PHONENEST – DIGITAL MOBILE SELLING PLATFORM

A PROJECT COMPONENT REPORT

Submitted by

PRADEEP KUMAR. R	**(Reg. No. 202204119)**
PRAVEEN. M	**(Reg. No. 202204121)**

for the Theory Cum Project Component

of

19CS694 – WEB USER INTERFACE DESIGN

during

VI Semester – 2024 – 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

(An Autonomous Institution affiliated to Anna University Chennai)

May 2025

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

(An Autonomous Institution affiliated to Anna University Chennai)

Department of Computer Science and Engineering

BONAFIDE CERTIFICATE

Certified that this project component report titled **PHONENEST-DIGITAL MOBILE SELLING PLATFORM** is the bonafide work of **R.PRADEEP KUMAR (Reg. No. 202204119),** and **M. PRAVEEN (Reg. No. 202204121)** who carried out this work under my guidance for the Theory cum Project Component course **"19CS694 – WEB USER INTERFACE DESIGN"** during the sixth semester.

Dr.K. MUTHAMIL SUDAR, M.E., Ph.D.
Assistant Professor
Course Instructor
Department of Computer Science & Engg.
Mepco Schlenk Engineering College
Sivakasi.

Dr. J. RAJA SEKAR, M.E.,Ph.D.
Professor
Head of the Department
Department of Computer Science & Engg.
Mepco Schlenk Engineering College
Sivakasi.

Submitted for viva-Voce Examination held at **MEPCO SCHLENK ENGINEERING COLLEGE (Autonomous), SIVAKASI** on **………/……/…. 20..........**

Internal Examiner **External Examiner**

ABSTRACT

In today's fast-paced technological era, finding the perfect mobile phone that suits individual needs can be a challenging and time-consuming task. To address this, we have developed PhoneNest, an application designed to revolutionize the way users browse, compare, and purchase mobile devices. By leveraging mobile technology, PhoneNest provides a seamless digital platform where users can explore a wide range of mobile phones, and complete purchases effortlessly. This application enables users to register, log in, and browse through an extensive catalog of products with detailed specifications. Users can view product availability and opt for direct delivery. Additionally, the app integrates a real-time database to ensure smooth operations, secure data handling, and up-to-date product information. By simplifying the mobile shopping experience, PhoneNest aims to bridge the gap between customers and retailers, providing a convenient, efficient, and reliable solution for all mobile shopping needs.

ACKNOWLEDGEMENT

First and foremost, we thank the **LORD ALMIGHTY** for his abundant blessings that is showered upon our past, present and future successful endeavors.

We extend our sincere gratitude to our college management and Principal **Dr. S. Arivazhagan M.E., Ph.D.,** for providing sufficient working environment such as systems and library facilities. We also thank him very much for providing us with adequate lab facilities, which enable us to complete our project.

We would like to extend our heartfelt gratitude to **Dr. J. Raja Sekar M.E., Ph.D.,** Professor and Head, Department of Computer Science and Engineering, Mepco Schlenk Engineering College for giving me the golden opportunity to undertake a project of this nature and for his most valuable guidance given at every phase of our work.

We would also like to extend our gratitude and sincere thanks to **Dr.K. Muthamil Sudar M.E., Ph.D.,** Assistant Professor, Department of Computer Science and Engineering, Mepco Schlenk Engineering College for being our Project Mentor. He has put his valuable experience and expertise in directing, suggesting and supporting us throughout the Project to bring out the best.

Our sincere thanks to our revered **faculty members and lab technicians** for their help over this project work.

Last but not least, we extend our indebtedness towards out beloved family and our friends for their support which made the project a successful one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 PERSPECTIVE

This project aims to revolutionize the mobile phone shopping experience by providing a seamless digital platform for users and admins. With a focus on user convenience and efficient management, PhoneNest offers a user-friendly interface where users can register, log in, browse a wide range of mobile phones, add items to their cart, view purchase history, and manage their profiles. On the admin side, they can register, log in, add, delete, and update product listings, and track customer orders. This system enhances the user experience by eliminating the need for physical visits to mobile stores, streamlining the shopping process, and enabling users to access the platform anytime and anywhere.

1.2 OBJECTIVES

The main objective of PhoneNest is to provide a seamless, efficient, and user-friendly platform for users to explore and purchase mobile phones online. This application aims to bridge the gap between customers and mobile retailer, ensuring convenience and accessibility. By offering features like product browsing, filtering, it empowers users to make informed decisions, while also enabling store administrators to manage their inventory and orders efficiently.

1.3 SCOPE

The scope of the project is to create an efficient and user-friendly Digital Mobile Selling platform where users can easily browse mobile phones, and make secure purchases. At the same time, store administrators will be able to manage product listings, track inventory, and process customer orders seamlessly. The overarching goal is to develop a dynamic and user-centric platform that redefines the standards of online mobile shopping by providing convenience, reliability, and a modern shopping experience.

CHAPTER 2

REQUIREMENT DESCRIPTION

2.1 FUNCTIONAL REQUIREMENTS

The functional requirement in Digital Mobile Selling Platform system is the collective information about what are the operations available in the system.

- The system should provide authentication functionality for valid users, including users and admins.
- Users should be able to browse available mobile phones.
- The system should allow users to add mobiles to their cart and modify quantities as needed.
- It should provide the functionality for admins to add, delete and modify mobiles.
- The system should display the mobile details to all the users, uploaded at the admin side.

2.2 NON-FUNCTIONAL REQUIREMENTS

The non-functional requirement describes about platform and physical resource required for building the Digital Mobile Selling Platform.

- The User details, mobile details, and order details should be securely stored in the MongoDB database.
- The application must be user friendly and must be very interactive to the user as well as admin
- Users must have access to the internet while using the application to browse mobiles, place orders, and receive notifications.

CHAPTER 3

SYSTEM DESIGN

3.1 ARCHITECTURE DESIGN

Architectural diagram implies the flow of the system. The flow starts whether the user as well as admin has account or not. If the user or admin had an account, it directly takes to the login page. After the user logged into the system, it enables the user to view mobile details, availability and place orders. The user could also view his/her purchase details. On the other hand, on the admin side, they can upload images and add mobiles, delete and modify them to be displayed for users. The admins could also view recent orders through our application.

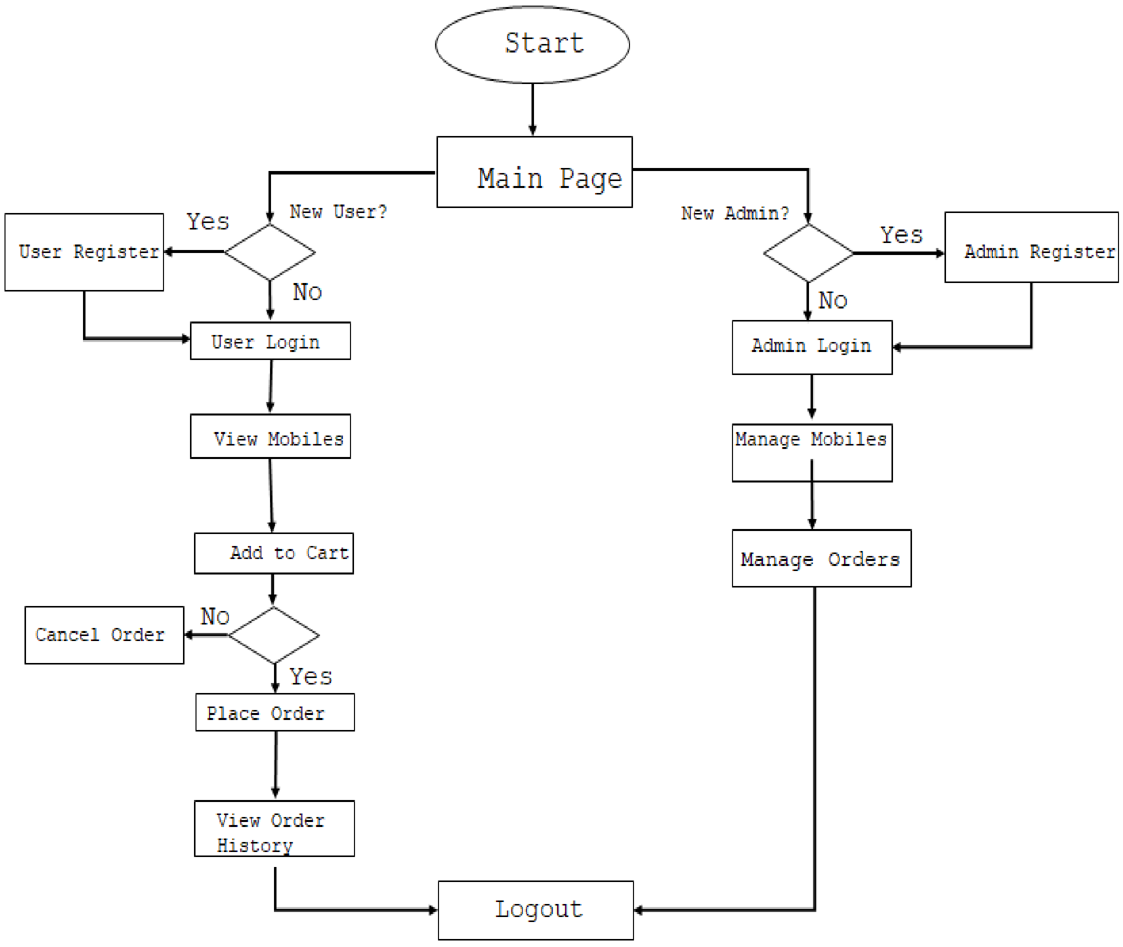


Figure 3.1: Architecture Diagram of Digital Mobile Selling Platform System

3.2 DESIGN COMPONENTS

3.2.1 Front End:

The PhoneNest Digital Mobile Selling Platform utilizes AngularJS for the front-end development, providing users with an interactive and visually appealing interface.

3.2.2 Back End:

The system uses mongo database and node js for back end to store data.

3.3 DATABASE DESCRIPTION

Listed below gives a description of database document schemas used for Digital Mobile Selling Platform.

3.3.1 User Description

As shown in **Table 3.1**, The user description stores detailed information about users who have registered on the platform. This includes fields such as name, email address, username, password, dob, and address. The user collection serves as the primary source of user data for authentication, registration, and profile management functionalities.

Table 3.1: User Description

Attribute Name	Type	Constraint(s)	Description
Name	String	Not null	Name of the user
DOB	String	Maximum length of 10	Date of birth of user
Address	String	Not null	Address of user
Email id	String	Should contain @symbol	Email id of the user
Username	String	Not null and unique	Username for the login
Password	String	Not null	Password for the login

3.3.2 Mobile Description

The mobile collection contains data pertaining to the mobiles added on the platform. Each document in this collection includes attributes such as mobile name, description, stock, price and an image. The data of mobiles are maintained by the admin for each mobile.

