LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

CS5003NI Data Structure and Specialist Programming

AY: Academic Year 2024/25

30 Credit Year Long Module

30% Individual Coursework 02

Submission: Milestone 2

Student Name: Piyush Karn

Project Title: Bake-Ease

London Met ID: 23048660

College ID: NP01AI4A230119

Assignment Due Date: Monday, May 19, 2025

Assignment Submission Date: Monday, May 19, 2025

Submitted to: Mr. Prithivi Maharjan

GitHub Link	https://github.com/piiyuushh/Bake_Ease

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

23048660 Piyush Karn.docx

Islington College,Nepal

Document Details

Submission ID
trn:oid:::3618:96602289

Submission Date
May 19, 2025, 8:54 AM GMT+5:45

Download Date
May 19, 2025, 8:56 AM GMT+5:45

File Name
23048660 Piyush Karn.docx

File Size
54.4 KB

52 Pages

9,015 Words

46,366 Characters

15% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **118** Not Cited or Quoted 14%
  Matches with neither in-text citation nor quotation marks
- **3** Missing Quotations 1%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

2% Internet sources
0% Publications
15% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- 118 Not Cited or Quoted 14%
  Matches with neither in-text citation nor quotation marks
- 3 Missing Quotations 1%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

2% Internet sources
0% Publications
15% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	
islingtoncollege on 2025-01-10		3%
2	Submitted works	
islingtoncollege on 2025-01-10		<1%
3	Submitted works	
Asia Pacific International College on 2021-12-17		<1%
4	Submitted works	
islingtoncollege on 2025-01-22		<1%
5	Submitted works	
ESoft Metro Campus, Sri Lanka on 2025-04-22		<1%
6	Submitted works	
University of Wales Institute, Cardiff on 2024-09-27		<1%
7	Internet	
www.coursehero.com		<1%
8	Submitted works	
Bangkok Patana School on 2012-03-15		<1%
9	Submitted works	
Saint Francis College on 2025-04-04		<1%
10	Submitted works	
University of Wales Institute, Cardiff on 2024-09-30		<1%

Abstract

The **Bake Ease** bakery management web application is a dynamic Java-based system developed using Java Server Pages (JSP), Servlets, and MySQL. It is designed to streamline and modernize bakery operations by providing a user-friendly platform for product management, order processing, customer feedback, and inventory updates. Developed using Eclipse IDE and deployed via Apache Tomcat Server, the application ensures smooth integration between frontend and backend components. Database design follows normalization principles, ensuring data consistency and minimizing redundancy. With features like real-time order tracking, a comprehensive admin dashboard, and secure authentication for both customers and administrators, Bake Ease aims to enhance user experience and operational efficiency for bakeries operating in the digital space. This open-source project is scalable and adaptable, encouraging future development and community contributions.

Acknowledgement

I would like to express our sincere gratitude to all my fellow friends and respected tutor who contributed to the successful completion of this project. First and foremost, I thank our tutors and instructors for their invaluable guidance, constructive feedback, and continuous support throughout the development of this system.

My heartfelt thanks go to my friends and peers who provided motivation and feedback during the design and testing phases. I also acknowledge the online resources, reference articles, and platforms like Canva for wireframing, GitHub for code management, and MySQL via XAMPP for database management, which played a vital role in the development process.

Lastly, I would like to extend my gratitude to our college community for their encouragement and support throughout the development of this project. This project is a small step toward digitizing local bakery businesses, and we hope it serves as a useful and evolving tool in the days ahead.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Bake Ease, a bakery management web application built using Java service pages package. This project specializes in management of bakery related operations for bakeries operating online for customer convenience.

1.1. Overview and Purpose

The project is built using Eclipse IDE, for designing we have used Canva for wireframe development and various references articles for research purpose. To design the actual web components JSP files and core CSS files are used inside the project. For developing database, we have used MySQL server through Xampp. To maintain the dynamicity of the project, special care has been taken care, and many things have been taken into consideration in frontend, backend and database integration. The source code for this project has been published on GitHub as a repository for user accessibility.

1.2. Audience

The project is made for bakery service providers who specialize in building online forum for order management and customer order and feedback.

1.3. Aims, Objectives and Features

1.3.1. Aim

The aim of this project this to provide user convenience as per their need and maintain the bakery management system easy and comfortable for both user and the bakery owner and since this project is opensource so in future anyone from the internet can help to make it better.

1.3.2. Objectives

- To make the bakery management system convenient
- To make user experience better and better
- Streamline inventory operations
- Real time updates

1.3.3. Features

The features of this bakery management system web application are:

- **Admin dashboard**: the dashboard will allow admin to access bakery items and edit information of the items in the bakery and perform CRUD operation.
- **Products Page**: this option allows user to see all the currently available products and make their order decision.
- **Order Page**: this is the page where the user can make their order about what they need and that order will be delivered.
- **Authentication**: Login/Signup page for admin and customers.
- **Contact/Feedback**: This page will ask customer about their feedback and their user experience.

2. Database Design and Development

2.1. Normalization

Normalization is the process of structuring data in a database. It involves creating tables and defining relationships between them based on rules that safeguard the data and enhance the database's flexibility by reducing redundancy and preventing inconsistent dependencies. (Ahmed, n.d.)

2.1.1. Unnormalized form (UNF)

In database normalization, **Unnormalized Form (UNF)** refers to a table that contains redundant data, such as repeating groups or multi-valued attributes. This form does not adhere to any normalization rules and serves as the starting point for the normalization process, which aims to organize data to reduce redundancy and improve integrity. (Tonyloi, n.d.)

Here in the context of this project the **unnormalized form (UNF)** looks somewhat like this. **OrderID, CustomerName, Phone, ProductName, Category, Quantity, Price, OrderDate, Address**

This is the unnormalized form which will be further discussed and improvised in the further steps.

2.1.2. First Normal Form (1NF)

In database normalization, **First Normal Form (1NF)** is a property of a relation in a relational database. A relation is in first normal form if and only if no attribute domain has relations as elements. Or more informally, that no table column can have tables as values. (Wikipedia, n.d.)

The **First Normal Form (1NF)** of the system looks like this:

User id, username, email, password, role

Table: Orders

OrderID, CustomerID, OrderDate

Table: OrderDetails

OrderDetailID, OrderID, ProductName, Category, Quantity, Price

2.1.3. Second Normal Form (2NF)

Second normal form is a database normalization step focused on eliminating partial dependencies. It was introduced by Edgar F. Codd, the pioneer of relational databases, as part of his work on normalization. (Fayard, n.d.)

