities = user.getAuthorities();
			for (GrantedAuthority grantedAuthority : authorities) {
				role = grantedAuthority.getAuthority();
			}
			String token = jwtUtils.generateToken(user.getUsername(),role);
			return ResponseEntity.ok(token);
		}catch (BadCredentialsException e) {
			return ResponseEntity.notFound().build();
		}
	}

}
```

3.5: INTEGRATION:

Spring Boot and React can be integrated to build modern web applications with a scalable and efficient architecture.

Spring Boot is a popular Java-based framework used for building web applications. It provides many features, including auto-configuration, embedded server, and simplified dependency management, which makes it easy to set up and run a Spring application.

React, on the other hand, is a JavaScript library used for building user interfaces. It is a popular choice for front-end development due to its performance, flexibility, and ability to create reusable components.

To integrate Spring Boot and React, one approach is to build a RESTful API with Spring Boot that can be consumed by a React-based front-end. This can be achieved by using the Spring MVC framework to handle HTTP requests and responses and exposing the API endpoints as RESTful services.

On the React side, the UI can be built using React components that consume the API endpoints exposed by the Spring Boot application. React components can be organized into a component hierarchy and can interact with the API using Axios, a popular HTTP client library for JavaScript.

To make it easier to deploy and manage the integrated application, Spring Boot can be configured to serve the React-based UI as a static resource. This can be done by placing the React build files in the Spring Boot application's resources/static directory, allowing the React UI to be served by the Spring Boot embedded server.

Overall, integrating Spring Boot and React can result in a powerful and scalable architecture for building modern web applications. The combination of Spring Boot's robust back-end capabilities and React's flexible front-end components can help developers build applications that are easy to maintain, scalable, and efficient.

3.6: DEPLOYMNET:

4. TESTING:

The testing has been done by visiting each page, and checking whether any page generates any error while loading or if the user enters an inappropriate data.

Following table depicts all the results generated while testing.

SR. NO.	TEST CASE	EXPECTED RESULT	ACTUAL RESULT	MESSAGE
1	Home Page Visit	User should be able to see all the buttons, UI and page should load fast.	Same as expected.	-
2	Login	When user clicks on Login, he/she should get redirected to Login page, where he/she can login or register	Same as expected.	• Login successful!! (for valid credentials) • Invalid Credentials (for invalid credentials)
3	Register	User should be able to enter all the required fields. If user enters invalid details, or leaves any field empty, he/she should receive an error. On successful registration, he/she should get redirected to login page.	Same as expected.	• User Registered Successfully (if all details are valid) • Invalid details (if details entered are invalid, or empty field detected)
4	Weather	User should be able to check weather of the selected district.	Same as expected.	-
5	APMC	User should be able to check the APMC data.	Same as expected.	-
6	Add Product (Seller)	Seller should be able to enter his product details. On successful addition, user should be	Same as expected.	• Product added successful (if details are valid) • Invalid details (for

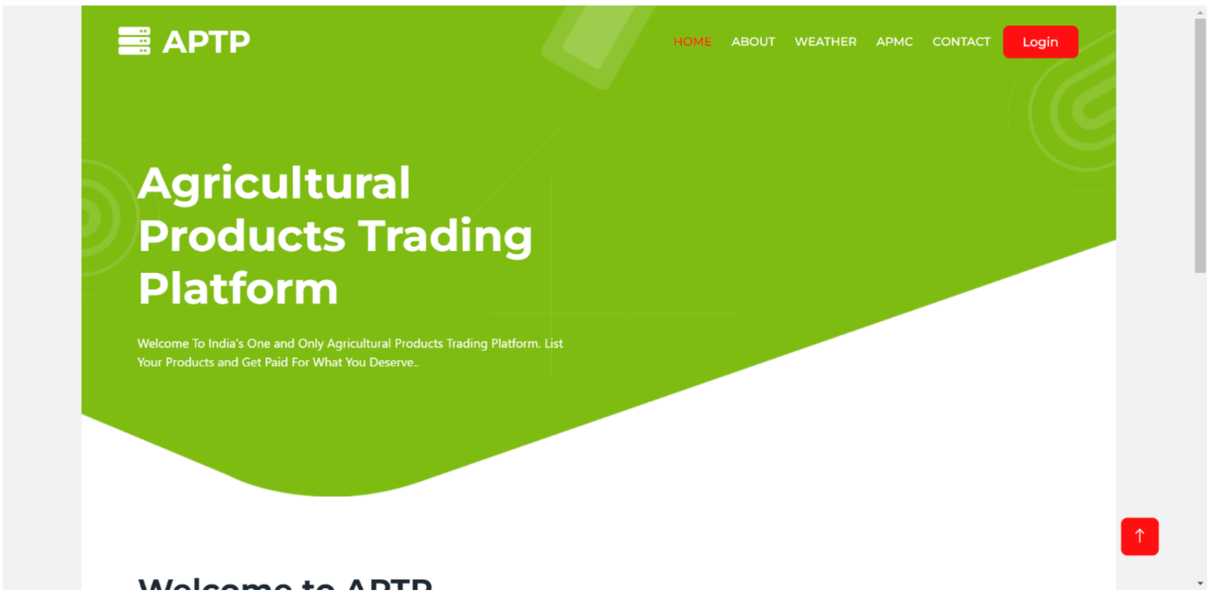
		redirected to ‘My Orders’. If invalid details are entered or field is left empty, user should receive an error.		invalid/empty details)
7	My Products (Seller)	Seller should be able to see all his/her pending products. Also, should be able to delete invalid product details.	Same as expected.	-
8	Search for buyer (Seller)	Seller should be able to see all the pending requests on the platform. If he/she has the required product, he/she can click on ‘sell’ button.	Same as expected.	-
9	My Orders/ Transactions (Seller)	Seller should be able to check his/her previous orders. Also, when he/she clicks on ‘Transaction’ button, should be able to check transaction details for that particular order.	Same as expected.	-
10	Edit Profile (Seller)	Seller should be able to edit profile and change basic details. Seller can not change email id.	Same as expected.	-
11	Add Requirement (Buyer)	Buyer should be able to enter his requirements. On successful addition, user should be redirected to ‘My Requirements’. If invalid details are entered or field is left empty, user should receive an error.	Same as expected.	• Requirement added successful (if details are valid) • Invalid details (for invalid/empty details)

12	My Requirements (Buyer)	Buyer should be able to see all his/her pending requirements. Also, should be able to delete invalid requirement details.	Same as expected.	-
13	Search for seller (Buyer)	Buyer should be able to see all the pending requests on the platform. If he/she has the required product, he/she can click on ‘buy’ button.	Same as expected.	-
14	My Orders/ Transactions (Buyer)	Buyer should be able to check his/her previous orders. Also, when he/she clicks on ‘Transaction’ button, should be able to check transaction details for that particular order.	Same as expected.	-
15	Edit Profile (Buyer)	Buyer should be able to edit profile and change basic details. Buyer can not change email id.	Same as expected.	-
16	All Users (Admin)	Admin should be able to check all users including buyers, sellers and admins. Admin can delete invalid or inactive user.	Same as expected.	-
17	All Orders/ Transactions (Admin)	Admin should be able to check all executed orders on the platform and can check the respective transaction.	Same as expected.	-
18	All Products (Admin)	Admin should be able to all the products as well as requirements on the platform.	Same as expected.	-

19	Edit Profile (Admin)	Admin should be able to edit his/her profile.	Same as expected.	-
20	Logout	User should be able to see ‘Log out’ button after successful login. On logout, he/she should get redirected to sign in page	Same as expected.	User Logged Out Successfully.

5. USER INTERFACE

- Home Page:

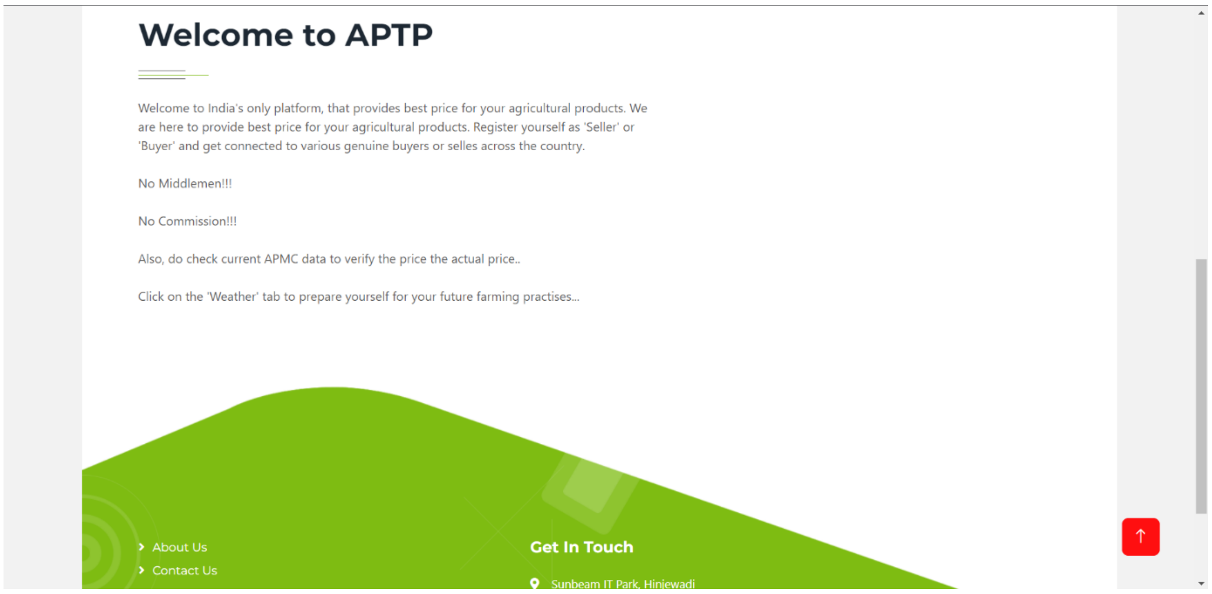


APTP

HOME ABOUT WEATHER APMC CONTACT Login

Agricultural Products Trading Platform

Welcome To India's One and Only Agricultural Products Trading Platform. List Your Products and Get Paid For What You Deserve..