Table 3.2: Mobile Description

Attribute Name	Type	Constraint(s)	Description
Name	String	Not null	Name of the mobile
Price	Double	Not null	Price of the mobile
Description	String	Not null	Description for the mobile
Stock	Integer	Not null	No. of mobiles available
Image	File	Not null	Image of each restaurant

3.3.3 Cart Items Description

The Cart items collection contains data about the mobile items added to cart by the customer. It includes attributes such as Cart ID, Mobile item name, quantity ordered and price of each mobile item. This data is to be confirmed for purchase and will be delivered to the customer.

Table 3.3: Cart Item Description

Attribute Name	Type	Constraint(s)	Description
Cart ID	Integer	Not null	Unique number assigned to each cart containing mobile items.
Name	String	Not null	Name of the mobile item
Quantity	Integer	Not null	Quantity available
Total	Double	Not null	Total of mobile items

3.3.4 Purchase history Description

The purchase history collection contains data about the mobile details purchased by the customer with Date and Time of purchase. It also includes the status of delivery and of purchase. This data can be viewed by customers for reference.

Table 3.4: Purchase History Description

Attribute Name	Type	Constraint(s)	Description
Items	String	Not null	Name of the mobile
Quantity	Integer	Not null	Quantity of the mobile
Price	Double	Not null	Price of the mobile
Username	String	Not null	Customer id of purchase
Date	Date	Follow the format dd-mm-yyyy hr:min:sec	Date of Purchase
Total	Double	Not null	Total amount of Purchase
Status	String	Not null	Status of Delivery (Delivered/ In Progress/ cancelled)

3.3.5 Orders Description

The orders collection is responsible for storing information related to user orders. Each order document includes fields such as order number, order date, mobile item, quantity and status for order placement and delivery. This collection facilitates admin for the management of user purchases and order tracking.

Table 3.5: Orders Description

Attribute Name	Type	Constraint(s)	Description
Order No	Integer	Not null	Order number of user purchase
Order date	Date	Follow the format dd-mm-yyyy	Order date of user purchase
Item	String	Not null	Items ordered by user
Total	Double	Not null	Total amount of items ordered
Status	String	Not null	Status of order process (Active/Cancelled)

3.4 LOW LEVEL DESIGN

The following section illustrates the functionalities of the system. This includes sign up and login for both user and admin and includes functionalities like purchase mobiles, add mobiles, orders etc.

3.4.1 UserRegister

Table 3.6 shows the signup details of the user.

Table 3.6 User Register Details

Files used	register.component.html , register.component.css, register.component.ts
Short Description	Allows the user to register to the application
Arguments	Name, Email id, Password, DOB, Address
Return	Success/Failure in registering
Pre-Condition	The user must have an email account
Post-Condition	The login page will be displayed
Exception	Failure of validation check
Actor	User

3.4.2 UserLogin

Table 3.7 shows the login details of the user.

Table 3.7 Login Details

Files used	login.component.html,login.component.css, login.component.ts
Short Description	Allows the user to login to the application
Arguments	Username, Password
Return	Success/Failure in login
Pre-Condition	The user must have a registered account
Post-Condition	The user page will be displayed
Exception	Invalid Username or password
Actor	User

3.4.3 AdminLogin

Table 3.8 shows the login details of Admin

Table 3.8 Admin Login Details

Files used	adminLogin.component.html , adminLogin.component.css, adminLogin.component.ts

Short Description	Allows the admin to login to the application
Arguments	Username, password
Return	Success/Failure in Login
Pre-Condition	The admin must have an account
Post-Condition	The admin dashboard page will be displayed
Exception	Invalid Username or password
Actor	Admin

3.4.4 Order Mobiles

Table 3.9 shows the mobile items ordered by user

Table 3.9 Mobile Order Details

Files used	ordermobile.component.html, ordermobile.component.css, ordermobile.component.ts
Short Description	Allows the user to place order the mobile items
Arguments	mobile name, Quantity, price
Return	Success/Failure in placing order
Pre-Condition	The user must add the mobiles to cart
Post-Condition	The purchase history will be updated
Exception	Cancel items on confirm purchase
Actor	User

3.4.5 Add Mobile items

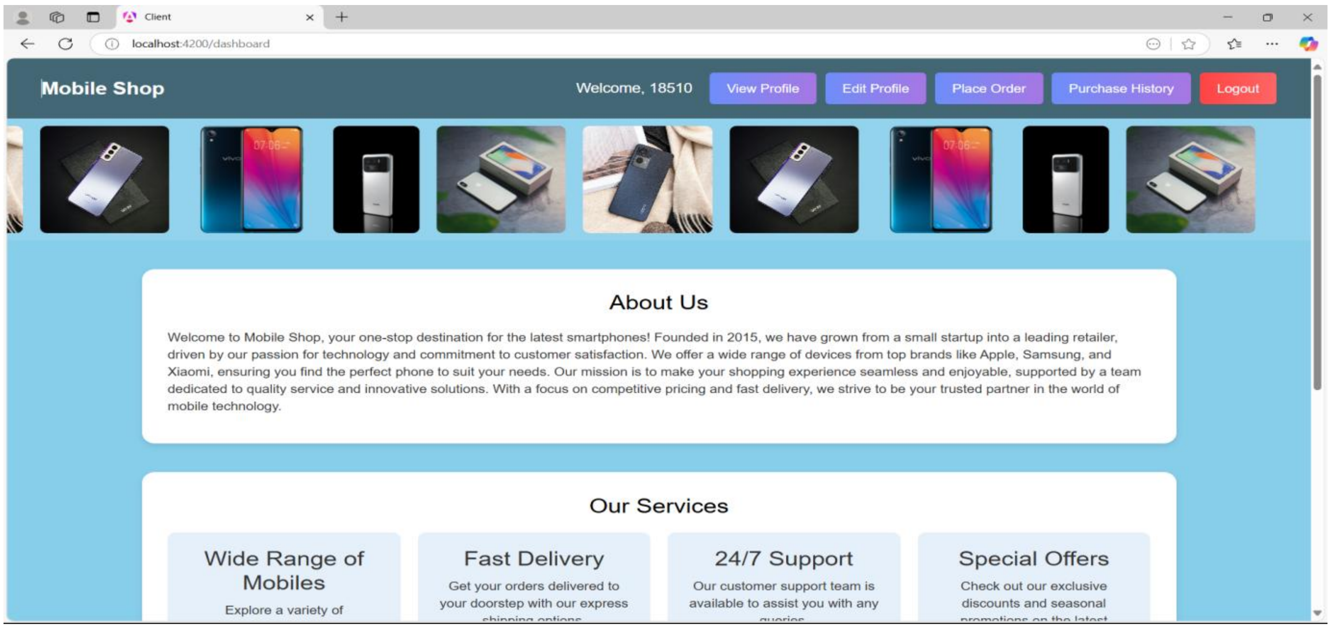
Table 3.10 shows the mobile items added by admin

Table 3.10 Add Mobile Details

Files used	addmobile.component.html, addmobile.component.css, addmobile.component.ts
Short Description	Allows the admin to add,delete and modify mobiles
Arguments	Mobile name, Quantity, price
Return	Success/Failure in Mobile Items addition/deletion/updation
Pre-Condition	The admin must login to add items to it.
Post-Condition	The added mobile items by admin will be displayed to user for order
Exception	nil
Actor	Admin

3.5 USER INTERFACE DESIGN

3.5.1 Main Activity



About Us

Welcome to Mobile Shop, your one-stop destination for the latest smartphones! Founded in 2015, we have grown from a small startup into a leading retailer, driven by our passion for technology and commitment to customer satisfaction. We offer a wide range of devices from top brands like Apple, Samsung, and Xiaomi, ensuring you find the perfect phone to suit your needs. Our mission is to make your shopping experience seamless and enjoyable, supported by a team dedicated to quality service and innovative solutions. With a focus on competitive pricing and fast delivery, we strive to be your trusted partner in the world of mobile technology.

Our Services

Wide Range of Mobiles
Explore a variety of

Fast Delivery
Get your orders delivered to your doorstep with our express

24/7 Support
Our customer support team is available to assist you with any

Special Offers
Check out our exclusive discounts and seasonal

Figure 3.2: Main layout homepage of Digital Mobile Selling Platform

This **Figure 3.2** provides an interface for main activity of the Digital mobile selling platform where both the users and admin can use the interface.