The Second Normal Form of the system looks like this:

Table: Products

ProductID, ProductName, Category, Price

Table: OrderDetails (Updated)

OrderDetailID, OrderID, ProductID, Quantity

2.1.4. Third Normal Form (3NF)

When a table is in **2NF**, it eliminates repeating groups and redundancy, but it does not eliminate transitive partial dependency. This means a non-prime attribute is dependent on another non-prime attribute. This is what the **third normal form (3NF)** eliminates. (Chris, n.d.)

According to previous normal forms, the third normal form becomes:

Table: Categories

CategoryID, CategoryName

Table: Products (Updated)

ProductID, ProductName, CategoryID, Price

2.1.5. Final Table

1. **users**
    - user id
    - username
    - email

- password
- phone
- role
- image_path

2. **Orders**
    - OrderID
    - CustomerID
    - OrderDate
    - OrderDetails
3. **OrderDetails**
    - OrderDetailID
    - OrderID
    - ProductID
    - Quantity
4. **Products**
    - ProductID
    - ProductName
    - Price
    - Category
    - Total sales
5. **Categories**
    - CategoryID
    - CategoryName

2.2. Entity Relationship Diagram

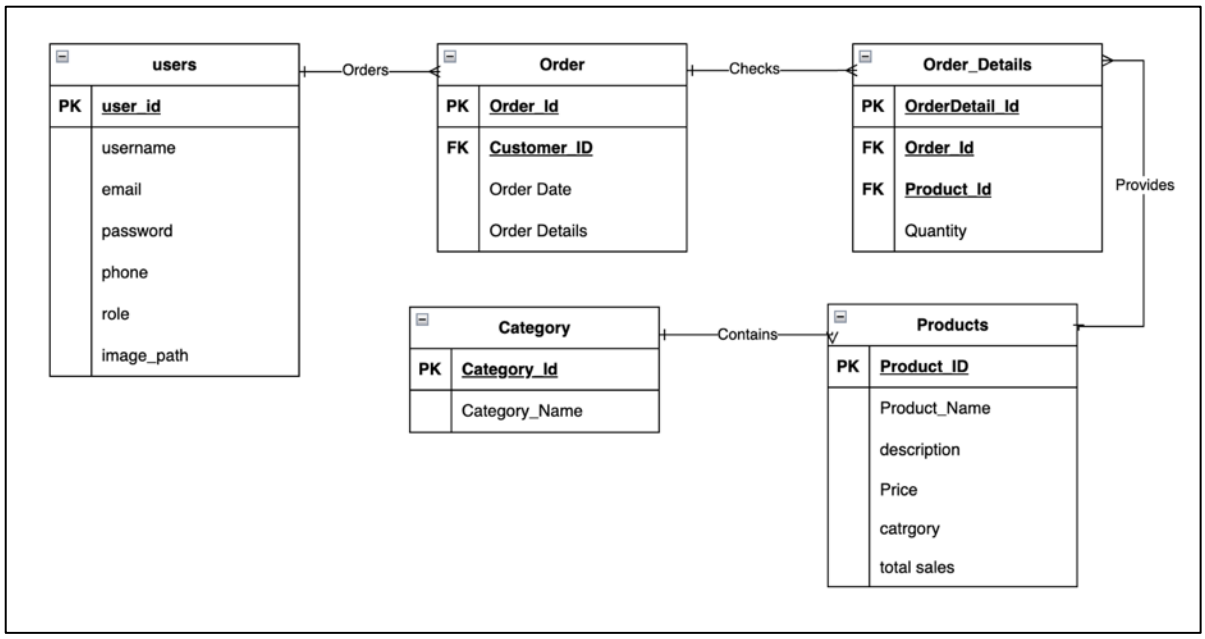


Figure 1: Entity Relationship diagram

2.3. Database Development

The Bakery Management System uses **MySQL** as its primary database management system to store and manage all application-related data such as users, products, orders, and feedback. MySQL offers high performance, data integrity, and seamless integration with Java technologies via **JDBC (Java Database Connectivity)**, making it ideal for dynamic web applications. Using **Eclipse IDE** and deploying through the **Apache Tomcat Server**, our system connects the frontend (JSP, HTML, CSS) and backend (Servlets, JDBC) to perform real-time database operations.

The SQL database is designed following the principles of **normalization** to minimize redundancy and ensure data consistency. The database schema consists of multiple relational tables such as:

- **users** – stores user data
- **products** – holds item details like name, category, price, etc.
- **orders** – records user orders linked to customers
- **order_details** – holds product-quantity info for each order
- **categories** – organizes products

The **authentication system** utilizes separate tables for customers and admins, allowing secure login and registration features. Using **JSP forms**, data is sent to **Servlets**, which use **JDBC** to interact with the MySQL database and execute **CRUD operations**. The **admin dashboard** provides features like adding new bakery items, updating product information, and deleting outdated records. On the customer side, users can browse available products through the **product catalog**, place orders from the **order page**, and submit reviews using the **feedback form**.

This structured database-driven approach ensures smooth communication between all layers of the application, enhances scalability, and provides a reliable platform for managing bakery operations digitally.