Welcome to APTP

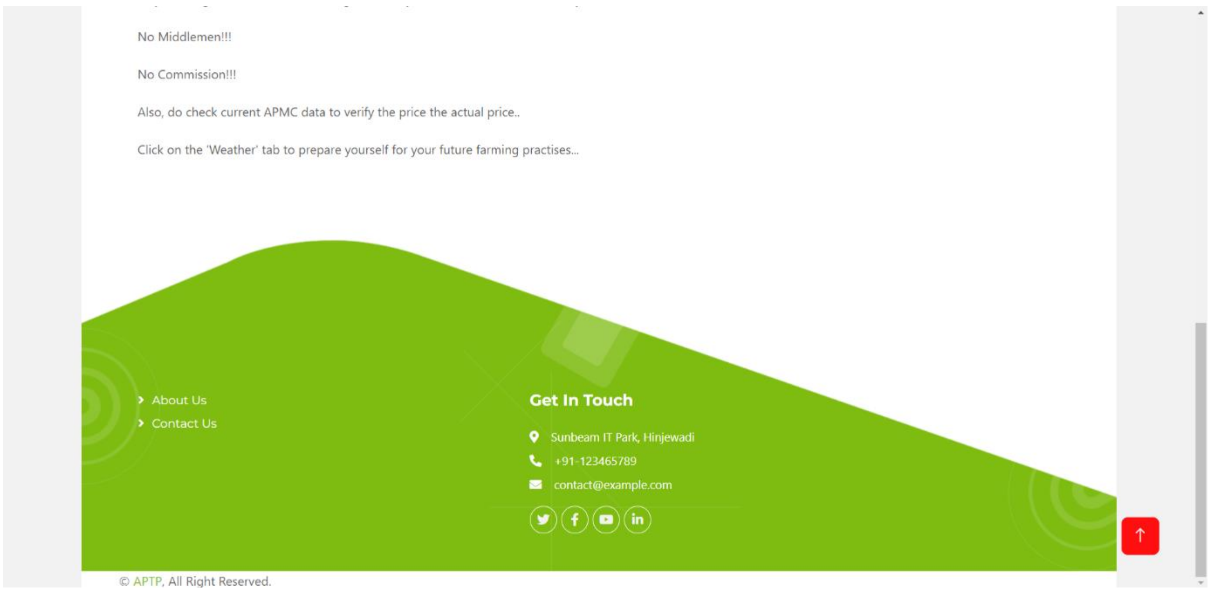
Welcome to India's only platform, that provides best price for your agricultural products. We are here to provide best price for your agricultural products. Register yourself as 'Seller' or 'Buyer' and get connected to various genuine buyers or selles across the country.

No Middlemen!!!

No Commission!!!

Also, do check current APMC data to verify the price the actual price..

Click on the 'Weather' tab to prepare yourself for your future farming practises...



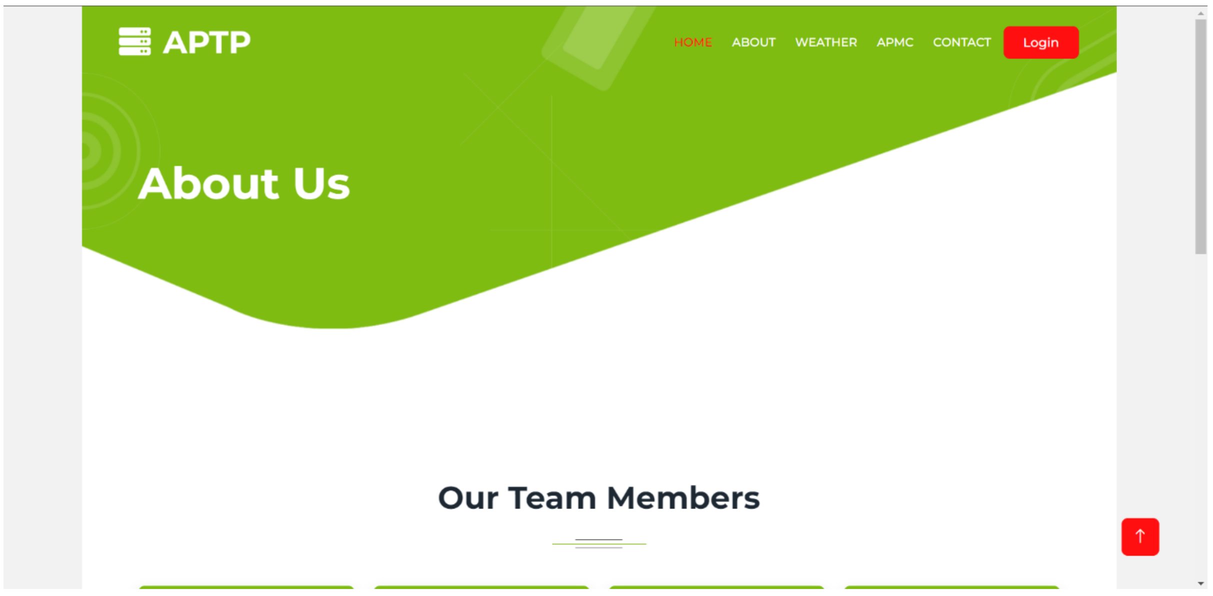
No Middlemen!!!

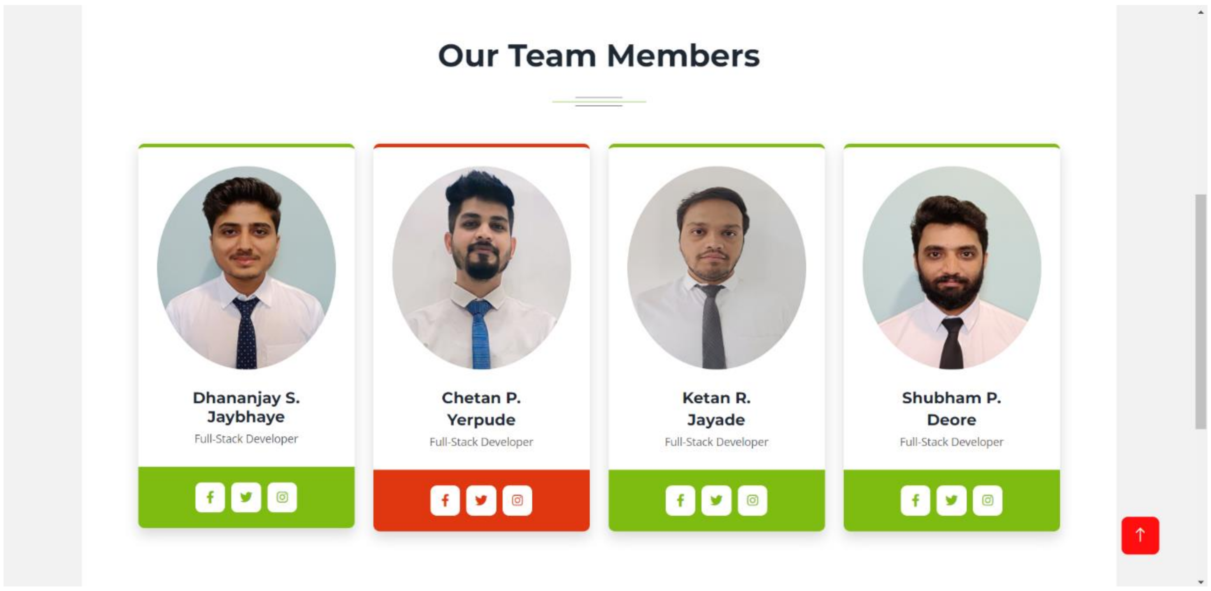
No Commission!!!

Also, do check current APMC data to verify the price the actual price..

Click on the 'Weather' tab to prepare yourself for your future farming practises...

- About Us Page:





Our Team Members

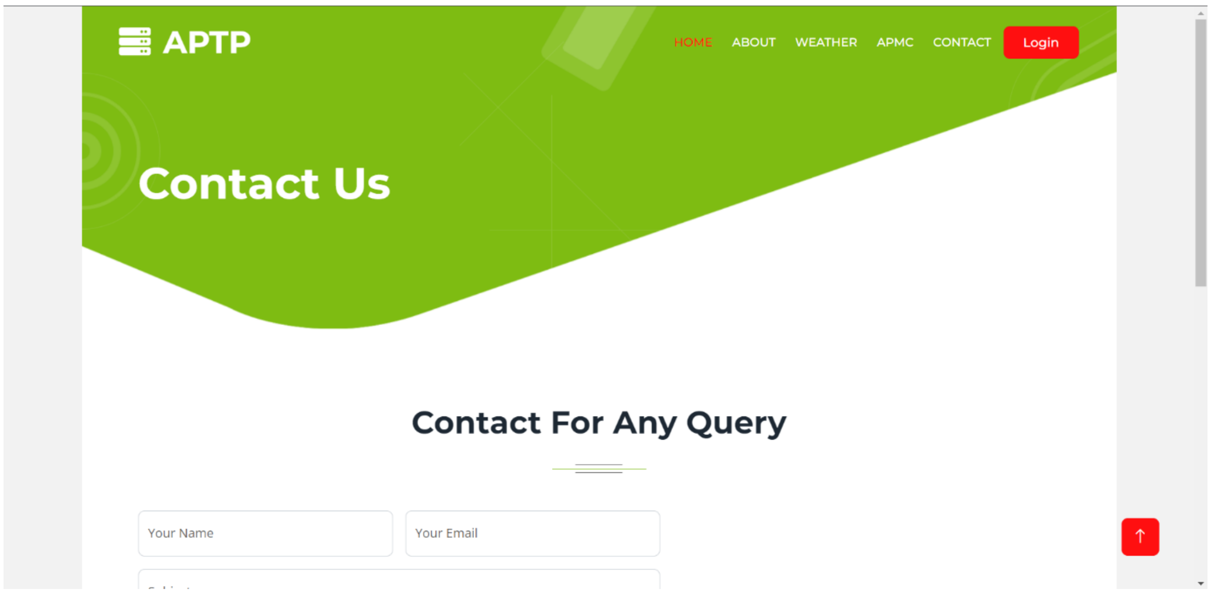
Dhananjay S. Jaybhaye
Full-Stack Developer

Chetan P. Yerpude
Full-Stack Developer

Ketan R. Jayade
Full-Stack Developer

Shubham P. Deore
Full-Stack Developer

- Contact Us Page:

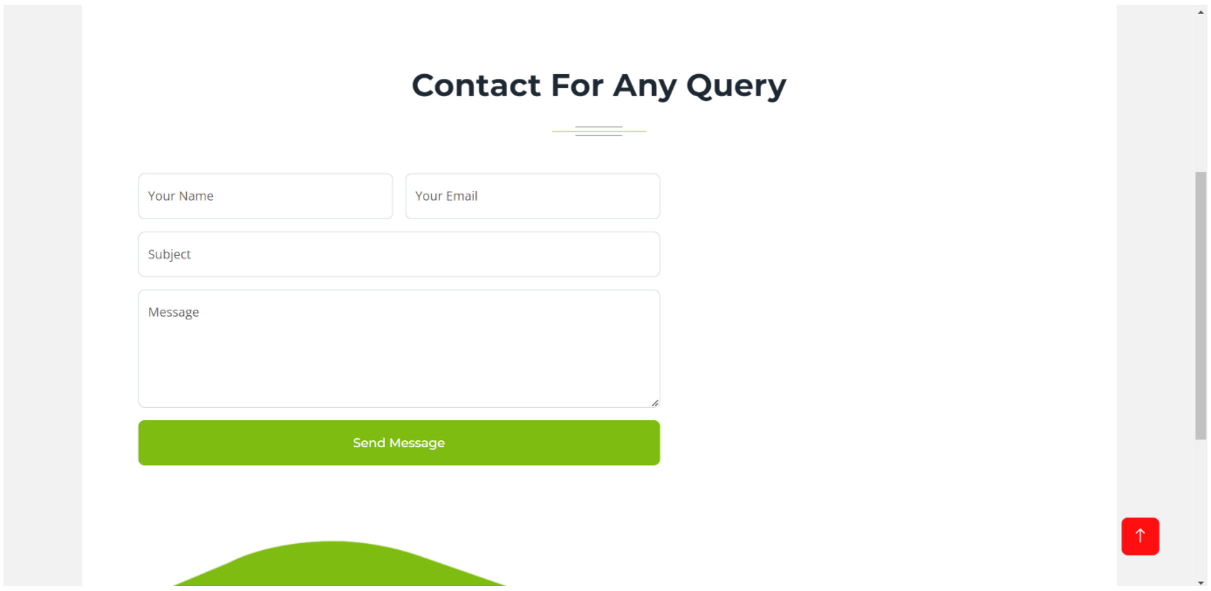


Contact Us