3.5.2 Admin Login page

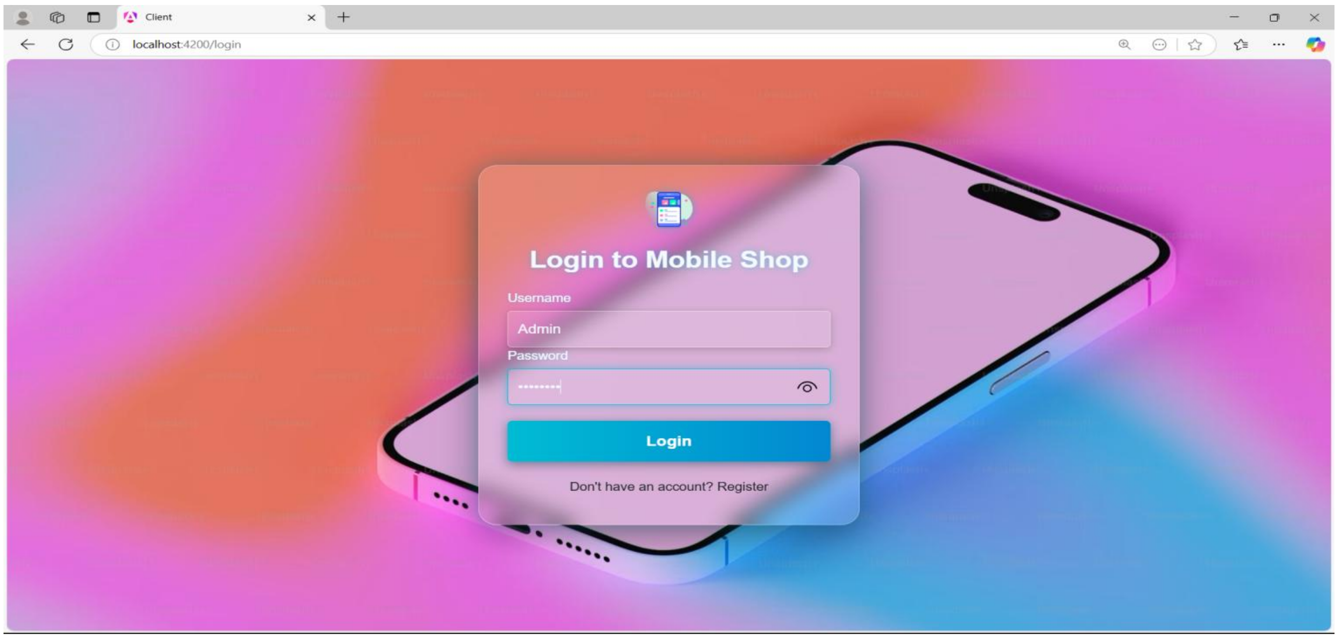
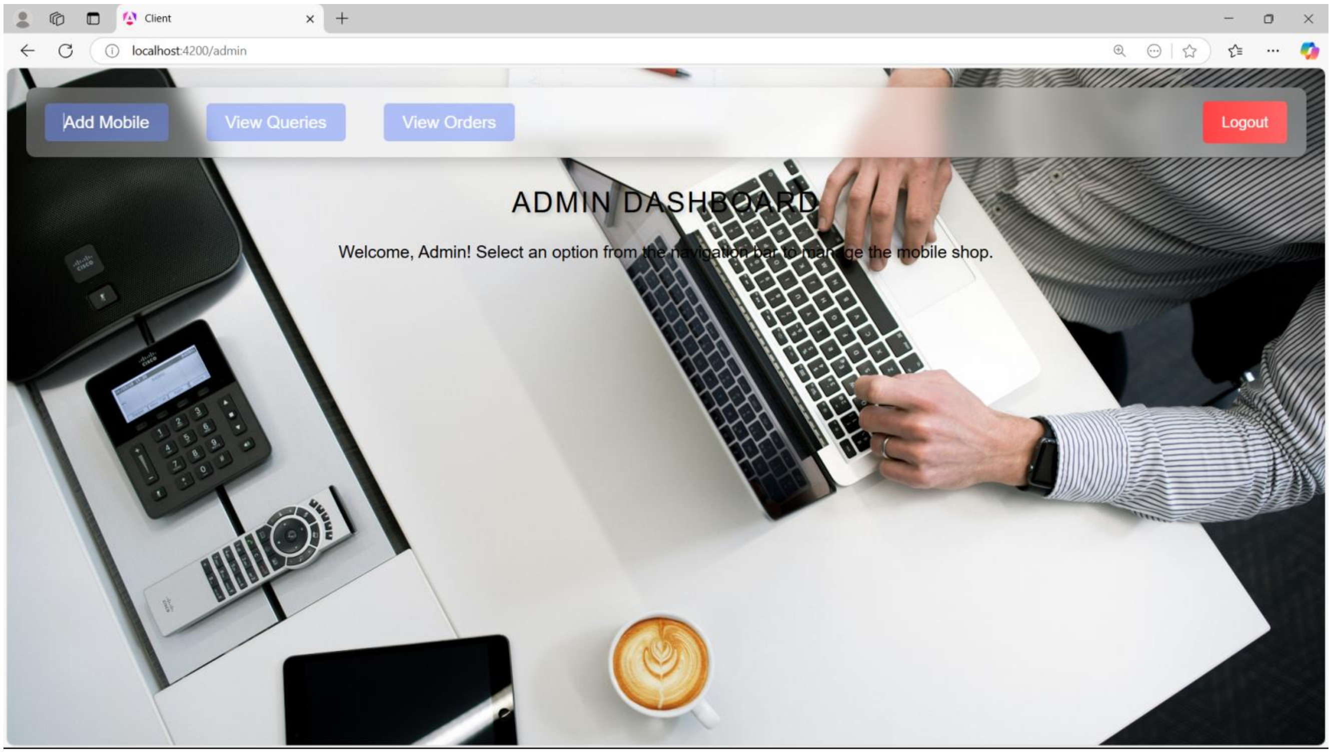


Figure 3.3: Admin Login page

This **Figure 3.3** provides interface about login page for with registered username and password.

3.5.3 Admin dashboard page



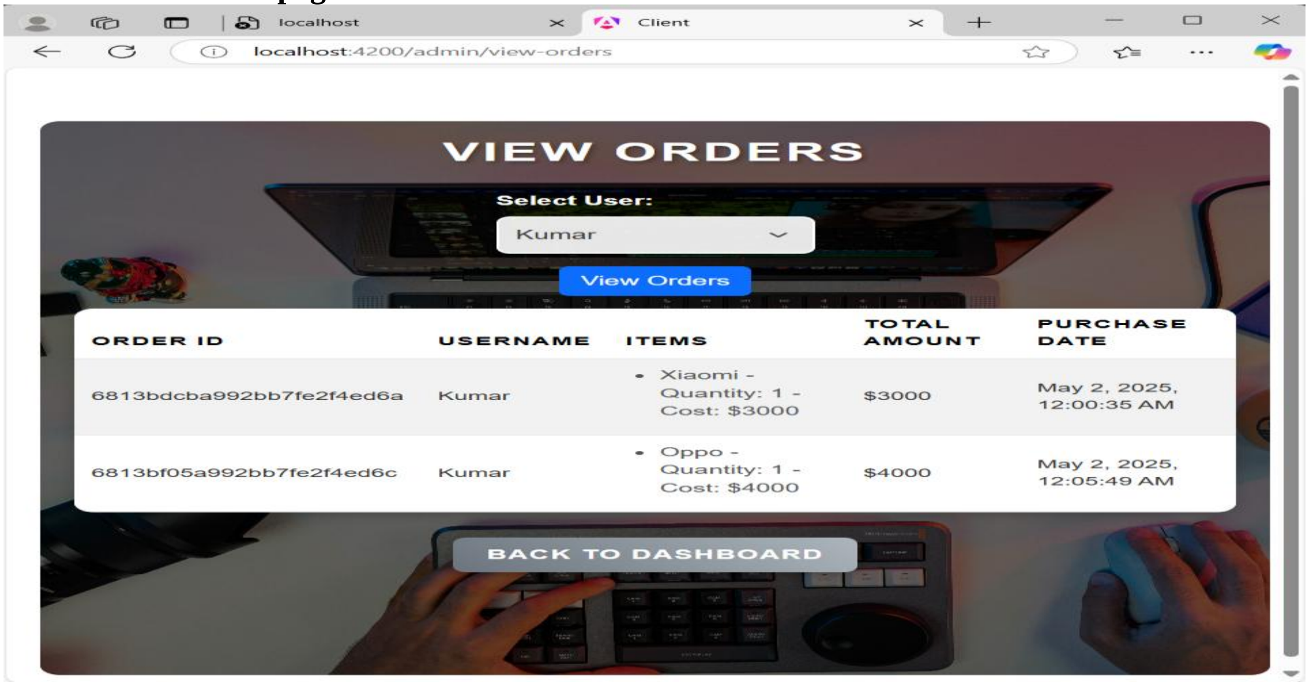
ADMIN DASHBOARD

Welcome, Admin! Select an option from the navigation bar to manage the mobile shop.

Figure 3.4: Admin Dashboard page

This **Figure 3.5** provides interface for the admin to view Recent orders, add mobile items, and view queries.

3.5.4 Recent orders page



VIEW ORDERS

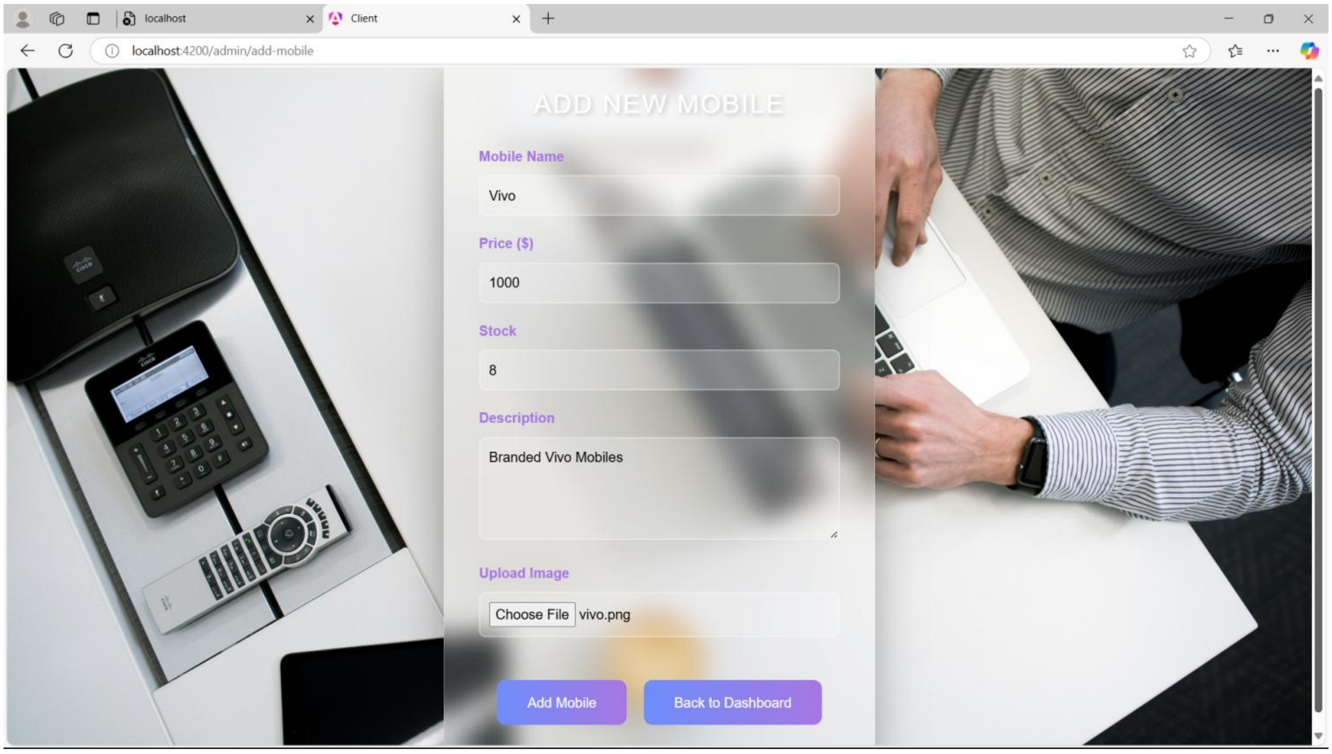
Select User: Kumar

ORDER ID	USERNAME	ITEMS	TOTAL AMOUNT	PURCHASE DATE
6813bdcba992bb7fe2f4ed6a	Kumar	• Xiaomi - Quantity: 1 - Cost: $3000	$3000	May 2, 2025, 12:00:35 AM
6813bf05a992bb7fe2f4ed6c	Kumar	• Oppo - Quantity: 1 - Cost: $4000	$4000	May 2, 2025, 12:05:49 AM

Figure 3.5: Recent orders page

This **Figure 3.5** provides interface for the admin to view Recent orders which consists of Pending order details and status of delivery which can be updated by admin.