2.3.1. Table 1: Users

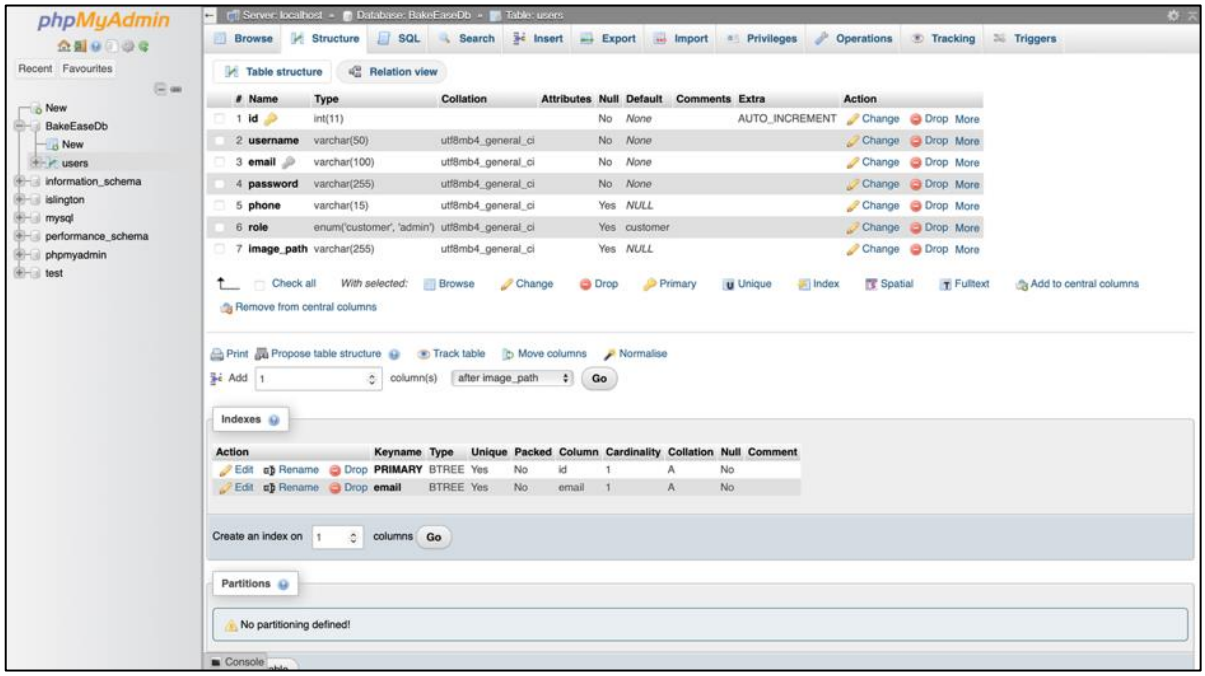


Figure 2: User table structure

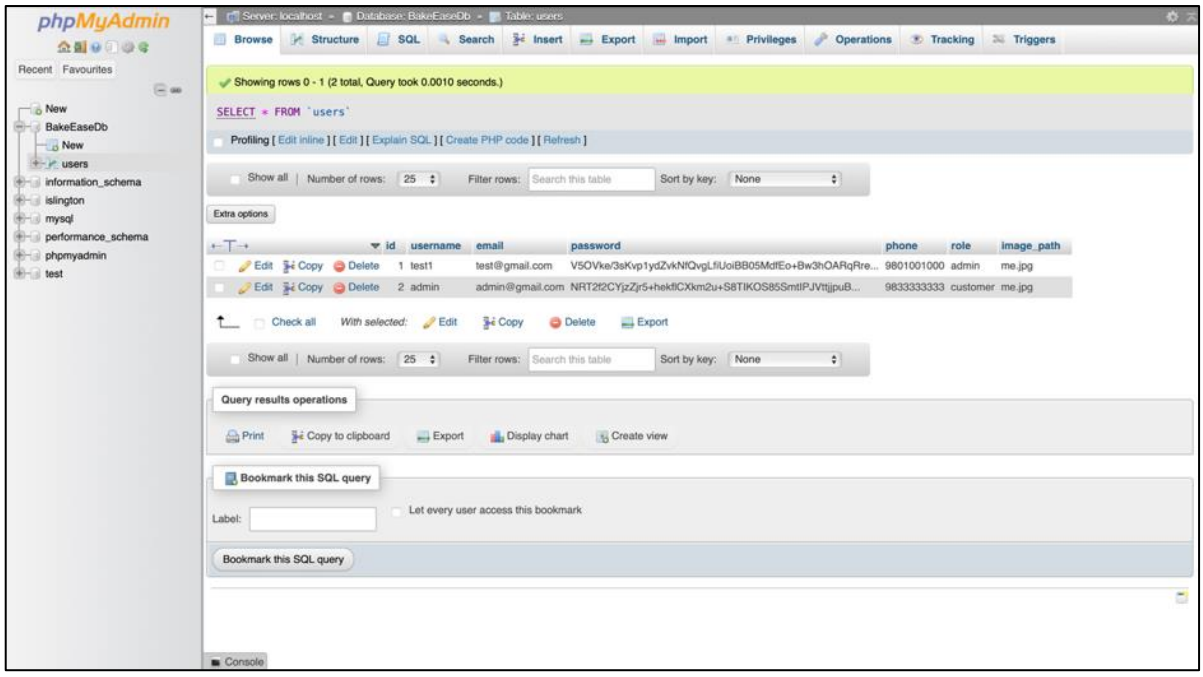


Figure 3: data in users table

2.3.2. Table 2: Products

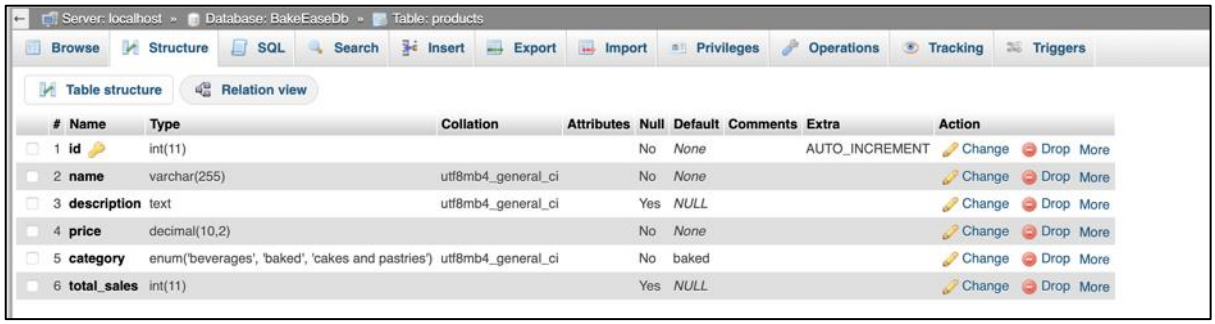


Figure 4: table structure of products table

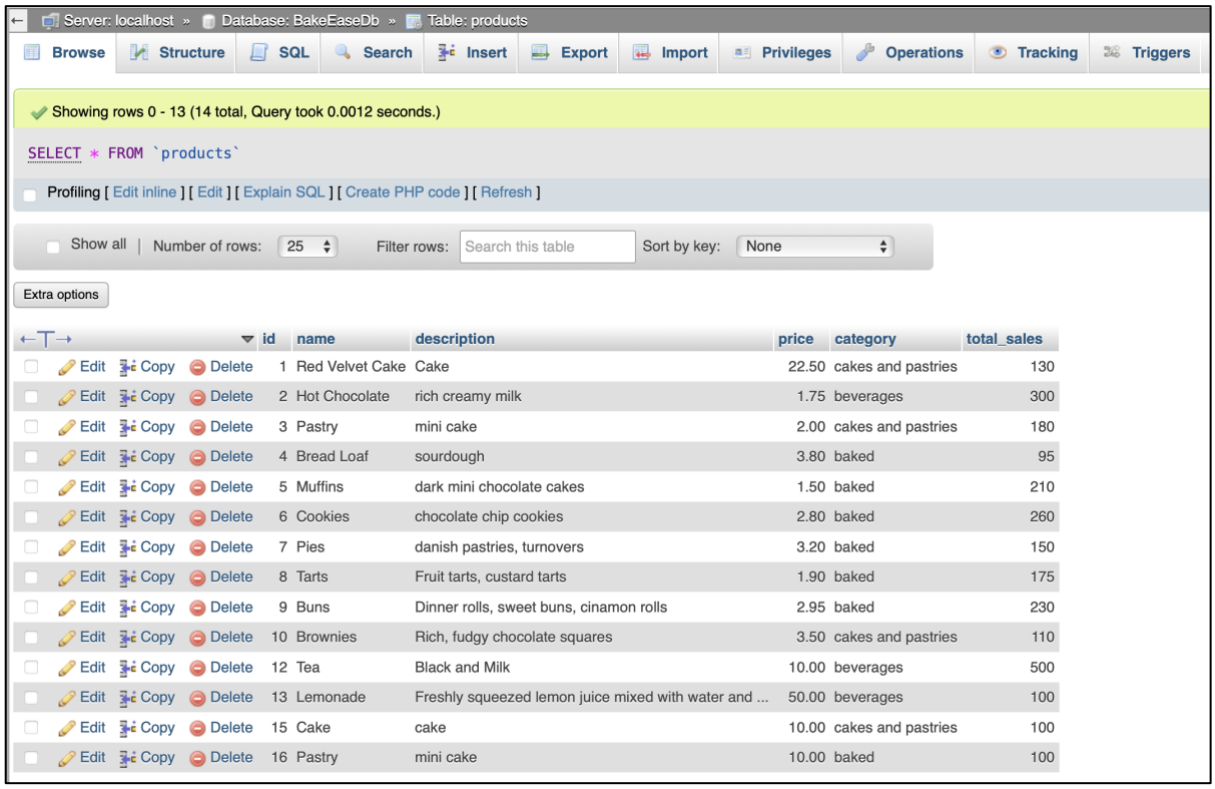


Figure 5: values inside products table

3. Wireframe and Design

Websites with their references were used to design the wireframe for our system. We have used Canva to design the wireframe and we implemented that design using Eclipse IDE in our webpage.

3.1. Navigation Bar