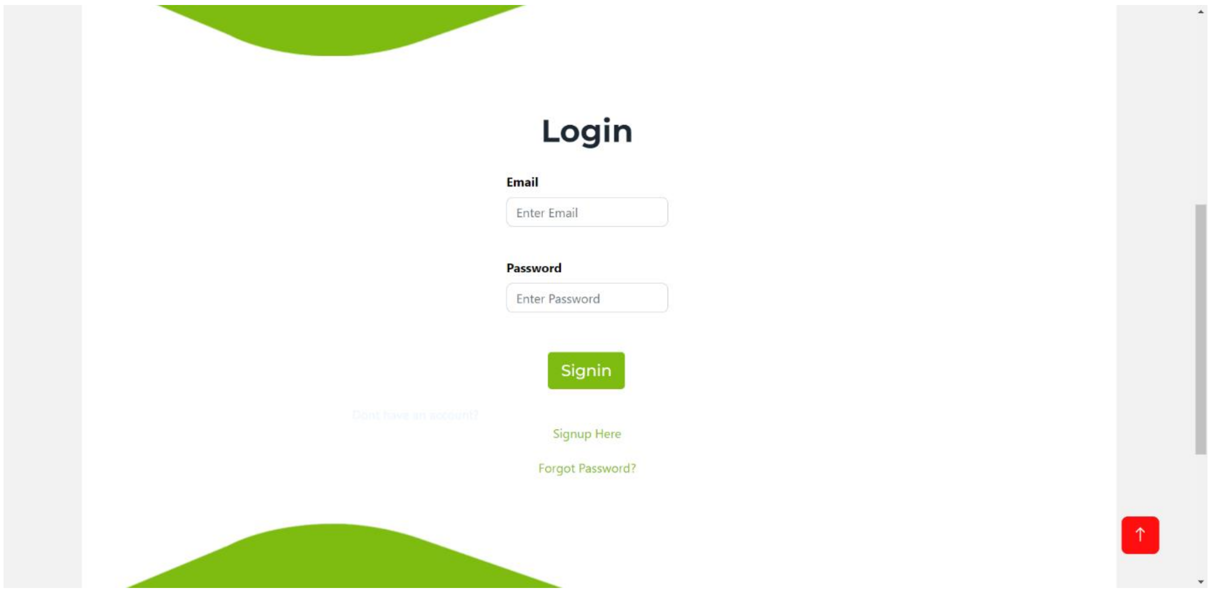
Contact For Any Query

Your Name

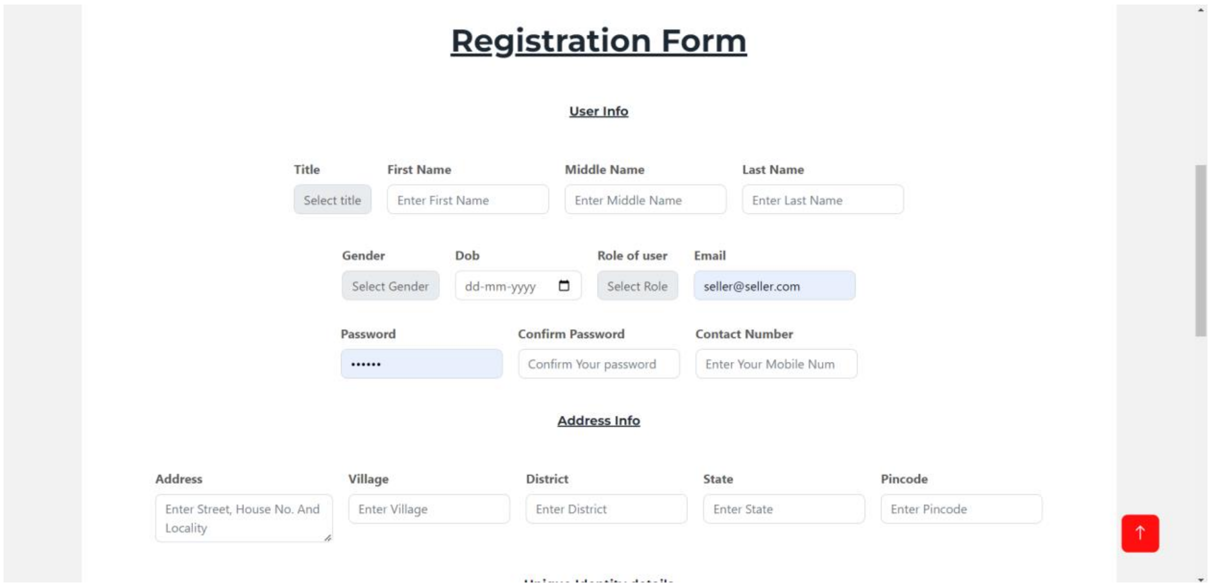
Your Email


Contact For Any Query
Your Name
Your Email
Subject
Message
Send Message

- Login Page:


Login
Email
Enter Email
Password
Enter Password
Signin
Dont have an account?
Signup Here
Forgot Password?

- Registration Page:



Registration Form

User Info

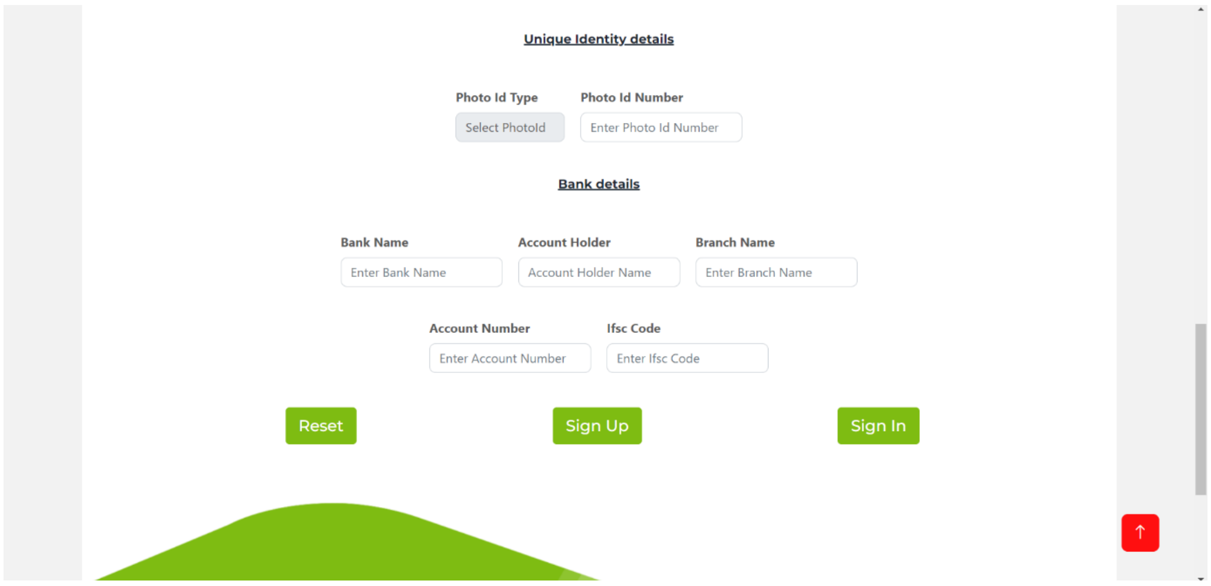
Title	First Name	Middle Name	Last Name
Select title	Enter First Name	Enter Middle Name	Enter Last Name

Gender	Dob	Role of user	Email
Select Gender	dd-mm-yyyy	Select Role	seller@seller.com

Password	Confirm Password	Contact Number
••••••	Confirm Your password	Enter Your Mobile Num

Address Info

Address	Village	District	State	Pincode
Enter Street, House No. And Locality	Enter Village	Enter District	Enter State	Enter Pincode



Unique Identity details

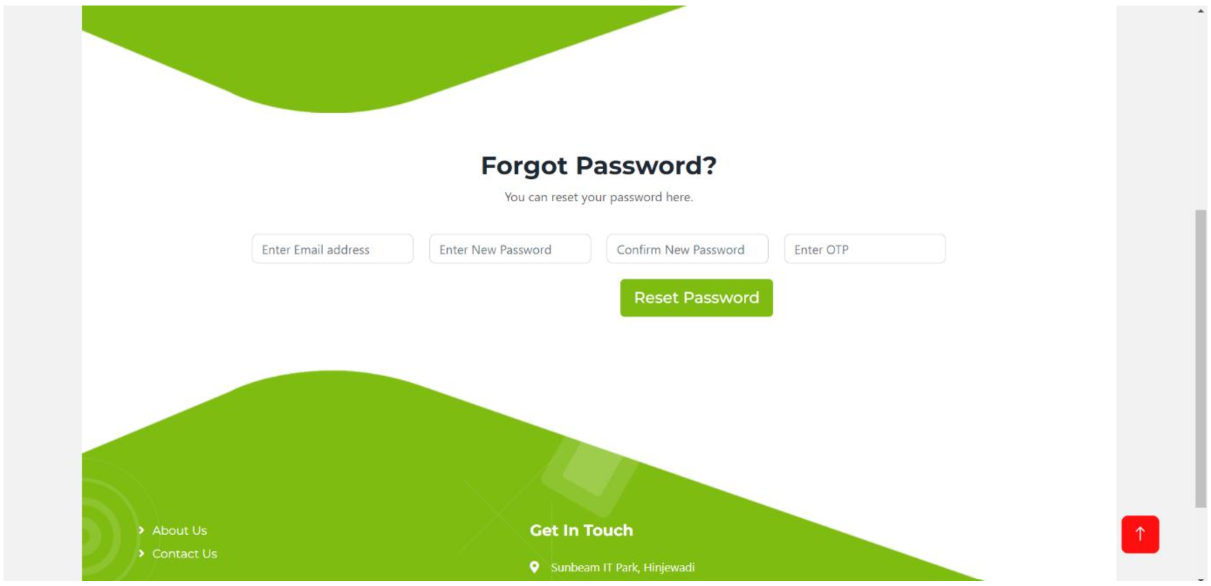
Photo Id Type	Photo Id Number
Select PhotoId	Enter Photo Id Number

Bank details

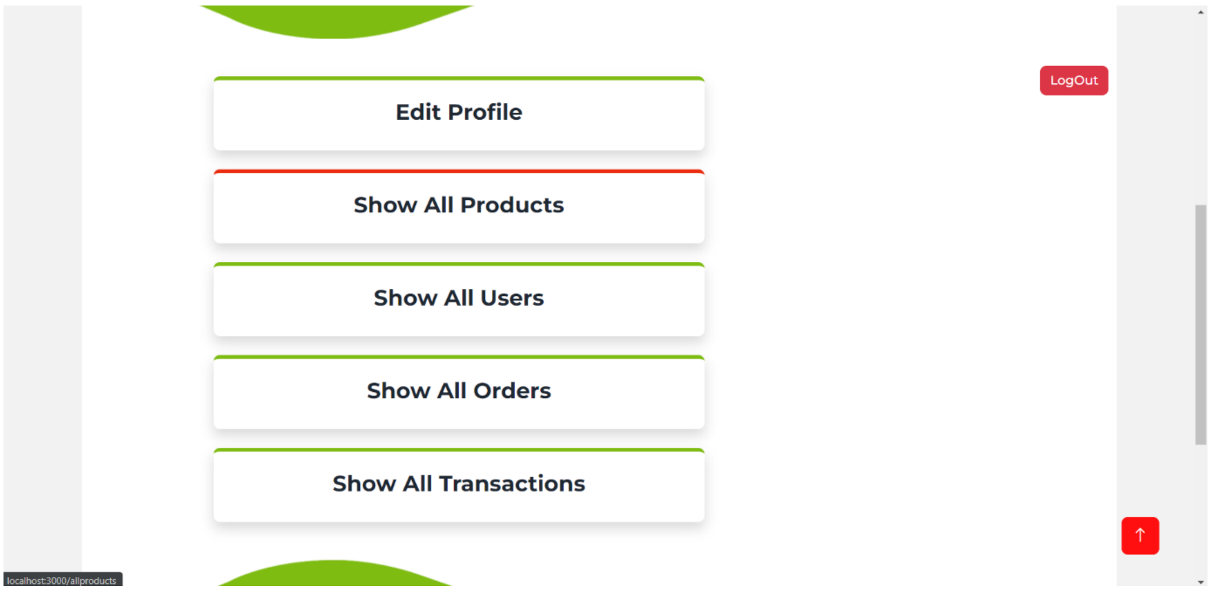
Bank Name	Account Holder	Branch Name
Enter Bank Name	Account Holder Name	Enter Branch Name