3.5.5 Add Mobile Items Page


localhost:4200/admin/add-mobile

ADD NEW MOBILE

Mobile Name: Vivo

Price ($): 1000

Stock: 8

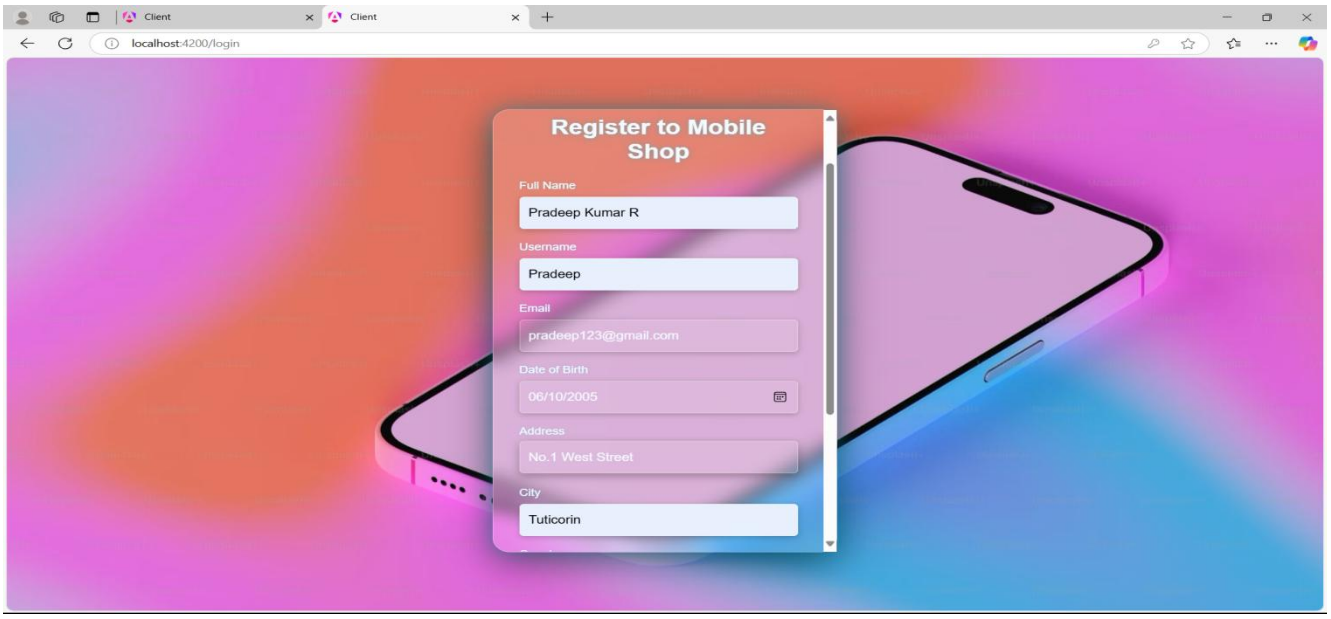
Description: Branded Vivo Mobiles

Upload Image: Choose File vivo.png

Figure 3.6: Add Mobile items page

This **Figure 3.6** provides interface for the admin to add, delete and edit the mobile items to be displayed in user side.

3.5.6 User Sign Up page


localhost:4200/login

Register to Mobile Shop

Full Name: Pradeep Kumar R

Username: Pradeep

Email: pradeep123@gmail.com

Date of Birth: 06/10/2005

Address: No.1 West Street

City: Tuticorin

Figure 3.7: User Sign Up page

This **Figure 3.7,** this interface provides signup page for user, where they can register themselves with their details.

3.5.7 User Login page

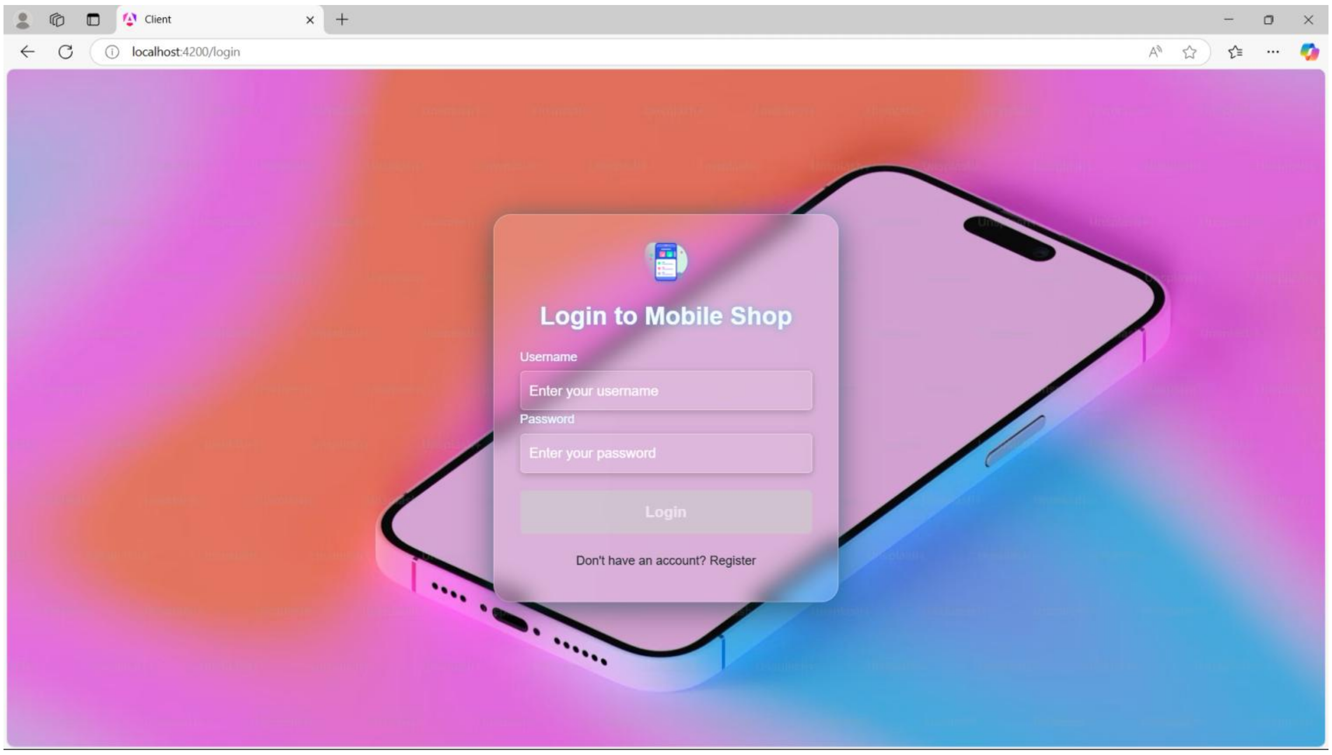


Figure 3.8: User Login page

This **Figure 3.8,** this interface provides login page for users where they can login with their username and password

3.5.8 Order mobile

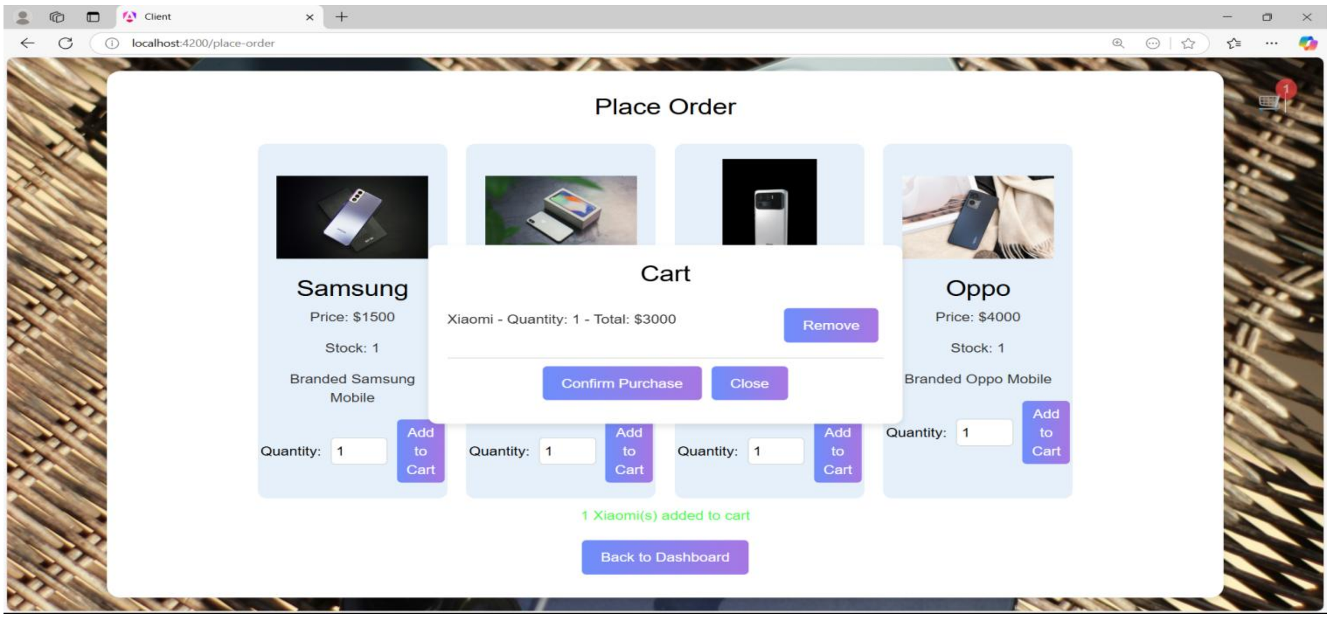
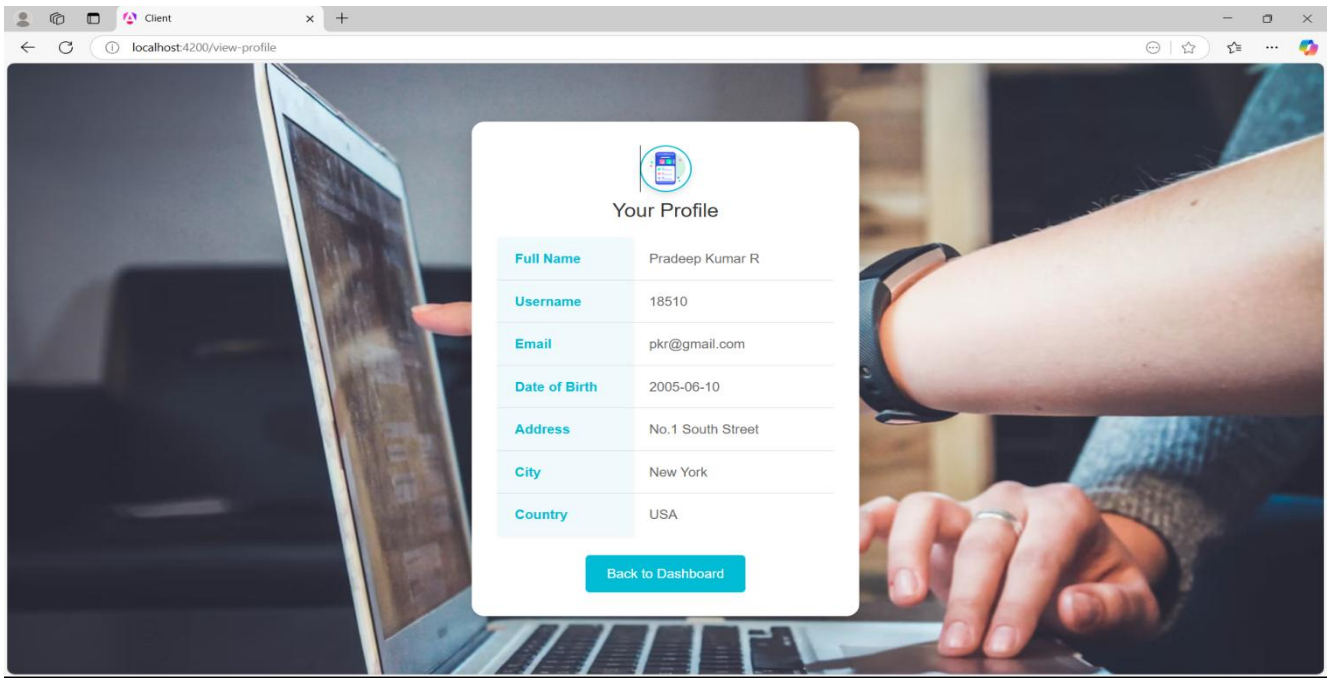