Navigation is an important element in a webpage as it allows user to navigate to different pages on the website.

3.1.1. Research Reference

For designing the navbar for my website, I took reference of an ecommerce middle layer website Tokari. The website's navigation bar matches my requirements as well as the design looks almost like what I need in my bakery website. (Tokari, n.d.)

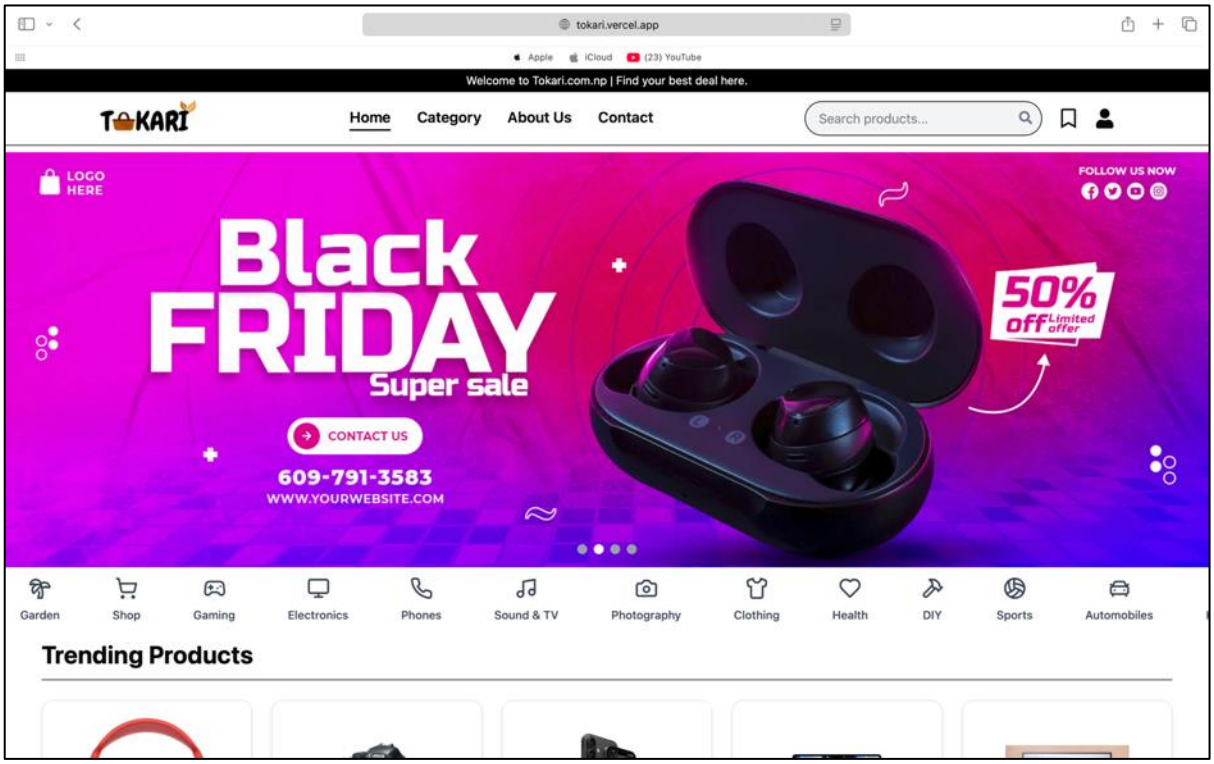


Figure 6: Referece of Nav bar

The reasons why I chose this as my reference are:

- Minimal Design
- Short company title above the navigation bar
- Format and elements selection