Account Number	Ifsc Code
Enter Account Number	Enter Ifsc Code

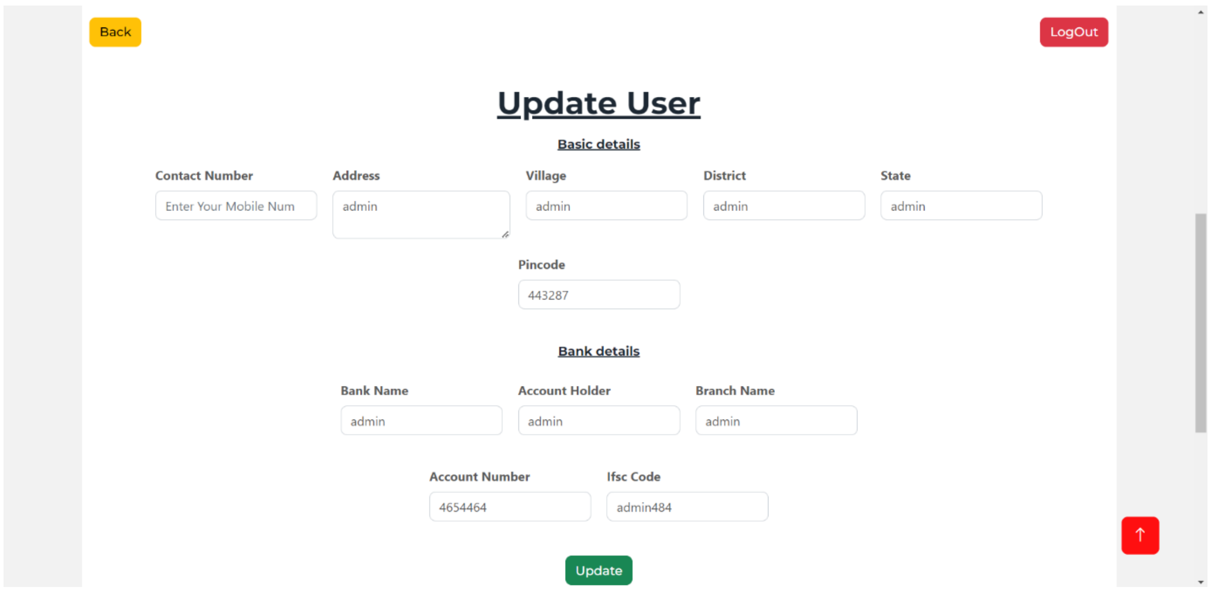
- Forgot Password Page:



- Admin Login:
  - Admin Dashboard:



- Admin Edit Profile:



Update User

Basic details

Contact Number	Address	Village	District	State
Enter Your Mobile Num	admin	admin	admin	admin

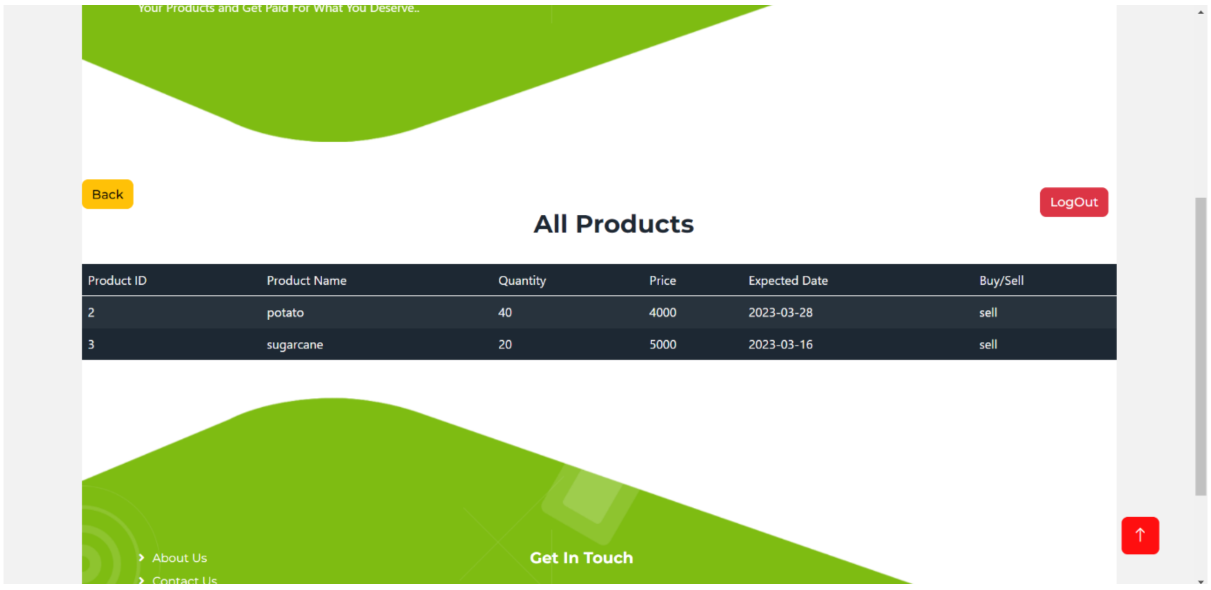
Pincode: 443287

Bank details

Bank Name	Account Holder	Branch Name
admin	admin	admin

Account Number	Ifsc Code
4654464	admin484

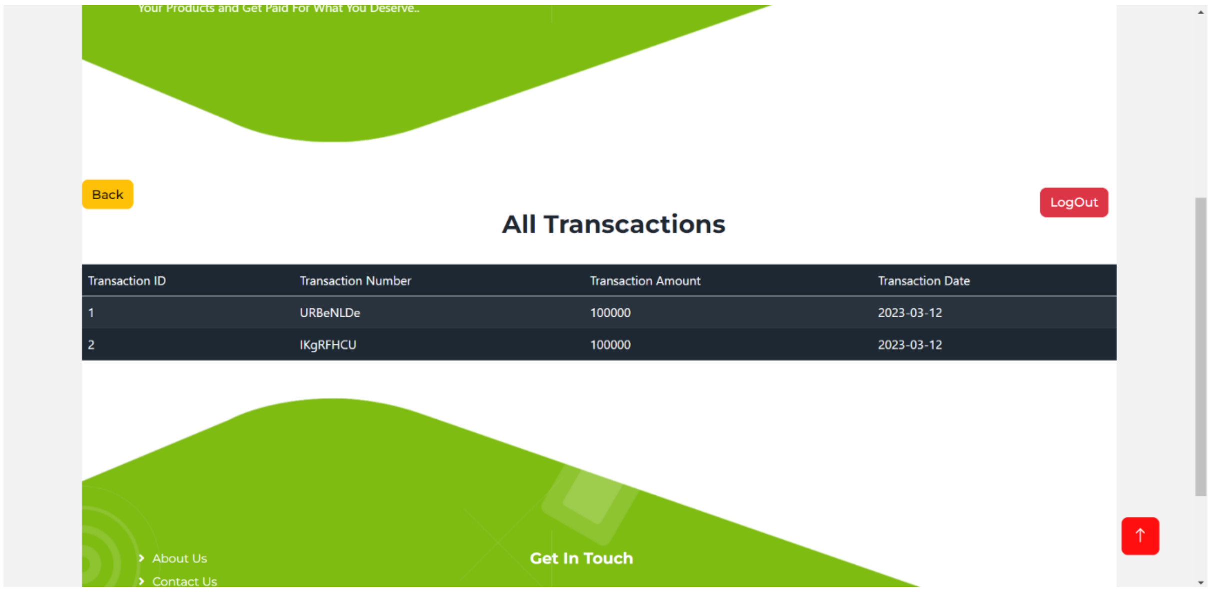
- Admin Show All Products:



All Products

Product ID	Product Name	Quantity	Price	Expected Date	Buy/Sell
2	potato	40	4000	2023-03-28	sell
3	sugarcane	20	5000	2023-03-16	sell

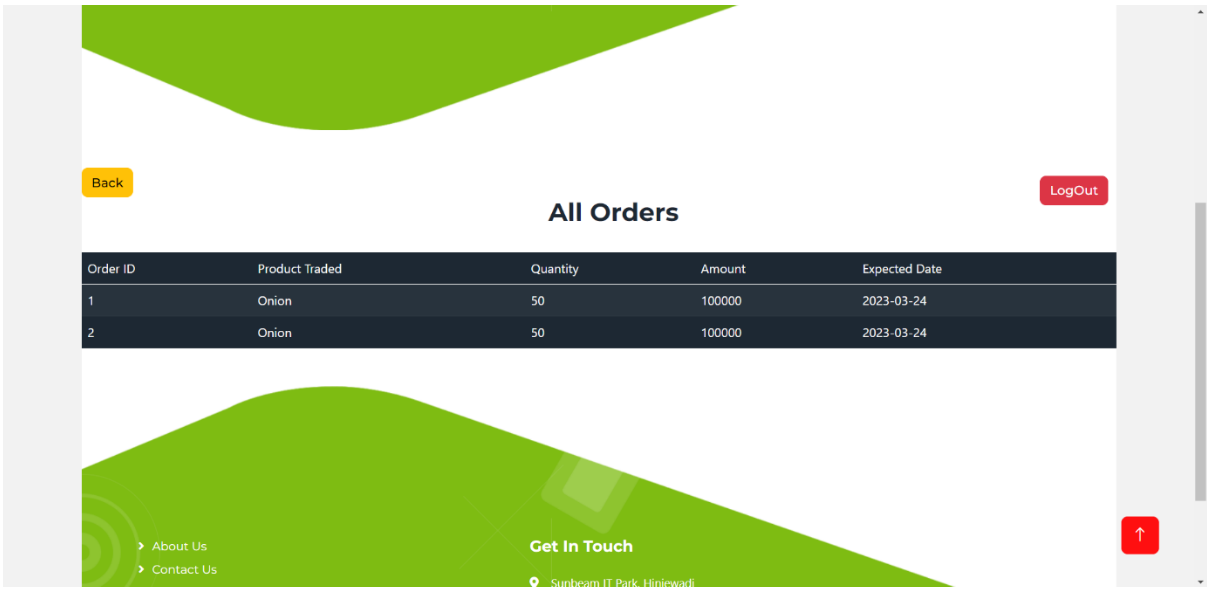
- Admin Show All Transactions:



All Transcactions

Transaction ID	Transaction Number	Transaction Amount	Transaction Date
1	URBeNLDe	100000	2023-03-12
2	IKgRFHCU	100000	2023-03-12

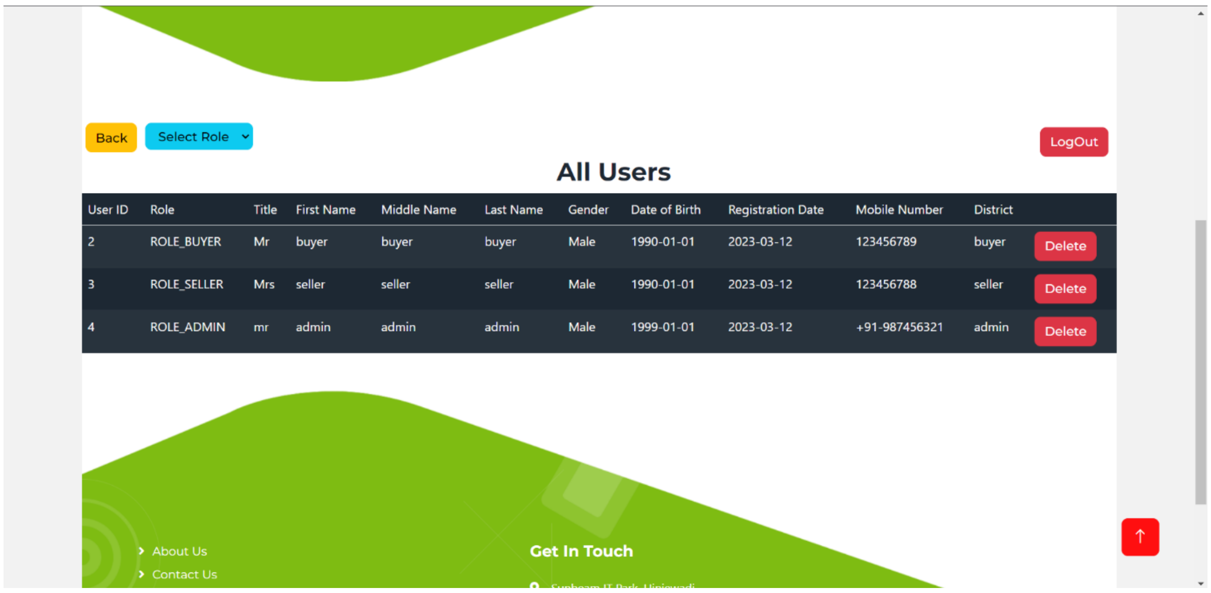
- Admin Show All Orders:



All Orders

Order ID	Product Traded	Quantity	Amount	Expected Date
1	Onion	50	100000	2023-03-24
2	Onion	50	100000	2023-03-24