Figure 3.9: Order Mobile page

This **Figure 3.9,** this interface provides order mobile page for users where they can place orders for mobile items available in our application.

3.5.9 View Profile

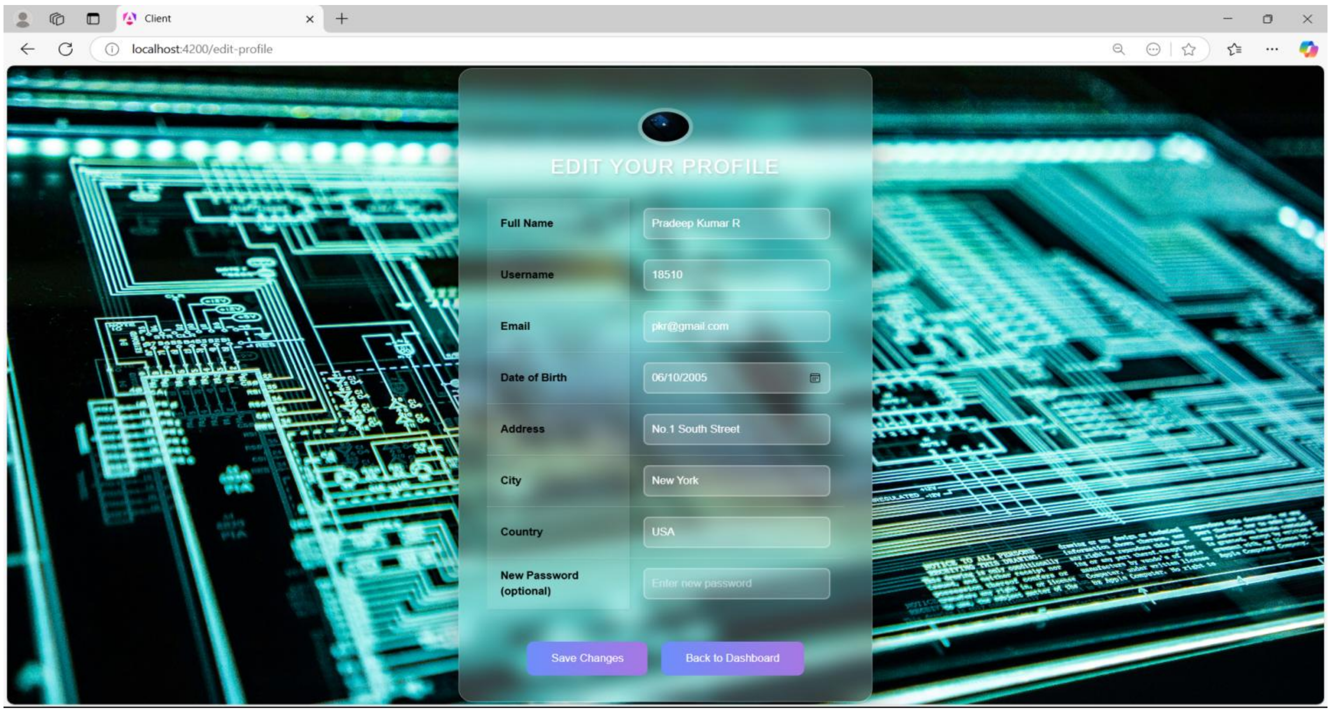


Full Name	Pradeep Kumar R
Username	18510
Email	pkr@gmail.com
Date of Birth	2005-06-10
Address	No.1 South Street
City	New York
Country	USA

Figure 3.10: View Profile page

This **Figure 3.10,** this interface provides View Profile page for users where they can view their details in our application.

3.5.10 Edit Profile

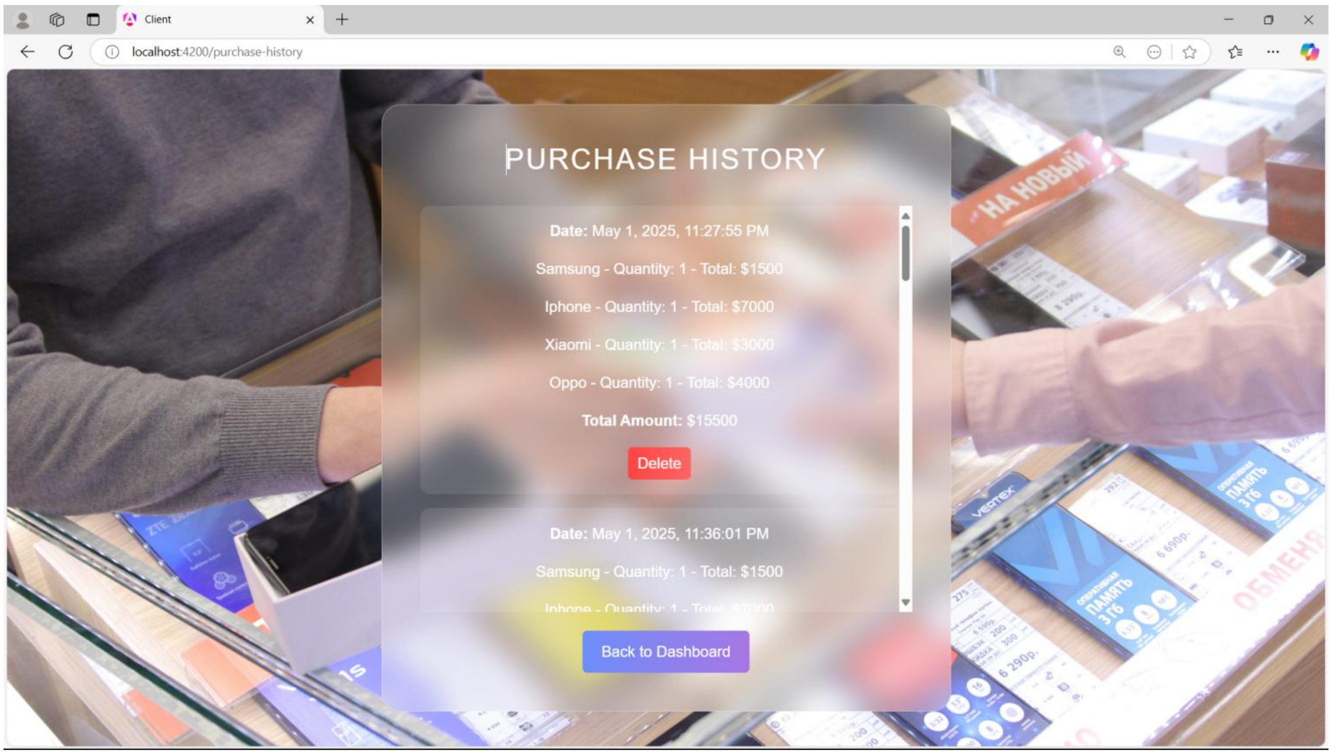


Full Name	Pradeep Kumar R
Username	18510
Email	pkr@gmail.com
Date of Birth	06/10/2005
Address	No.1 South Street
City	New York
Country	USA
New Password (optional)	Enter new password

Figure 3.11: Edit Profile page

This **Figure 3.11,** this interface provides purchase details page for users where they can view their purchase details.

3.5.11 View Purchase History



PURCHASE HISTORY

Date: May 1, 2025, 11:27:55 PM

Samsung - Quantity: 1 - Total: $1500

Iphone - Quantity: 1 - Total: $7000

Xiaomi - Quantity: 1 - Total: $3000

Oppo - Quantity: 1 - Total: $4000

Total Amount: $15500

Date: May 1, 2025, 11:36:01 PM

Samsung - Quantity: 1 - Total: $1500

Iphone - Quantity: 1 - Total: $7000

Figure 3.12: View Purchase History page

This **Figure 3.12,** this interface provides view purchase history page for users where they can view all the purchase made by them.