3.1.2. Wireframe Design

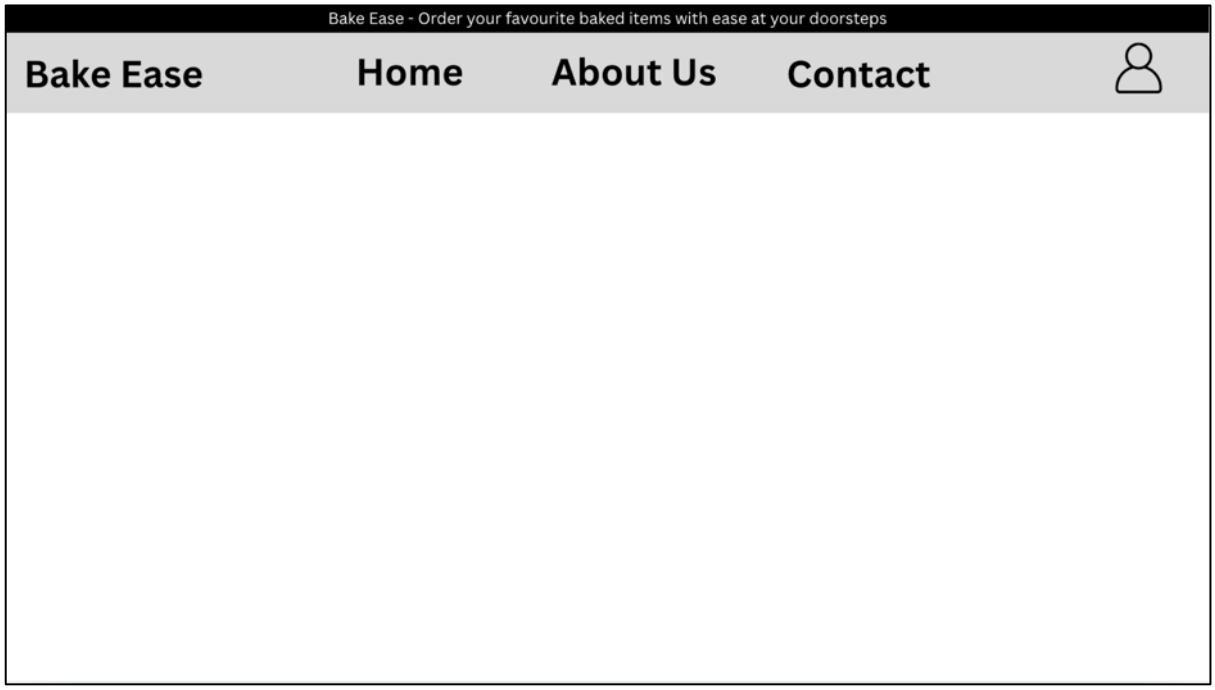


Figure 7: Designing the navbar wireframe in Canva

3.1.3. Design Development

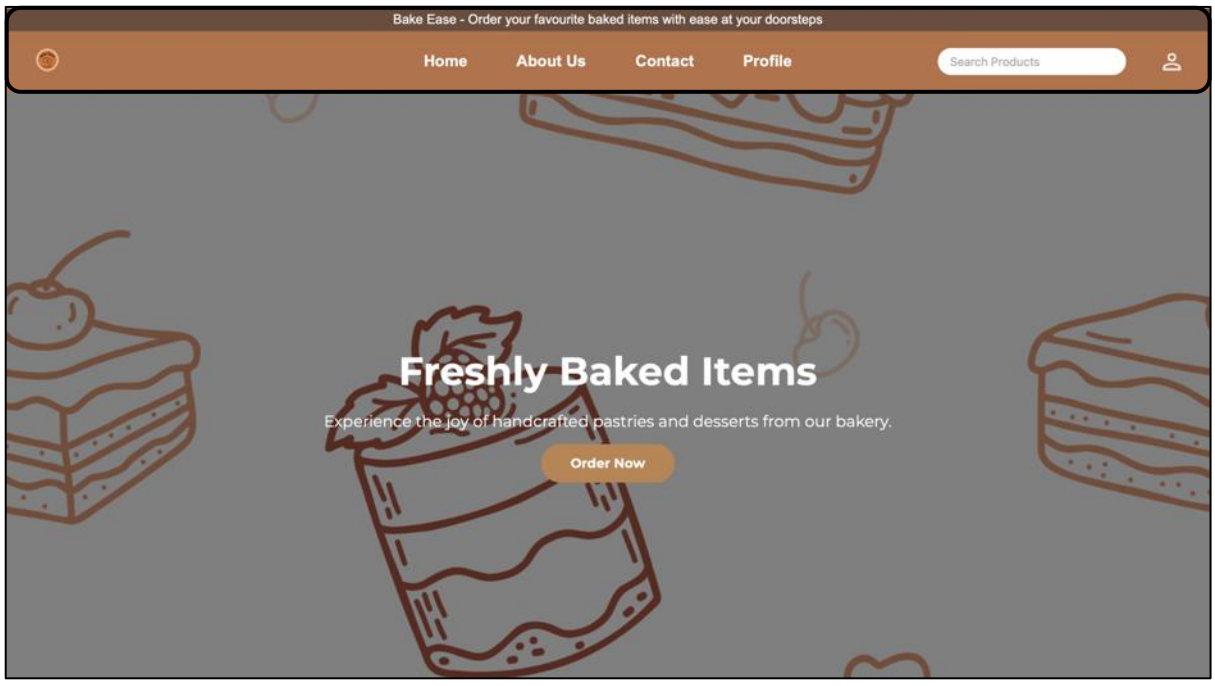


Figure 8: Design Development of navigation bar

3.2. Home Page

The home is the main entry point of the website as it is the first component that user interacts with. Hence the page is supposed to be attractive to improve user's impression towards the website.