- Admin Show All Users:

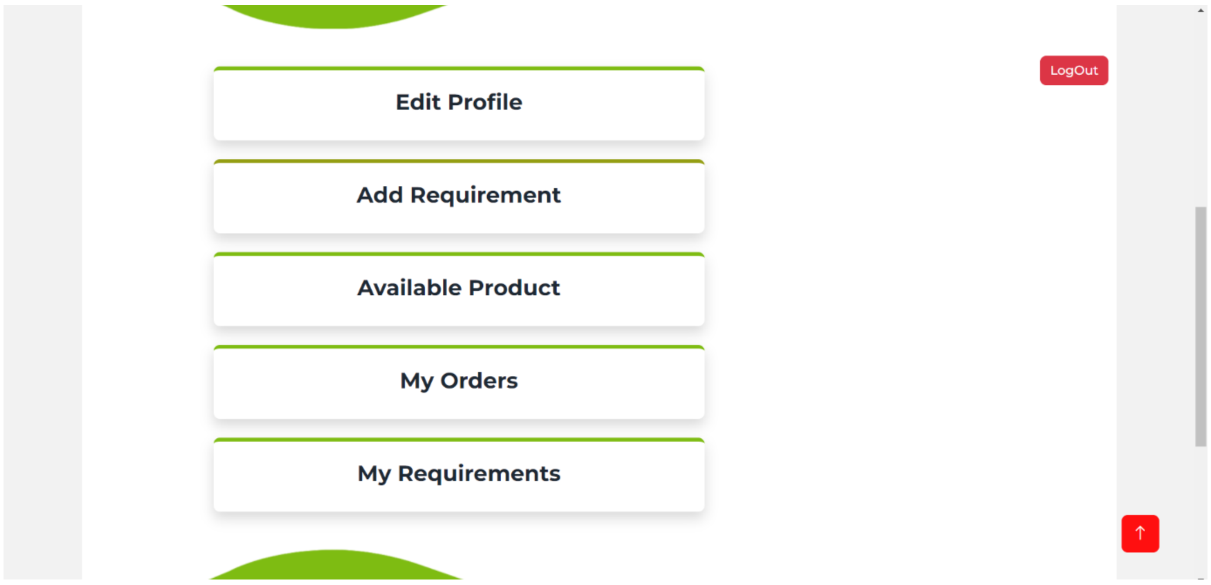


All Users

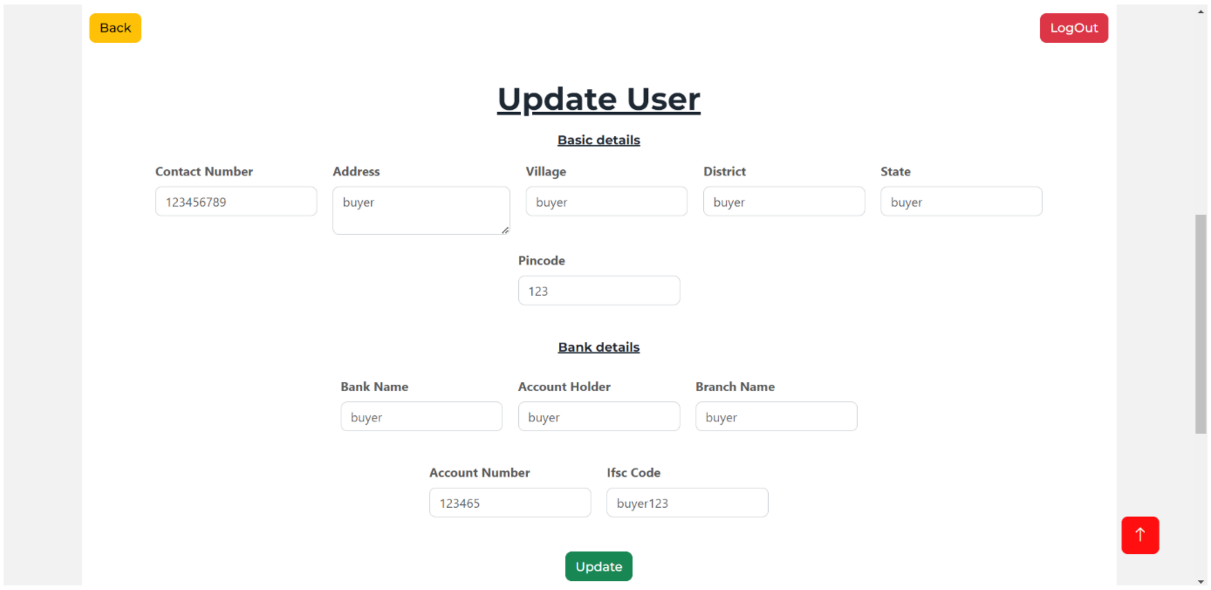
User ID	Role	Title	First Name	Middle Name	Last Name	Gender	Date of Birth	Registration Date	Mobile Number	District	
2	ROLE_BUYER	Mr	buyer	buyer	buyer	Male	1990-01-01	2023-03-12	123456789	buyer	Delete
3	ROLE_SELLER	Mrs	seller	seller	seller	Male	1990-01-01	2023-03-12	123456788	seller	Delete
4	ROLE_ADMIN	mr	admin	admin	admin	Male	1999-01-01	2023-03-12	+91-987456321	admin	Delete

- Buyer Login:

- Buyer Dashboard:



- Buyer Edit Profile:



Update User

Basic details

Contact Number	Address	Village	District	State
123456789	buyer	buyer	buyer	buyer

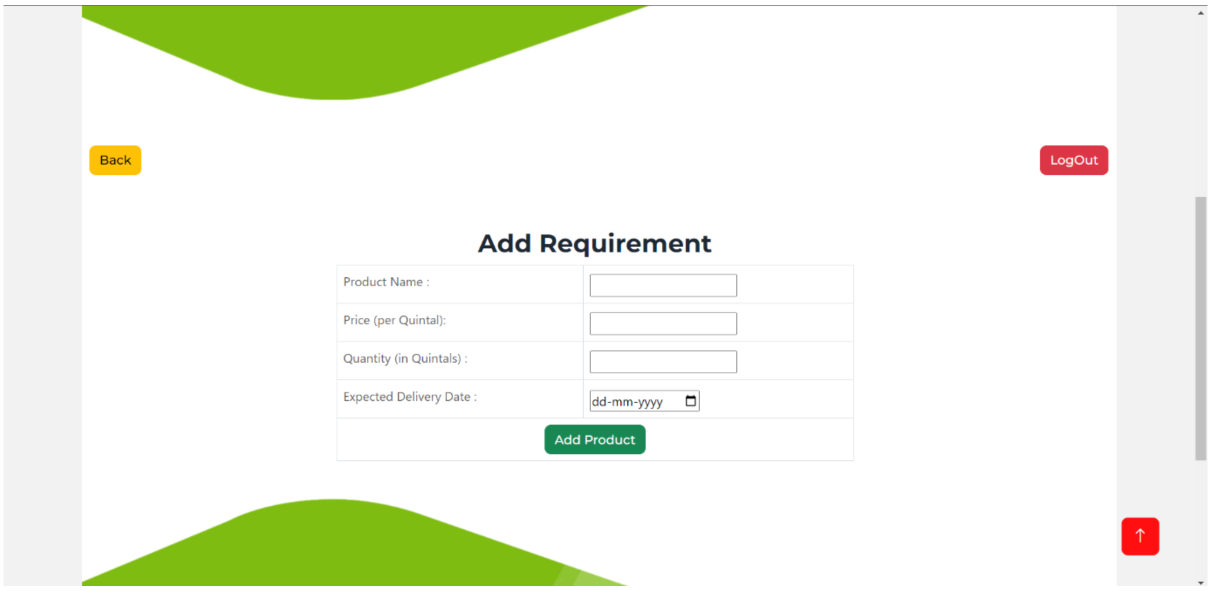
Pincode: 123

Bank details

Bank Name	Account Holder	Branch Name
buyer	buyer	buyer

Account Number	Ifsc Code
123465	buyer123

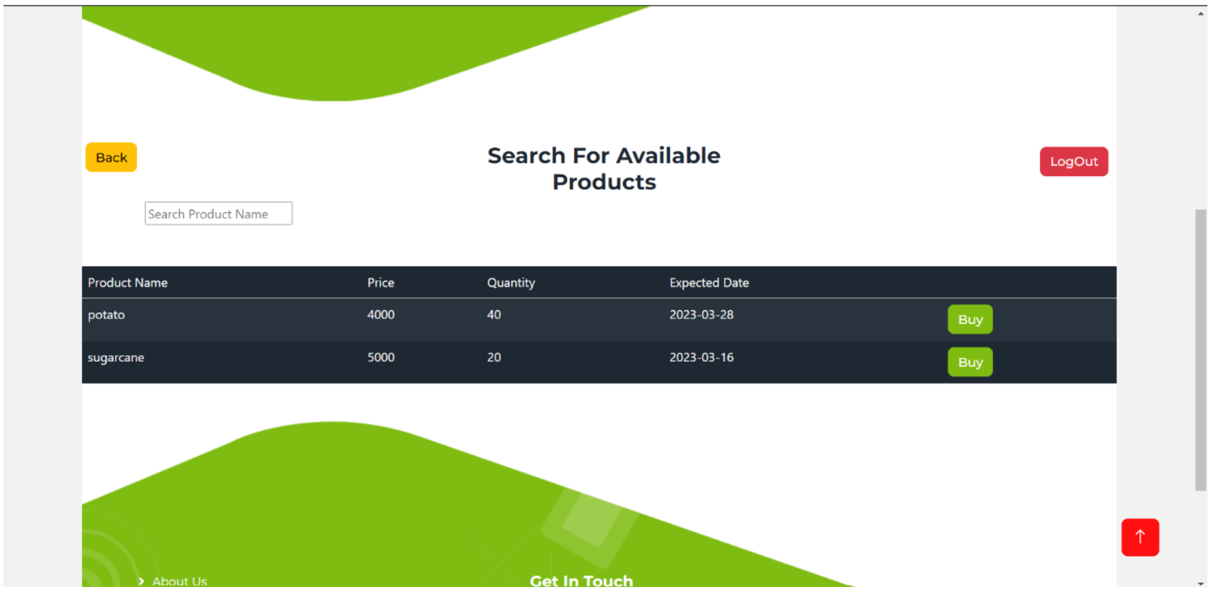
- Buyer Add Requirement:



Add Requirement

Product Name :	
Price (per Quintal):	
Quantity (in Quintals) :	
Expected Delivery Date :	dd-mm-yyyy

- Buyer Available Products:



Back

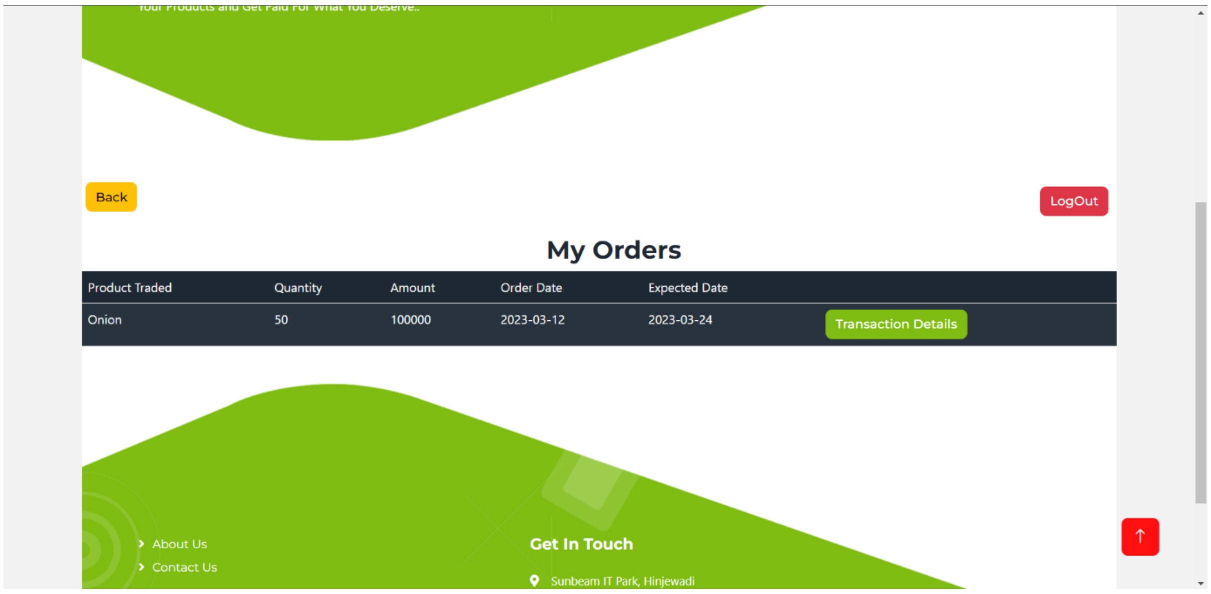