CHAPTER 4
SYSTEM IMPLEMENTATION

4.1 ADMIN LOGIN IMPLEMENTATION

The admin logs into the page with registered username and password. On successful login, the admin is redirected to admin dashboard page.

```
GET username, password
    IF username, password valid
            RETURN adminpage
    ELSE
            TOAST Invalid Credential
```

4.2 PENDING ORDERS IMPLEMENTATION

The pending orders of the mobiles by users is displayed on this page and the admin updates the status of delivery of the products.

```
GET requestFields
    IF requestFields valid
            RETURN updated details
    ELSE
            TOAST enter valid details
```

4.3 ADD MOBILES IMPLEMENTATION

The mobiles name, quantity and price are added and updated at the admin side, which are displayed to the user.

```
GET requestFields
    IF requestFields valid
            RETURN added/updated details
    ELSE
            TOAST enter valid details
```

4.4 USER REGISTER IMPLEMENTATION

The signup form is to register the user with the application , once the user is registered with all details satisfying validation checks, he can use it for login.

```
GET requestFields
   IF requestFields valid
           RETURN added to db
   ELSE
           TOAST enter valid details
```

4.5 USER LOGIN IMPLEMENTATION

The user logs into the page with registered username and password. On successful login, the user is redirected to user dashboard page to view restaurants and place order of mobile items.

```
GET username, password
   IF username, password valid
           RETURN userpage
   ELSE
           TOAST Invalid Credential
```

4.6 ORDER MOBILE IMPLEMENTATION

The user selects mobile items from the available mobiles and adds that to his/her cart, which is then confirmed and the order is placed

```
GET requestFields
   IF requestFields valid
           RETURN added to db
   ELSE
           TOAST enter valid details
```

CHAPTER 5
RESULTS AND DISCUSSION

5.1 TEST CASES AND RESULTS

5.1.1 Test Cases and Results for Admin Login function:

The **Table 5.1, Table 5.2** shows that the possible test data for the both positive and negative test case given below, if the admin is already having account, then the output is true otherwise false.

Table 5.1: Positive Test Case and result for Admin Login

Test Case ID	TC1
Test Case Description	It tests whether the given login details are valid or not
Test Data	Praveen45,ne%12345
Expected Output	TRUE
Result	PASS

Table 5.2: Negative Test Case and result for Admin Login

Test Case ID	TC2
Test Case Description	It tests whether the given login details are valid or not
Test Data	*(&#}@
Expected Output	FALSE
Result	PASS

5.1.2 Test Cases and Results for Add Mobile items function:

The **Table 5.3, Table 5.4** shows that the possible test data for the both positive and negative test case given below.

Table 5.3: Positive Test Case and result for Add Mobile items

Test Case ID	TC1
Test Case Description	It tests whether the given mobile items details satisfies the constraints
Test Data	VivoV27, 2, Rs.34499
Expected Output	TRUE
Result	PASS

Table 5.4: Negative Test Case and result for Add mobile items

Test Case ID	TC2
Test Case Description	It tests whether the given mobile items details satisfies the constraints
Test Data	OppoF23, -1, Rs.100
Expected Output	FALSE (quantity is negative)
Result	PASS

5.1.3 Test Cases and Results for user login function:

The **Table 5.5, Table 5.6** shows that the possible test data for the both positive and negative test case given below, if the user is already having account, then the output is true otherwise false.

Table 5.5: Positive Test Case and result for user login

Test Case ID	TC1
Test Case Description	It tests whether the given login details are valid or not
Test Data	abcd@gmail.com, 1234@ABC
Expected Output	TRUE
Result	PASS

Table 5.6: Negative Test Case and result for user Login

Test Case ID	TC2
Test Case Description	It tests whether the given login details are valid or not
Test Data	xxx@gmail.com, 333
Expected Output	FALSE
Result	PASS

5.1.4 Test Cases and Results for User Register function:

The **Table 5.7, Table 5.8** shows that the possible test data for the both positive and negative test case given below, if the user is registered with valid userid, password and details, then return true else return false.

Table 5.7: Positive Test Case and result for register

Test Case ID	TC1
Test Case Description	It tests whether the given register details are valid or not
Test Data	Praveen, prvn@gmail.com, 9876512345, Madurai, 1234@qwe, 12342qwe
Expected Output	TRUE
Result	PASS

Table 5.8: Negative Test Case and result for register

Test Case ID	TC2
Test Case Description	It tests whether the given register details are valid or not
Test Data	Password : s@123456rw Confirm password : abcd
Expected Output	FALSE
Result	PASS

5.1.5 Test Cases and Results for Order Mobile items function:

The **Table 5.9, Table 5.10** shows that the possible test data for the both positive and negative test case given below.

Table 5.9: Positive Test Case and result for order mobile items

Test Case ID	TC1
Test Case Description	It tests whether the given mobile items details satisfies the constraints
Test Data	VivoV27, 2, Rs.27499
Expected Output	TRUE
Result	PASS

Table 5.10: Negative Test Case and result for order mobile items

Test Case ID	TC2
Test Case Description	It tests whether the given mobile items details satisfies the constraints
Test Data	OppoF23, -1, Rs.100
Expected Output	FALSE (quantity is negative)
Result	PASS

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT(S)

In conclusion, PhoneNest, our digital mobile selling platform, provides a seamless and efficient experience for both users and store admins. By offering functionalities such as user registration, product browsing, secure purchasing, and inventory management, PhoneNest streamlines the mobile shopping process and enhances user convenience. The platform ensures user satisfaction through features like detailed product catalogs, easy order placement, and transparent purchase history tracking. Moving forward, future enhancements could focus on implementing advanced recommendation algorithms to personalize the shopping experience, integrating real-time delivery tracking for enhanced transparency, and expanding the platform to support additional payment methods for increased flexibility. Additional features such as AI-powered chatbots for customer support can also be explored. With continuous improvements and innovations, PhoneNest aims to redefine the digital mobile retail industry and become the go-to platform for mobile enthusiasts alike.

APPENDIX – A

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENT:

Memory	:	128GB ROM
Processor	:	Core i5 Processor
Disk	:	40 GB Operating system

SOFTWARE REQUIREMENT:

Operating System	:	Windows 11 (64 bit)
DBMS	:	MongoDB (version 6.0.5)
IDE used	:	Visual Studio Code
Markup language	:	HTML
Programming language	:	Typescript
Scripting language	:	CSS
Framework	:	AngularJS (version 17.3.5)

APPENDIX – B

SOURCE CODE

add-mobile.component.ts

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-add-mobile',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule],
  templateUrl: './add-mobile.component.html',
  styleUrls: ['./add-mobile.component.css']
})
export class AddMobileComponent {
  mobileForm: FormGroup;
  errorMessage: string | null = null;
  successMessage: string | null = null;
  selectedFile: File | null = null;

  constructor(
    private fb: FormBuilder,
    private http: HttpClient,
    private router: Router
  ) {
    this.mobileForm = this.fb.group({
      name: ['', Validators.required],
      price: [0, [Validators.required, Validators.min(0)]],
      stock: [0, [Validators.required, Validators.min(0)]],
      description: ['', Validators.required]
    });

    if (localStorage.getItem('isAdmin') !== 'true') {
      this.router.navigate(['/dashboard']);
    }
  }

  onFileSelected(event: Event) {
    const input = event.target as HTMLInputElement;
    if (input.files && input.files.length > 0) {
      this.selectedFile = input.files[0];
      console.log('Selected File:', this.selectedFile.name);
    }
  }
  onSubmit() {
    if (this.mobileForm.valid && this.selectedFile) {
      const formData = new FormData();
      formData.append('name', this.mobileForm.value.name);
      formData.append('price', this.mobileForm.value.price.toString());
      formData.append('stock', this.mobileForm.value.stock.toString());
      formData.append('description', this.mobileForm.value.description);
      formData.append('image', this.selectedFile);

      console.log('Data Being Sent to Database:');
      formData.forEach((value, key) => {
        console.log(`${key}: ${value}`);
      });

      this.http.post('http://localhost:5000/addMobile', formData).subscribe({
        next: (response: any) => {
          this.successMessage = 'Mobile added successfully!';
          this.errorMessage = null;
          this.mobileForm.reset();
          this.selectedFile = null;
          const fileInput = document.getElementById('image') as HTMLInputElement;
          if (fileInput) fileInput.value = '';
          console.log('Server Response:', response);
        },
        error: (err) => {
          this.errorMessage = 'Failed to add mobile: ' + (err.error?.message || 'Unknown error');
          console.error('Error Response:', err);
        }
      });
    } else {
      this.errorMessage = 'Please fill out all required fields and select an image.';
    }
  }

  goBack() {
    this.router.navigate(['/admin']);
  }
}
```

admin.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';

@Component({
  selector: 'app-admin',
  standalone: true,
  imports: [CommonModule, RouterModule],
  templateUrl: './admin.component.html',
  styleUrls: ['./admin.component.css']
})
export class AdminComponent {
  constructor(private router: Router) {
    if (localStorage.getItem('isAdmin') !== 'true') {
      this.router.navigate(['/dashboard']);
    }
  }

  logout() {
    localStorage.removeItem('username');
    localStorage.removeItem('isAdmin');
    this.router.navigate(['/login']);
  }
}
```

dashboard.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-dashboard',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  username: string | null = null;

  constructor(private router: Router) {
    this.username = localStorage.getItem('username');
    if (!this.username) {
      this.router.navigate(['/login']);
    }
  }

  ngOnInit(): void {}
  viewProfile() {
    this.router.navigate(['/view-profile']);
  }

  editProfile() {
    this.router.navigate(['/edit-profile']);
  }

  placeOrder() {
    this.router.navigate(['/place-order']);
  }

  viewPurchaseHistory() {
    this.router.navigate(['/purchase-history']);
  }

  logout() {
    localStorage.removeItem('username');
    this.router.navigate(['/login']);
  }
}
```

edit-profile.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-edit-profile',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule],
  templateUrl: './edit-profile.component.html',
  styleUrls: ['./edit-profile.component.css']
})
export class EditProfileComponent implements OnInit {
  editForm: FormGroup;
  errorMessage: string | null = null;
  successMessage: string | null = null;
  isUsernameDisabled: boolean = true;

  constructor(
    private fb: FormBuilder,
    private http: HttpClient,
    private router: Router
  ) {
```

```
  this.editForm = this.fb.group({
    fullName: ['', Validators.required],
    username: ['', Validators.required],
    email: ['', [Validators.required, Validators.email]],
    dateOfBirth: ['', Validators.required],
    address: ['', Validators.required],
    city: ['', Validators.required],
    country: ['', Validators.required],
    password: ['', [Validators.minLength(6)]]
  });
}

ngOnInit() {
  const username = localStorage.getItem('username');
  if (username) {
    this.http.post('http://localhost:5000/getProfile', { username }).subscribe({
      next: (response: any) => {
        console.log('Profile loaded:', response);
        this.editForm.patchValue({
          fullName: response.fullName || '',
          username: response.username || '',
          email: response.email || '',
          dateOfBirth: response.dob || '',
          address: response.address || '',
          city: response.city || '',
          country: response.country || ''
        });
      },
      error: (err) => {
        console.error('Error loading profile:', err);
        this.errorMessage = 'Failed to load profile: ' + (err.error?.message || 'Unknown error');
      }
    });
  } else {
    this.errorMessage = 'No user logged in';
    this.router.navigate(['/login']);
  }
}

onSubmit() {
  if (this.editForm.valid) {
    const updatedProfile = {
      ...this.editForm.value,
      dob: this.editForm.value.dateOfBirth
    };
    const username = localStorage.getItem('username');
    if (username) {
      updatedProfile.username = username;
      if (!updatedProfile.password) {
        delete updatedProfile.password;
      }
```

```
        this.http.post('http://localhost:5000/updateProfile', updatedProfile).subscribe({
          next: (response: any) => {
            console.log('Profile updated:', response);
            this.successMessage = 'Profile updated successfully!';
            this.errorMessage = null;
            // Removed automatic navigation
          },
          error: (err) => {
            console.error('Error updating profile:', err);
            this.errorMessage = 'Failed to update profile: ' + (err.error?.message || 'Unknown error');
            this.successMessage = null;
          }
        });
      }
    } else {
      this.errorMessage = 'Please fill out all required fields correctly.';
    }
  }

  goBack() {
    this.router.navigate(['/dashboard']);
  }
}
```

login.component.ts

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';
import { ReactiveFormsModule, FormsModule} from '@angular/forms';