3.2.1. Research Reference

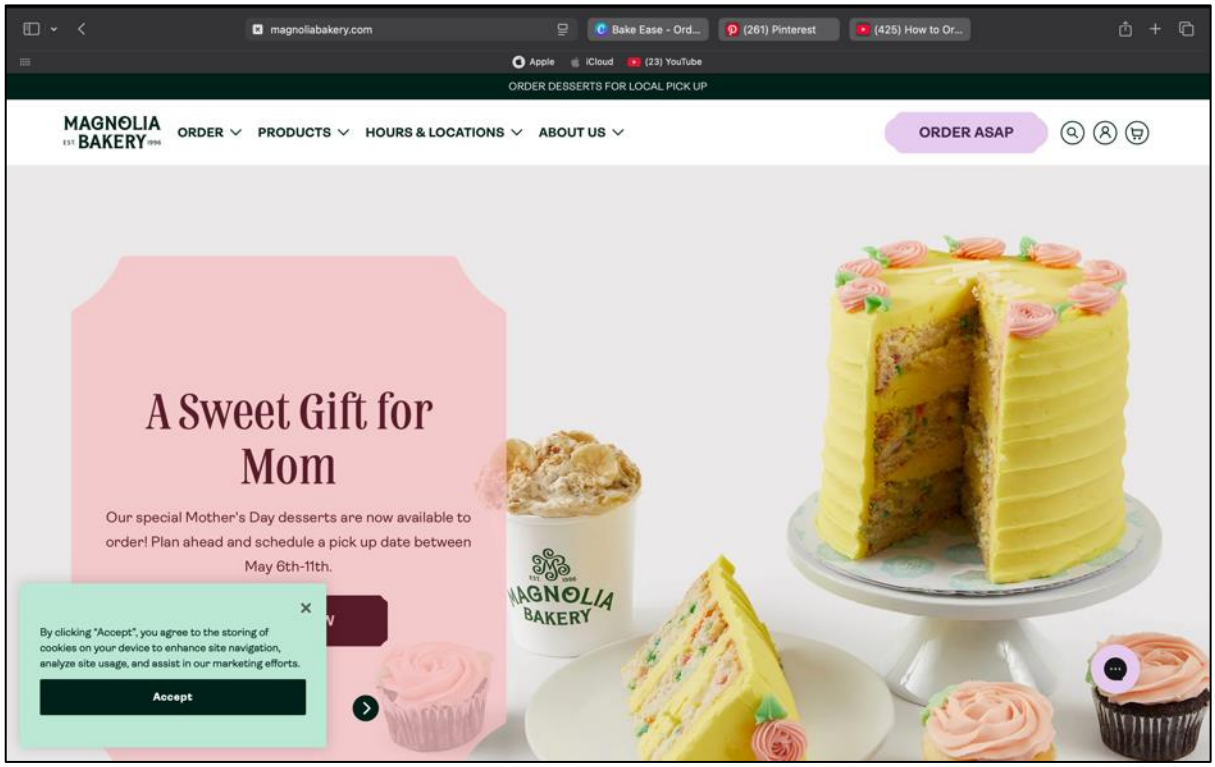


Figure 9: Research reference for Home page (i)

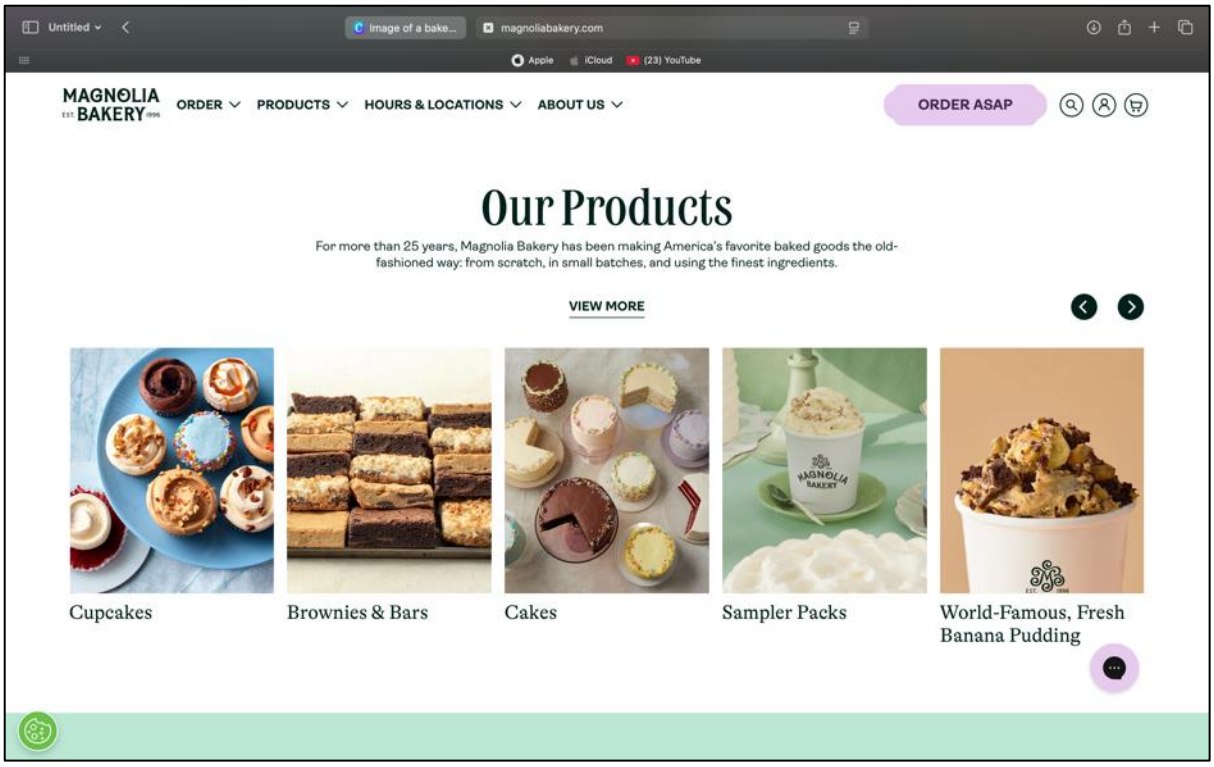


Figure 10: research reference for home page (ii)

The reasons why I chose this as my reference are:

- Hero section banner design
- Products card design
- Design elements

3.2.2. Wireframe Design

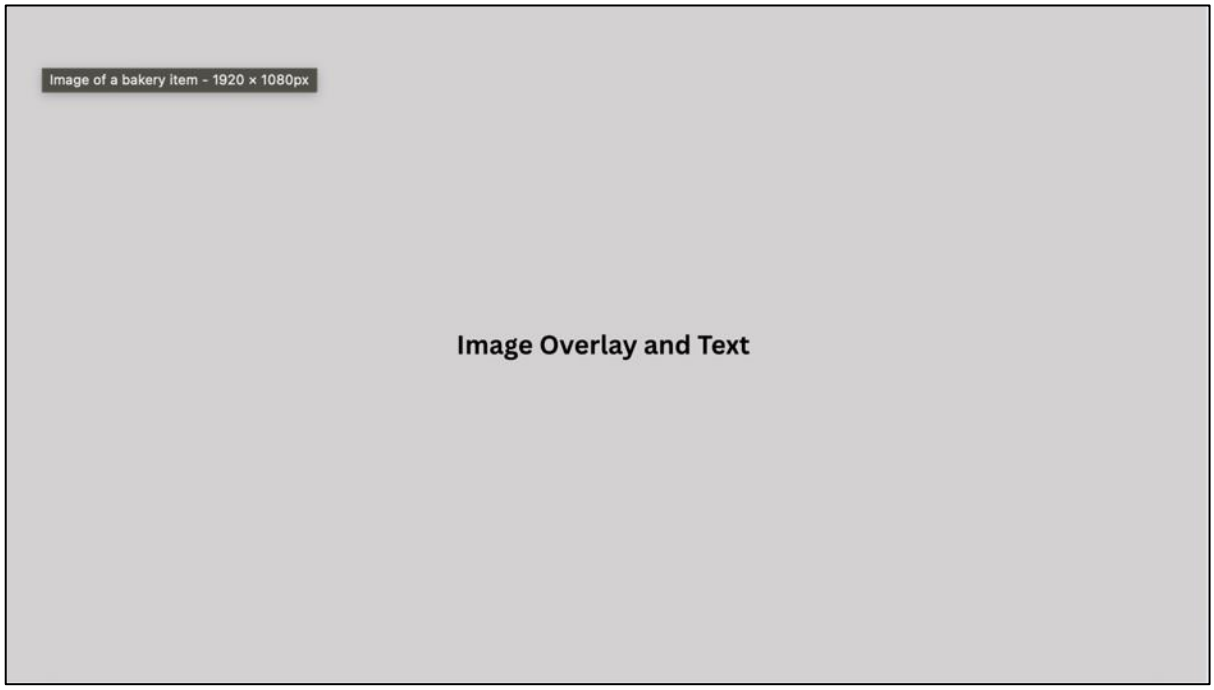


Figure 11: wireframe of Home page (i)

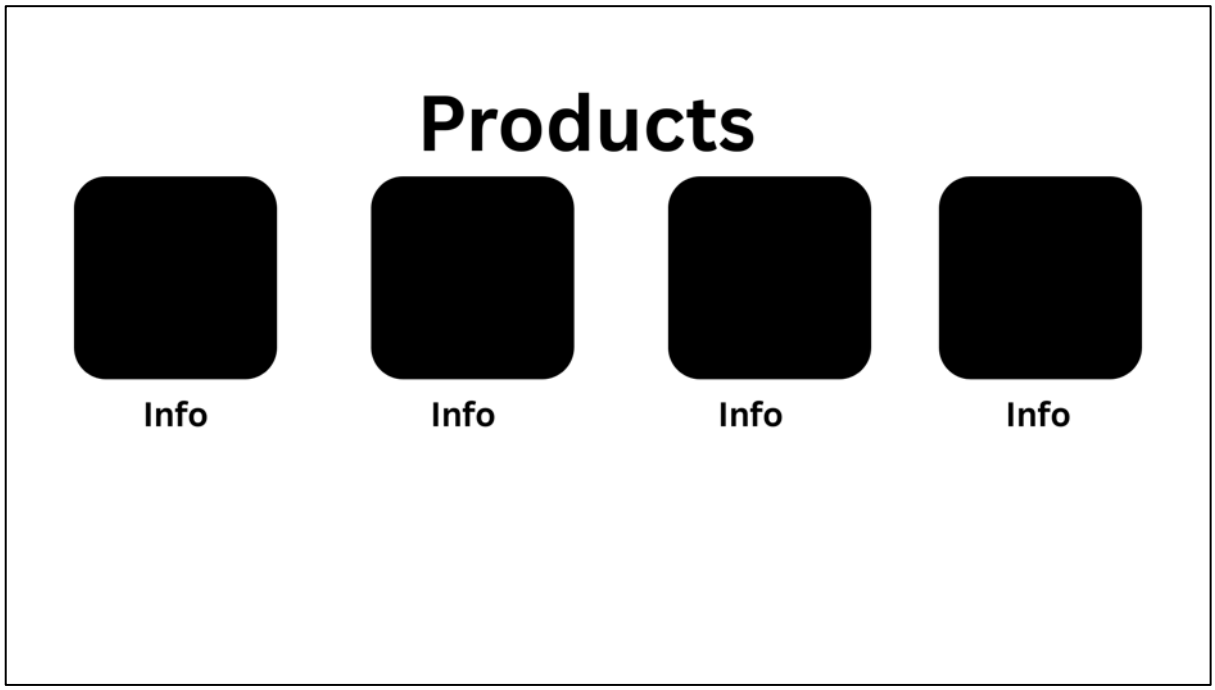


Figure 12: wireframe of home page (ii)

3.2.3. Design Development

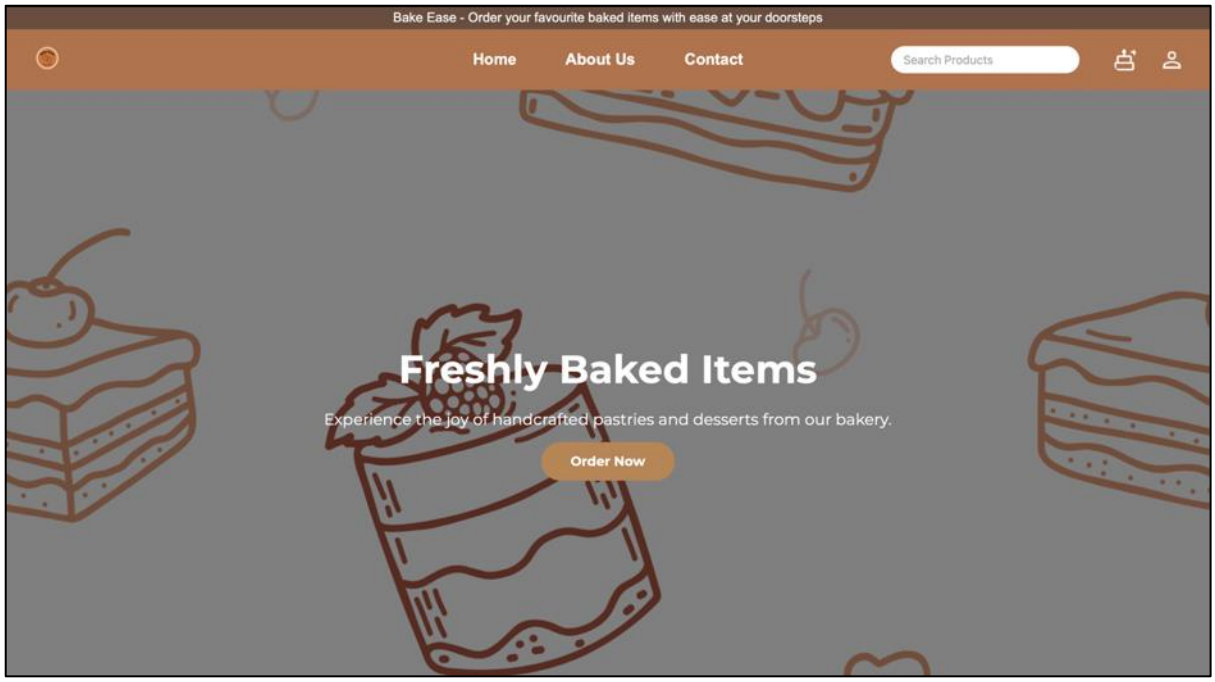


Figure 13: Home page design (i)