Search For Available Products

LogOut

Search Product Name

Product Name	Price	Quantity	Expected Date	
potato	4000	40	2023-03-28	Buy
sugarcane	5000	20	2023-03-16	Buy

- Buyer My Orders:



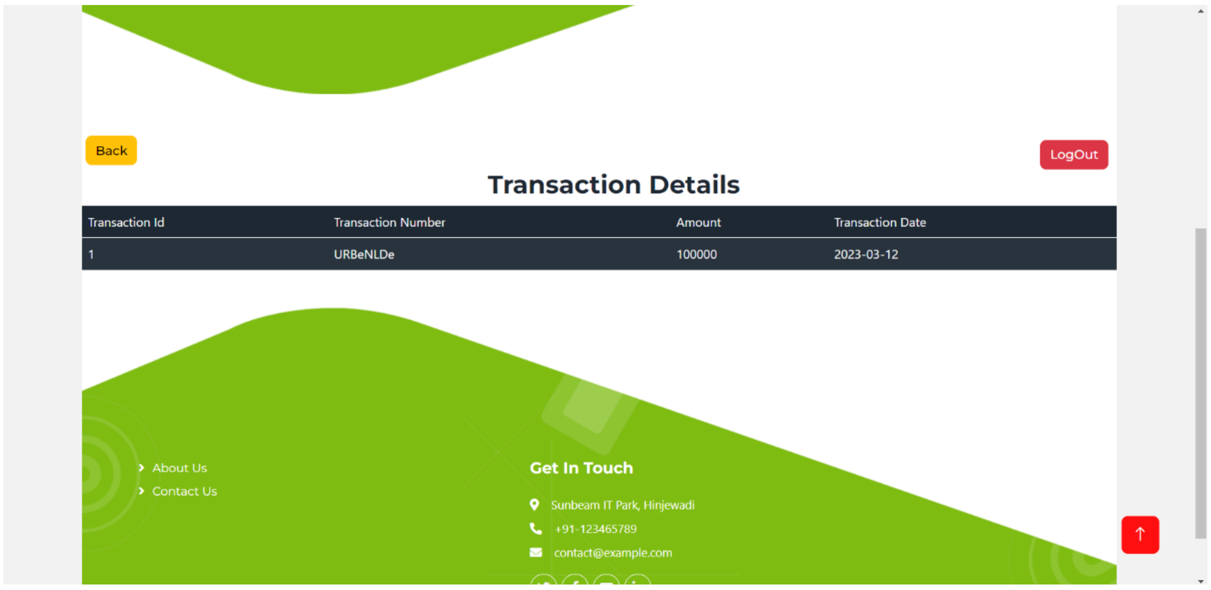
Back

LogOut

My Orders

Product Traded	Quantity	Amount	Order Date	Expected Date	
Onion	50	100000	2023-03-12	2023-03-24	Transaction Details

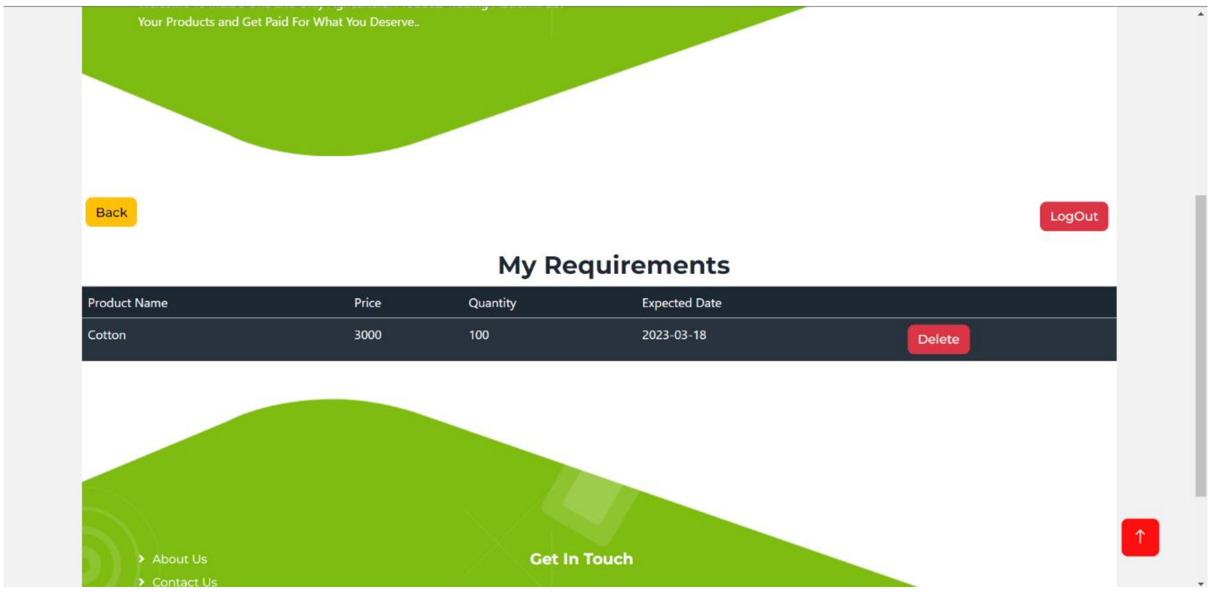
- Buyer Transaction Details:



Transaction Details

Transaction Id	Transaction Number	Amount	Transaction Date
1	URBeNLDe	100000	2023-03-12

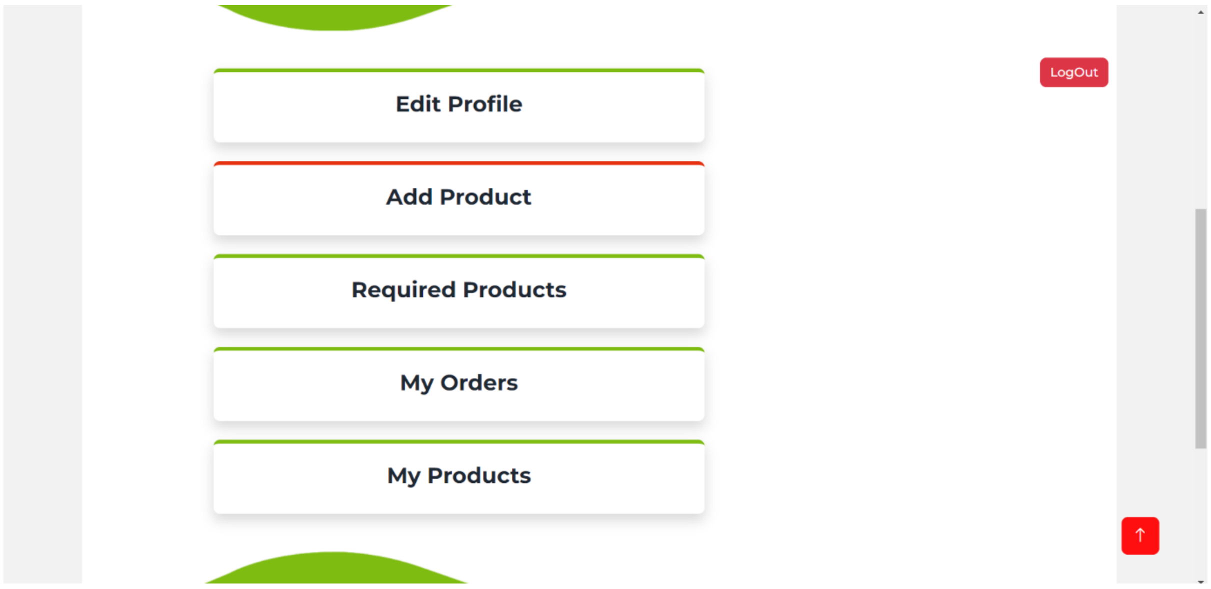
- Buyer My Requirements:



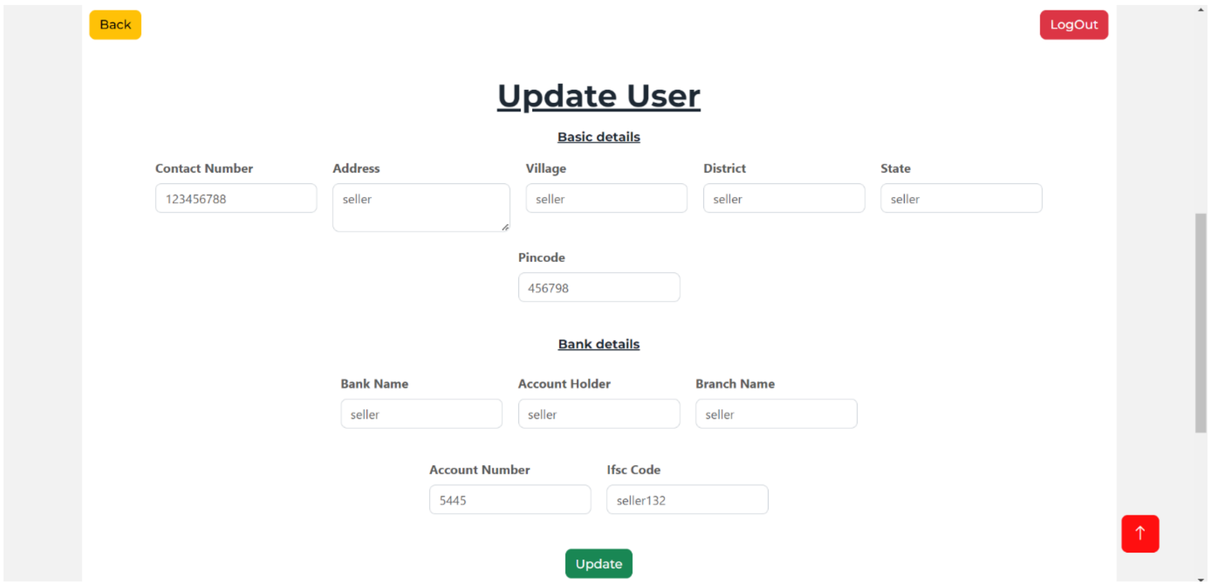
My Requirements

Product Name	Price	Quantity	Expected Date	
Cotton	3000	100	2023-03-18	Delete

- Seller Login:
  - Seller Dashboard:



  - Seller Edit Profile:



Update User

Basic details

Contact Number	Address	Village	District	State
123456788	seller	seller	seller	seller

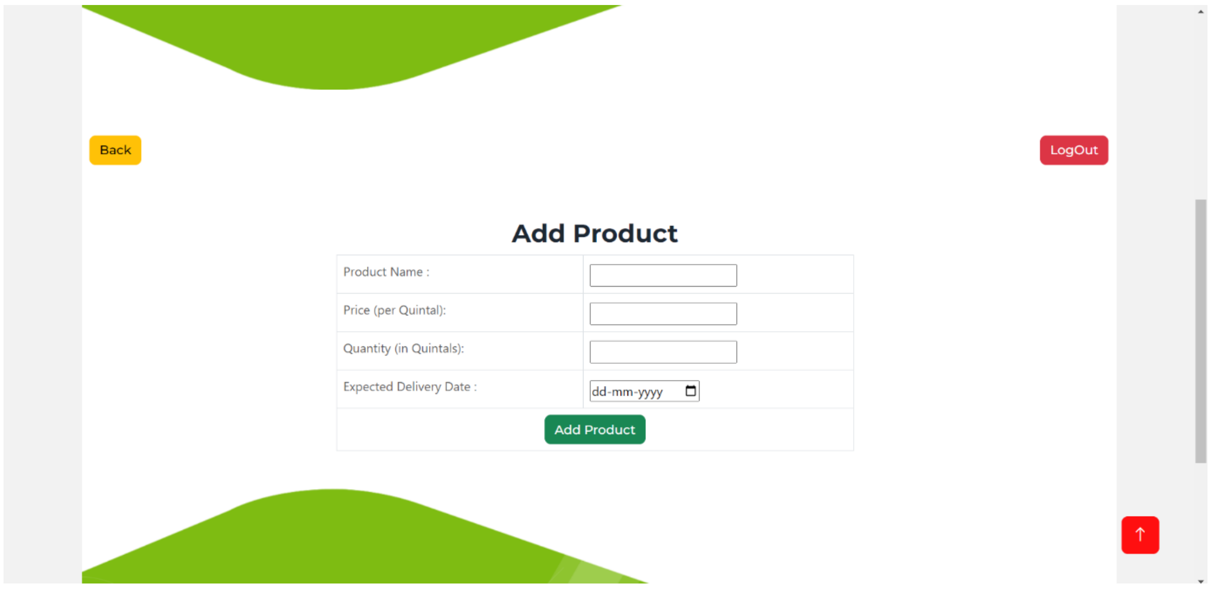
Pincode
456798

Bank details

Bank Name	Account Holder	Branch Name
seller	seller	seller

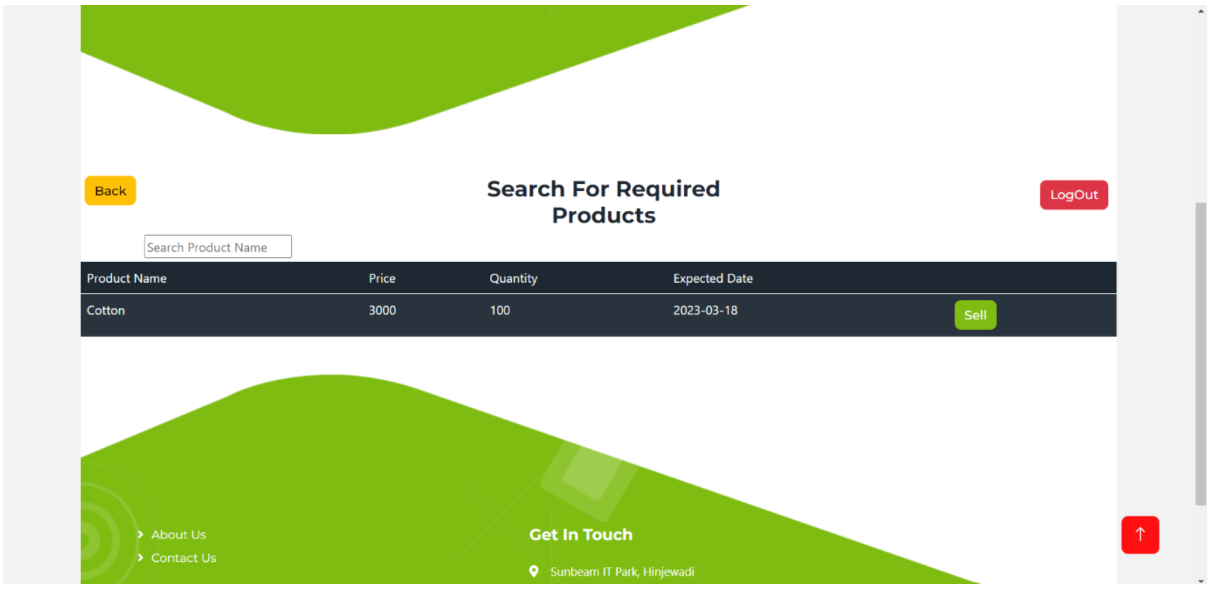