@Component({
  selector: 'app-login',
  standalone: true,
  imports: [CommonModule, ReactiveFormsModule,FormsModule],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  loginForm: FormGroup;
  registerForm: FormGroup;
  errorMessage: string | null = null;
  successMessage: string | null = null;
  isLoginMode = true;
```

```
constructor(
  private fb: FormBuilder,
  private http: HttpClient,
  private router: Router
) {
  this.loginForm = this.fb.group({
    username: ['', Validators.required],
    password: ['', Validators.required]
  });

  this.registerForm = this.fb.group({
    fullName: ['', Validators.required],
    username: ['', Validators.required],
    email: ['', [Validators.required, Validators.email]],
    dob: ['', Validators.required],
    address: ['', Validators.required],
    city: ['', Validators.required],
    country: ['', Validators.required],
    password: ['', [Validators.required, Validators.minLength(6)]]
  });
}

toggleMode() {
  this.isLoginMode = !this.isLoginMode;
  this.errorMessage = null;
  this.successMessage = null;
}

onSubmit() {
  if (this.isLoginMode) {
    this.login();
  } else {
    this.register();
  }
}

private login() {
  if (this.loginForm.valid) {
    const { username, password } = this.loginForm.value;
    if (username === 'Admin' && password === '12345678') {
      this.successMessage = 'Admin login successful';
      this.errorMessage = null;
      localStorage.setItem('username', username);
      localStorage.setItem('isAdmin', 'true');
      this.router.navigate(['/admin']);
    } else {
      this.http.post('http://localhost:5000/login', this.loginForm.value).subscribe(
        (response: any) => {
          this.successMessage = response.message;
          this.errorMessage = null;
          localStorage.setItem('username', this.loginForm.value.username);
          localStorage.setItem('isAdmin', 'false');
          this.router.navigate(['/dashboard']);
        },
        (err) => {
          this.errorMessage = err.error.message || 'Login failed';
          this.successMessage = null;
        }
      );
    }
  }
}

  private register() {
    if (this.registerForm.valid) {
      this.http.post('http://localhost:5000/register', this.registerForm.value).subscribe(
        (response: any) => {
          this.successMessage = response.message;
          this.errorMessage = null;
          this.registerForm.reset();
          this.toggleMode();
        },
        (err) => {
          this.errorMessage = err.error.message || 'Registration failed';
          this.successMessage = null;
        }
      );
    }
  }
}
```

place-order.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import jsPDF from 'jspdf';
import 'jspdf-autotable';

@Component({
  selector: 'app-place-order',
  standalone: true,
  imports: [CommonModule, FormsModule],
  templateUrl: './place-order.component.html',
  styleUrls: ['./place-order.component.css']
})
export class PlaceOrderComponent implements OnInit {
  mobiles: any[] = [];
  cart: { mobile: any, quantity: number, totalCost: number }[] = [];
  errorMessage: string | null = null;
  successMessage: string | null = null;
  username: string | null = null;
  showCart: boolean = false;

  constructor(private http: HttpClient, private router: Router) {
    this.username = localStorage.getItem('username');
    if (!this.username) {
      console.log('No username found in localStorage, redirecting to login');
      this.router.navigate(['/login']);
    }
  }

  ngOnInit(): void {
    console.log('ngOnInit called, username:', this.username);
    this.loadMobiles();
    this.loadCart();
  }

  loadMobiles() {
    this.http.get('http://localhost:5000/getMobiles').subscribe({
      next: (response: any) => {
        this.mobiles = response.mobiles.map((mobile: any) => ({
          ...mobile,
          imageUrl: mobile.imageUrl.replace('/src', '')
        }));
        console.log('Mobiles loaded:', this.mobiles);
      },
      error: (err) => {
        this.errorMessage = 'Failed to load mobiles: ' + (err.error?.message || 'Unknown error');
        console.error('Error loading mobiles:', err);
      }
    });
  }

  loadCart() {
    if (!this.username) {
      console.log('No username found, cannot load cart');
      this.cart = [];
      return;
    }
    console.log('Loading cart for username:', this.username);
    this.http.post('http://localhost:5000/getCart', { username: this.username }).subscribe({
      next: (response: any) => {
        this.cart = response.items || [];
        console.log('Cart loaded successfully:', this.cart);
      },
      error: (err) => {
        this.errorMessage = 'Failed to load cart: ' + (err.error?.message || 'Unknown error. Initializing
empty cart.');
```

```
        this.cart = [];
        console.error('Error loading cart:', err);
      }
    });
  }

  saveCart() {
    if (!this.username) {
      console.log('No username found, cannot save cart');
      return;
    }
    console.log('Saving cart for username:', this.username, 'with items:', this.cart);
    this.http.post('http://localhost:5000/saveCart', { username: this.username, items: this.cart
}).subscribe({
      next: () => {
        console.log('Cart saved successfully');
      },
      error: (err) => {
        this.errorMessage = 'Failed to save cart: ' + (err.error?.message || 'Unknown error');
        console.error('Error saving cart:', err);
      }
    });
  }

  addToCart(mobile: any, quantityInput: HTMLInputElement) {
    if (!quantityInput) {
      this.errorMessage = 'Quantity input not found';
      console.log('Quantity input element not found');
      return;
    }

    const quantity = parseInt(quantityInput.value, 10);
    if (isNaN(quantity) || quantity <= 0) {
      this.errorMessage = 'Please enter a valid quantity';
      console.log('Invalid quantity entered:', quantityInput.value);
      return;
    }

    if (quantity > mobile.stock) {
      this.errorMessage = `Only ${mobile.stock} units of ${mobile.name} are available`;
      console.log('Quantity exceeds stock:', quantity, 'Stock:', mobile.stock);
      return;
    }

    const existingItem = this.cart.find(item => item.mobile._id === mobile._id);
    if (existingItem) {
      existingItem.quantity += quantity;
      existingItem.totalCost = existingItem.quantity * mobile.price;
    } else {
      this.cart.push({
        mobile,
        quantity,
        totalCost: quantity * mobile.price
      });
    }

    this.errorMessage = null;
    this.successMessage = `${quantity} ${mobile.name}(s) added to cart`;
    quantityInput.value = '1';
    console.log('Cart after adding item:', this.cart);
    this.saveCart();
  }

  toggleCart() {
    this.showCart = !this.showCart;
    console.log('Cart toggled, showCart:', this.showCart);
  }

  removeFromCart(index: number) {
    console.log('Removing item at index:', index);
    this.cart.splice(index, 1);
    this.saveCart();
  }

  confirmPurchase() {
    if (this.cart.length === 0) {
      this.errorMessage = 'Cart is empty';
      console.log('Attempted to confirm purchase with empty cart');
      return;
    }

    // Re-validate username
    this.username = localStorage.getItem('username');
    if (!this.username) {
      console.log('Username not found, redirecting to login');
      this.errorMessage = 'Session expired. Please log in again.';
      this.router.navigate(['/login']);
      return;
    }

    const orderData = {
      username: this.username,
      items: this.cart.map(item => ({
        mobileId: item.mobile._id,
        mobileName: item.mobile.name,
        quantity: item.quantity,
        price: item.mobile.price,
        totalCost: item.totalCost
      })),
      orderDate: new Date().toISOString()
    };
```

```
    console.log('Confirming purchase with data:', orderData);
    this.http.post('http://localhost:5000/placeOrder', orderData).subscribe({
      next: (response: any) => {
        this.successMessage = 'Order placed successfully! PDF will be generated shortly.';
        this.errorMessage = null;
        this.cart = [];
        this.loadMobiles();
        this.showCart = false;
        this.saveCart();
        console.log('Purchase confirmed, raw response:', response);
        console.log('Purchase confirmed, response.purchaseId:', response.purchaseId);

        const purchaseId = response.purchaseId;
        if (!purchaseId) {
          console.error('Purchase ID not returned in response:', JSON.stringify(response, null, 2));
          this.errorMessage = 'Failed to generate PDF: Purchase ID missing';
          return;
        }

        this.username = localStorage.getItem('username');
        if (!this.username) {
          console.error('Username not found during PDF generation');
          this.errorMessage = 'Failed to generate PDF: Username missing';
          return;
        }

        setTimeout(() => {
          console.log('Fetching purchase details to generate PDF with purchaseId (_id):', purchaseId);
          this.fetchPurchaseAndGeneratePDF(purchaseId);
        }, 3000);
      },
      error: (err) => {
        this.errorMessage = 'Failed to place order: ' + (err.error?.message || 'Unknown error');
        console.error('Error placing order:', err);
        if (err.status === 401) {
          console.log('Unauthorized error, redirecting to login');
          this.errorMessage = 'Session expired or unauthorized. Please log in again.';
          this.router.navigate(['/login']);
        }
      }
    });
  }

  fetchPurchaseAndGeneratePDF(purchaseId: string) {
    if (!this.username || !purchaseId) {
      console.error('Cannot fetch purchase: Missing username or purchaseId', { username: this.username,
purchaseId });
      this.errorMessage = 'Cannot generate PDF: Missing username or purchase ID';
      return;
    }
    console.log('Fetching purchase for username:', this.username, 'and purchaseId (_id):', purchaseId);
    this.http.post('http://localhost:5000/getPurchaseById', { username: this.username, purchaseId
}).subscribe({
      next: (purchase: any) => {
        console.log('Purchase details fetched:', purchase);
        this.generatePurchasePDF(purchase);
      },
      error: (err) => {
        this.errorMessage = 'Failed to fetch purchase details: ' + (err.error?.message || 'Unknown error');
        console.error('Error fetching purchase details:', err);
      }
    });
  }

  generatePurchasePDF(purchase: any) {
    const doc = new jsPDF();

    // Add title
    doc.setFontSize(20);
    doc.text('Purchase Receipt', 105, 20, { align: 'center' });

    // Add date
    doc.setFontSize(12);
    const purchaseDate = new Date(purchase.purchaseDate).toLocaleDateString();
    doc.text(`Date: ${purchaseDate}`, 14, 40);

    // Prepare table data
    const tableData = purchase.items.map((item: any) => [
      item.mobileName,
      item.quantity.toString(),
      `$${item.price}`,
      `$${item.totalCost}`
    ]);

    // Add table using autoTable
    (doc as any).autoTable({
      startY: 50,
      head: [['Mobile Name', 'Quantity', 'Price per Unit', 'Total']],
      body: tableData,
      theme: 'striped',
      headStyles: {
        fillColor: [110, 142, 251], // #6e8efb
        textColor: [255, 255, 255], // #ffffff
      },
      bodyStyles: {
        fillColor: [230, 240, 250], // #e6f0fa
        textColor: [0, 0, 0], // #000000
      },
      alternateRowStyles: {
        fillColor: [255, 255, 255] // White for alternate rows
      },
      margin: { top: 50 }
    });

    // Add total amount
    const finalY = (doc as any).lastAutoTable.finalY || 50;
    doc.text(`Total Amount: $${purchase.totalAmount}`, 200, finalY + 10, { align: 'right' });

    // Save the PDF
    doc.save('purchase_receipt.pdf');
    console.log('PDF generated and downloaded successfully using jsPDF');
  }

  goBack() {
    this.router.navigate(['/dashboard']);
    console.log('Navigating back to dashboard');
  }
}
```