Account Number	Ifsc Code
5445	seller132

- Seller Add Product:



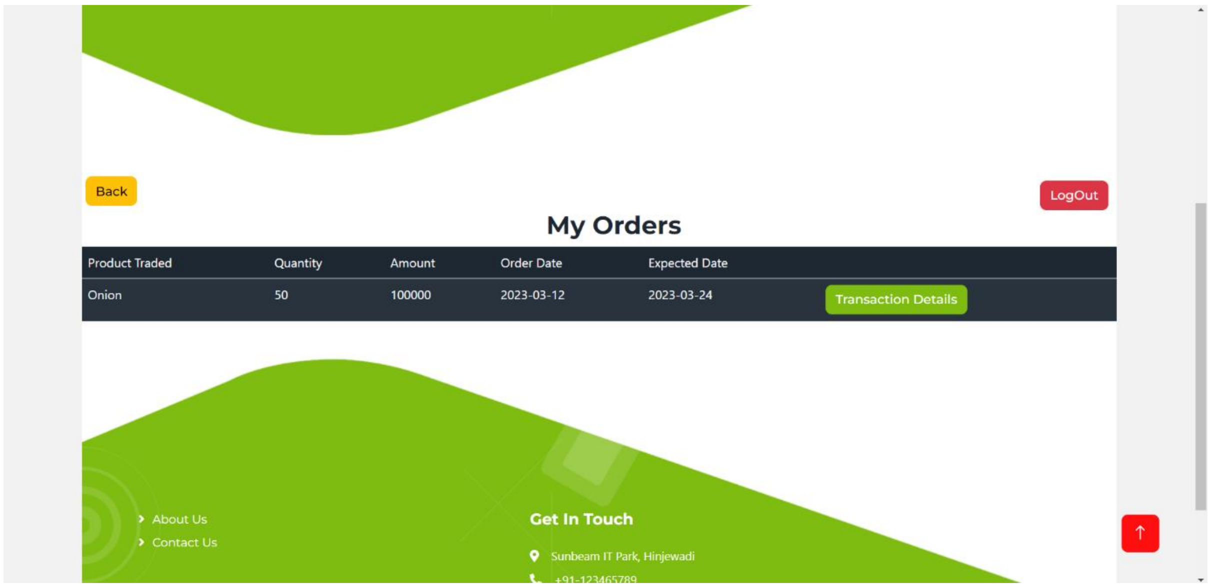
Product Name :	
Price (per Quintal):	
Quantity (in Quintals):	
Expected Delivery Date :	dd-mm-yyyy

- Seller Required Products:



Product Name	Price	Quantity	Expected Date	
Cotton	3000	100	2023-03-18	Sell

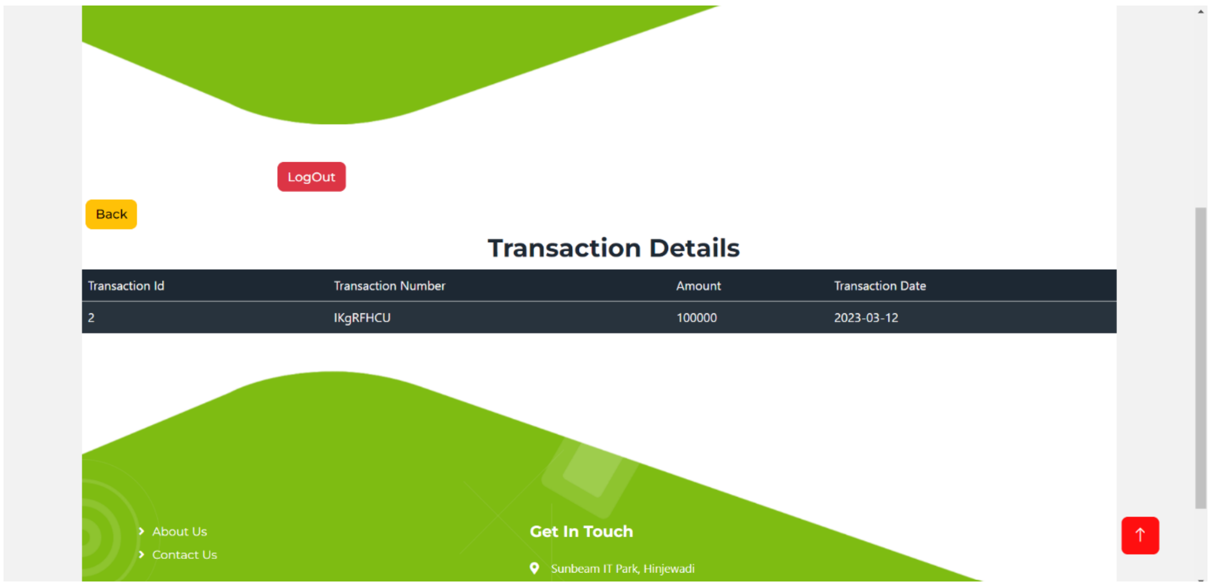
- Seller My Orders:



My Orders

Product Traded	Quantity	Amount	Order Date	Expected Date	
Onion	50	100000	2023-03-12	2023-03-24	Transaction Details

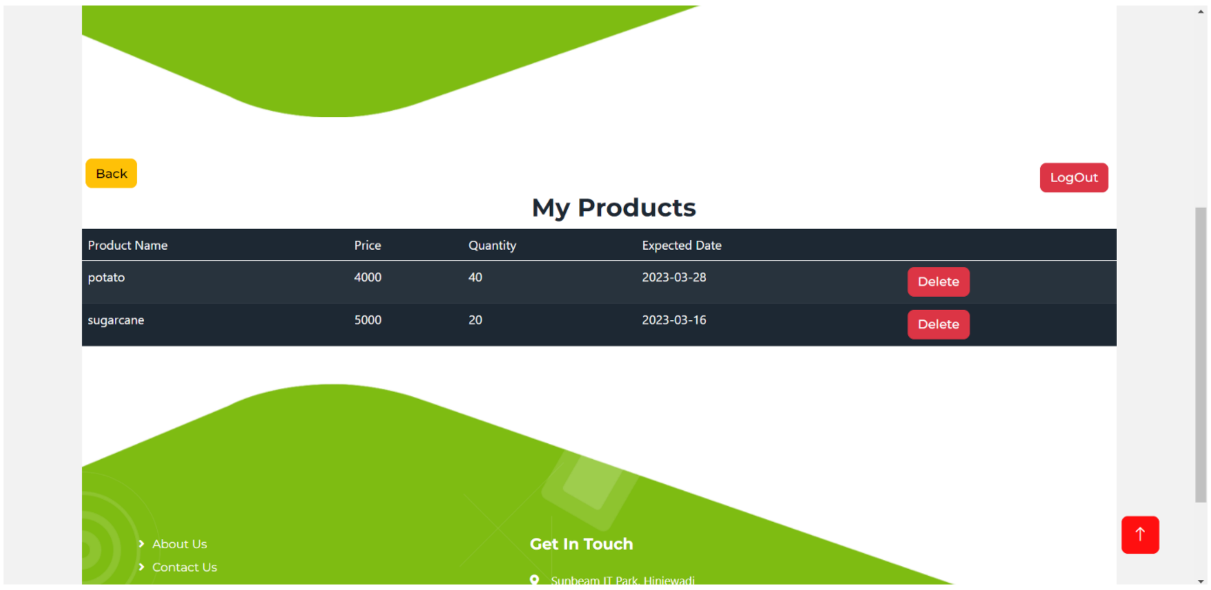
- Seller Transaction Details:



Transaction Details

Transaction Id	Transaction Number	Amount	Transaction Date
2	IKgRFHCU	100000	2023-03-12

- Seller My Products:



My Products

Product Name	Price	Quantity	Expected Date	
potato	4000	40	2023-03-28	Delete
sugarcane	5000	20	2023-03-16	Delete

6. CODING STANDARDS IMPLEMENTED

Naming and Capitalization

Below summarizes the naming convention for identifiers in Pascal casing is used mainly (i.e., capitalize first letter of each word) with camel casing (capitalize each word except for the first one) being used in certain circumstances.

IDENTIFIER	CASE	EXAMPLES	DESCRIPTION
Class	Pascal	User, Product, OrderExecuted, Transaction, etc.	Class names should be based on "objects" or "real things" and should generally be nouns. No '_' signs allowed. Do not use type prefixes like 'C' for class.
Method	Camel	getAllUsers, getAllProducts, getMyOrders, etc	Methods should use verbs or verb phrases.
Parameter	Camel	first_name, dob, etc.	Use descriptive parameter names. Parameter names should be descriptive enough that the name of the parameter and its type can be used to determine its meaning in most scenarios.
Interface	Pascal	ProductDao, UserDao, UserService, etc.	Do not use the '_' sign
Exception Class	Pascal with "Exception" suffix		

Comments

- Comment each type, each non-public type member, and each region declaration.
- Use end-line comments only on variable declaration lines. End-line comments are comments that follow code on a single line.
- Separate comments from comment delimiters (apostrophe) or // with one space.
- Begin the comment text with an uppercase letter.
- End the comment with a period.
- Explain the code; do not repeat it.