purchase-history.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-purchase-history',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './purchase-history.component.html',
  styleUrls: ['./purchase-history.component.css']
})
export class PurchaseHistoryComponent implements OnInit {
  purchases: any[] = [];
  errorMessage: string | null = null;
  username: string | null = null;

  constructor(private http: HttpClient, private router: Router) {
    this.username = localStorage.getItem('username');
    if (!this.username) {
      this.router.navigate(['/login']);
    }
  }

  ngOnInit(): void {
    this.loadPurchases();
  }

  loadPurchases() {
    this.http.post('http://localhost:5000/getPurchases', { username: this.username }).subscribe({
      next: (response: any) => {
        this.purchases = response.purchases;
      },
      error: (err) => {
        this.errorMessage = 'Failed to load purchases: ' + (err.error?.message || 'Unknown error');
      }
    });
  }

  deletePurchase(purchaseId: string) {
    this.http.post('http://localhost:5000/deletePurchase', { username: this.username, purchaseId
}).subscribe({
      next: (response: any) => {
        this.loadPurchases(); // Refresh the list
      },
      error: (err) => {
        this.errorMessage = 'Failed to delete purchase: ' + (err.error?.message || 'Unknown error');
      }
    });
  }

  goBack() {
    this.router.navigate(['/dashboard']);
  }
}
```

view-orders.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-view-orders',
  standalone: true,
  imports: [CommonModule, FormsModule],
  templateUrl: './view-orders.component.html',
  styleUrls: ['./view-orders.component.css']
})
export class ViewOrdersComponent implements OnInit {
  usernames: string[] = [];
  selectedUsername: string = '';
  orders: any[] = [];
  errorMessage: string | null = null;
  adminUsername: string | null = null;
  isAdmin: boolean = false;

  constructor(private http: HttpClient, private router: Router) {
    this.adminUsername = localStorage.getItem('username');
    this.isAdmin = localStorage.getItem('isAdmin') === 'true';
    console.log('Initialized with adminUsername:', this.adminUsername, 'isAdmin:', this.isAdmin);
    if (!this.adminUsername || !this.isAdmin) {
      console.log('Unauthorized access, redirecting to login');
      this.router.navigate(['/login']);
    }
  }

  ngOnInit(): void {
    this.loadUsernames();
  }

  loadUsernames() {
    if (!this.adminUsername) {
      this.errorMessage = 'Admin username not found in localStorage';
      console.error('Admin username not found:', this.adminUsername);
      return;
    }

    console.log('Sending request to load usernames with adminUsername:', this.adminUsername);
    this.http.post('http://localhost:5000/getAllUsernames', { adminUsername: this.adminUsername
}).subscribe({
      next: (response: any) => {
        console.log('Received response from getAllUsernames:', response);
        this.usernames = response.usernames || [];
        if (this.usernames.length === 0) {
          console.log('No usernames returned in response');
        }
        console.log('Usernames loaded:', this.usernames);
      },
      error: (err) => {
        this.errorMessage = 'Failed to load usernames: ' + (err.error?.message || 'Unknown error');
        console.error('Error loading usernames:', err, 'Status:', err.status, 'Error Body:', err.error);
      }
    });
  }

  viewOrders() {
    if (!this.selectedUsername) {
      this.errorMessage = 'Please select a username';
      this.orders = [];
      console.log('No username selected');
      return;
    }

    if (!this.adminUsername) {
      this.errorMessage = 'Admin username not found';
      this.orders = [];
      console.log('Admin username missing');
      return;
    }

    console.log('Fetching orders for username:', this.selectedUsername, 'with adminUsername:',
this.adminUsername);
    this.http.post('http://localhost:5000/getOrdersByUsername', { adminUsername:
this.adminUsername, username: this.selectedUsername }).subscribe({
      next: (response: any) => {
        console.log('Received response from getOrdersByUsername:', response);
        this.orders = response.orders || [];
        console.log('Orders loaded:', this.orders);
        if (this.orders.length === 0) {
          this.errorMessage = `No orders found for ${this.selectedUsername}`;
          console.log(`No orders found for username: ${this.selectedUsername}`);
        } else {
          this.errorMessage = null; // Clear the message if orders are found
        }
      },
      error: (err) => {
        this.errorMessage = 'Failed to load orders: ' + (err.error?.message || 'Unknown error');
        this.orders = [];
        console.error('Error loading orders:', err, 'Status:', err.status, 'Error Body:', err.error);
      }
    });
  }

  goBack() {
    console.log('Navigating back to admin dashboard. Current localStorage - username:',
localStorage.getItem('username'), 'isAdmin:', localStorage.getItem('isAdmin'));
    this.router.navigate(['/admin']);
  }
}
```

view-profile.component.ts

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { CommonModule } from '@angular/common';
import { Router } from '@angular/router';

@Component({
 selector: 'app-view-profile',
 standalone: true,
 imports: [CommonModule],
 templateUrl: './view-profile.component.html',
 styleUrls: ['./view-profile.component.css']
})
export class ViewProfileComponent implements OnInit, OnDestroy {
 userProfile: any = null;
 errorMessage: string | null = null;

 constructor(private http: HttpClient, private router: Router) {
   console.log('ViewProfileComponent constructed');
 }
```

```
  ngOnInit() {
    console.log('ViewProfileComponent ngOnInit started');
    const username = localStorage.getItem('username');
    if (username) {
      console.log('Fetching profile for username:', username);
      this.http.post('http://localhost:5000/getProfile', { username }).subscribe({
        next: (response: any) => {
          console.log('Profile fetch successful, response:', response);
          this.userProfile = {
            ...response,
            dateOfBirth: response.dob || null // Map dob to dateOfBirth, default to null if undefined
          };
        },
        error: (err) => {
          console.error('Profile fetch error:', err);
          this.errorMessage = 'Failed to fetch profile: ' + (err.error?.message || 'Unknown error');
          this.userProfile = null;
        }
      });
    } else {
      console.log('No username in localStorage');
      this.errorMessage = 'No username available. Please log in.';
      this.userProfile = null;
    }
  }

  ngOnDestroy() {
    console.log('ViewProfileComponent destroyed');
  }

  goBack() {
    console.log('Navigating back to dashboard');
    this.router.navigate(['/dashboard']);
  }
}
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { authGuard } from './auth.guard';

export const routes: Routes = [
  { path: 'login', loadComponent: () => import('./login/login.component').then(m =>
m.LoginComponent) },
  { path: 'dashboard', loadComponent: () => import('./dashboard/dashboard.component').then(m =>
m.DashboardComponent), canActivate: [authGuard] },
  { path: 'view-profile', loadComponent: () => import('./view-profile/view-profile.component').then(m
=> m.ViewProfileComponent), canActivate: [authGuard] },
  { path: 'edit-profile', loadComponent: () => import('./edit-profile/edit-profile.component').then(m =>
m.EditProfileComponent), canActivate: [authGuard] },
  { path: 'admin', loadComponent: () => import('./admin/admin.component').then(m =>
m.AdminComponent), canActivate: [authGuard] },
  { path: 'admin/add-mobile', loadComponent: () => import('./add-mobile/add-
mobile.component').then(m => m.AddMobileComponent), canActivate: [authGuard] },
  { path: 'admin/view-orders', loadComponent: () => import('./view-orders/view-
orders.component').then(m => m.ViewOrdersComponent), canActivate: [authGuard] },
  { path: 'place-order', loadComponent: () => import('./place-order/place-order.component').then(m =>
m.PlaceOrderComponent), canActivate: [authGuard] },
  { path: 'purchase-history', loadComponent: () => import('./purchase-history/purchase-
history.component').then(m => m.PurchaseHistoryComponent), canActivate: [authGuard] },
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: '**', loadComponent: () => import('./not-found.component').then(m =>
m.NotFoundComponent) }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

auth.guard.ts

```
import { inject } from '@angular/core';
import { CanActivateFn, Router } from '@angular/router';

export const authGuard: CanActivateFn = (route, state) => {
  const router = inject(Router);
  const username = localStorage.getItem('username');
  const isAdmin = localStorage.getItem('isAdmin') === 'true';

  console.log('AuthGuard triggered for route:', route.url, 'username:', username, 'isAdmin:', isAdmin);

  if (!username) {
    console.log('No username found in localStorage, redirecting to login');
    router.navigate(['/login']);
    return false;
  }

  // Allow access to admin routes only if user is an admin
  const isAdminRoute = route.url.some(segment => segment.path === 'admin');
  if (isAdminRoute && !isAdmin) {
    console.log('User is not an admin, redirecting to dashboard');
    router.navigate(['/dashboard']);
    return false;
  }

  return true;
};
```

REFERENCES

1. Node js:https://www.geeksforgeeks.org/how-to-build-online-mobile-order-system - using-node-js/
2. Crud operations using mean: https://www.dotnettricks.com/learn/angularmaterial/dataTable-crud-operations-mean-stack#:~:text=Let%20me%20explain%20that%20here,of%20employees%20in%20the%20Table.
3. Html and css: https://www.positronx.io/mean-stack-tutorial-angular-crud/