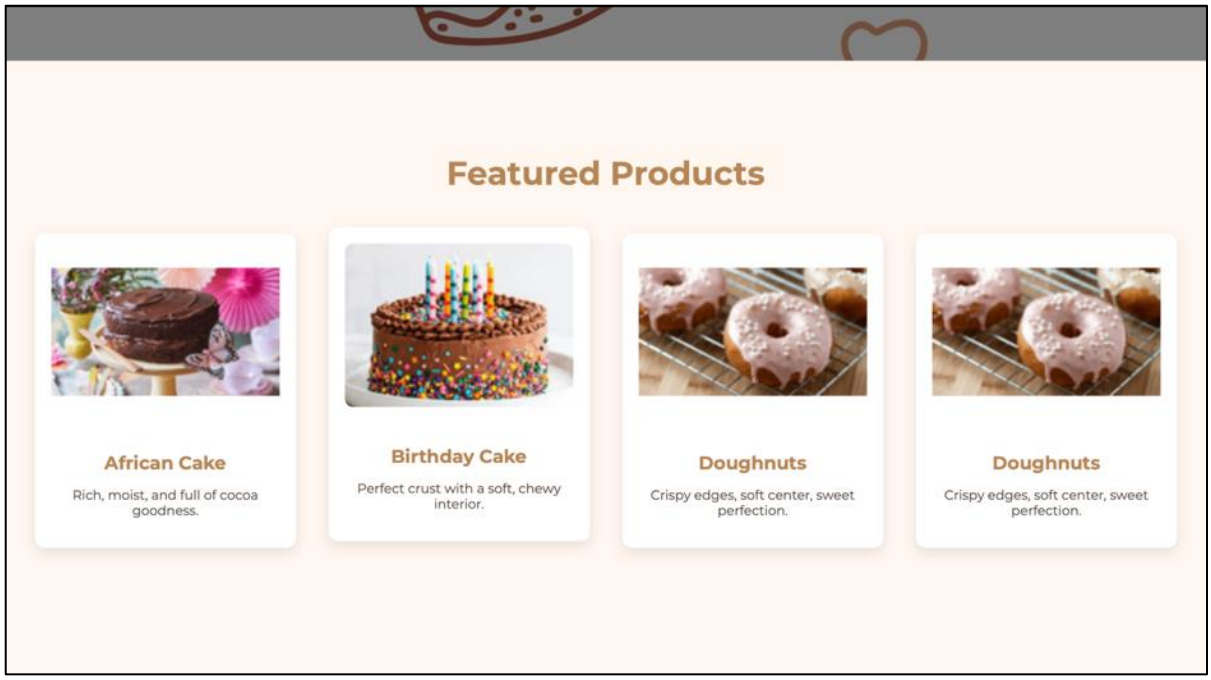


Figure 14: home page design (ii)

3.3. Products Page

The products page is an essential page that retrieves the data from the database and shows the available products to the user. Also, when the admin changes product information it shows the real time product information to the user. The reference for this product page is taken from daraz website. (Anon., n.d.)

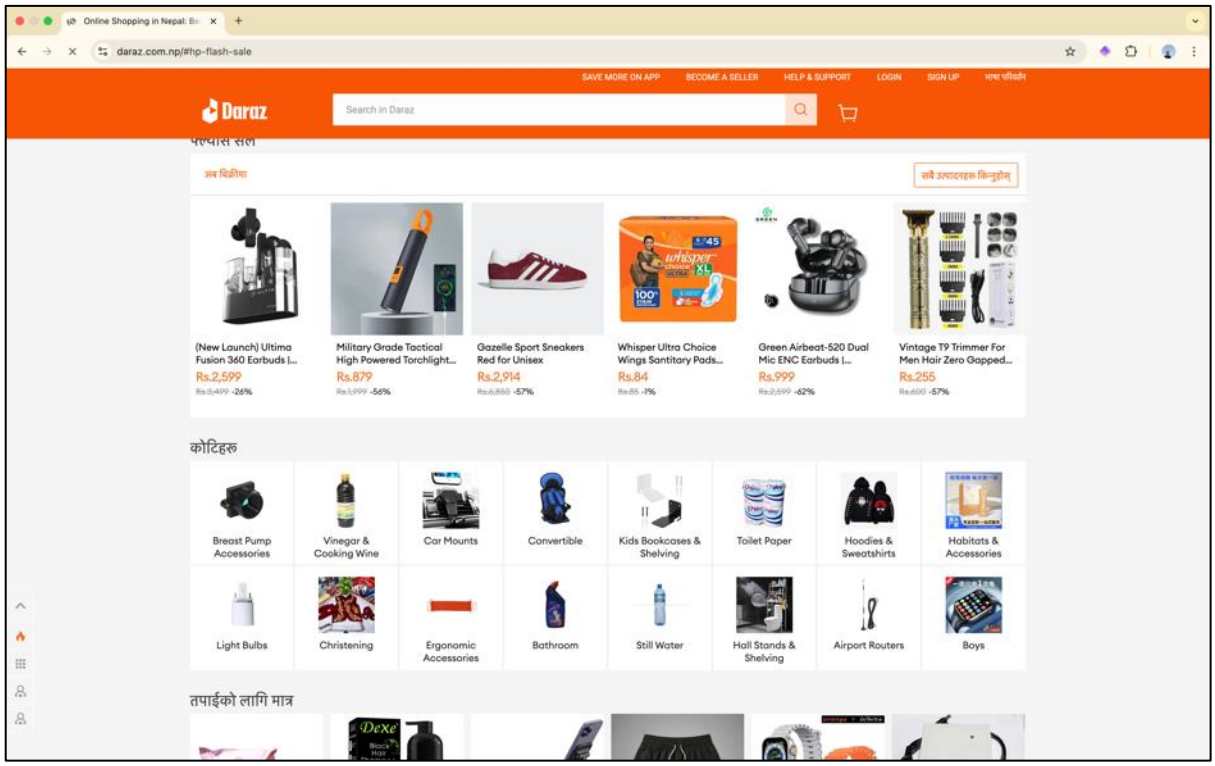


Figure 15: reference for product page

The main elements that convinced me to choose this website as my reference are the arrangements of product elements and their way of showing product description.

3.3.1. Wireframe Design

3.3.1.1. Design Development

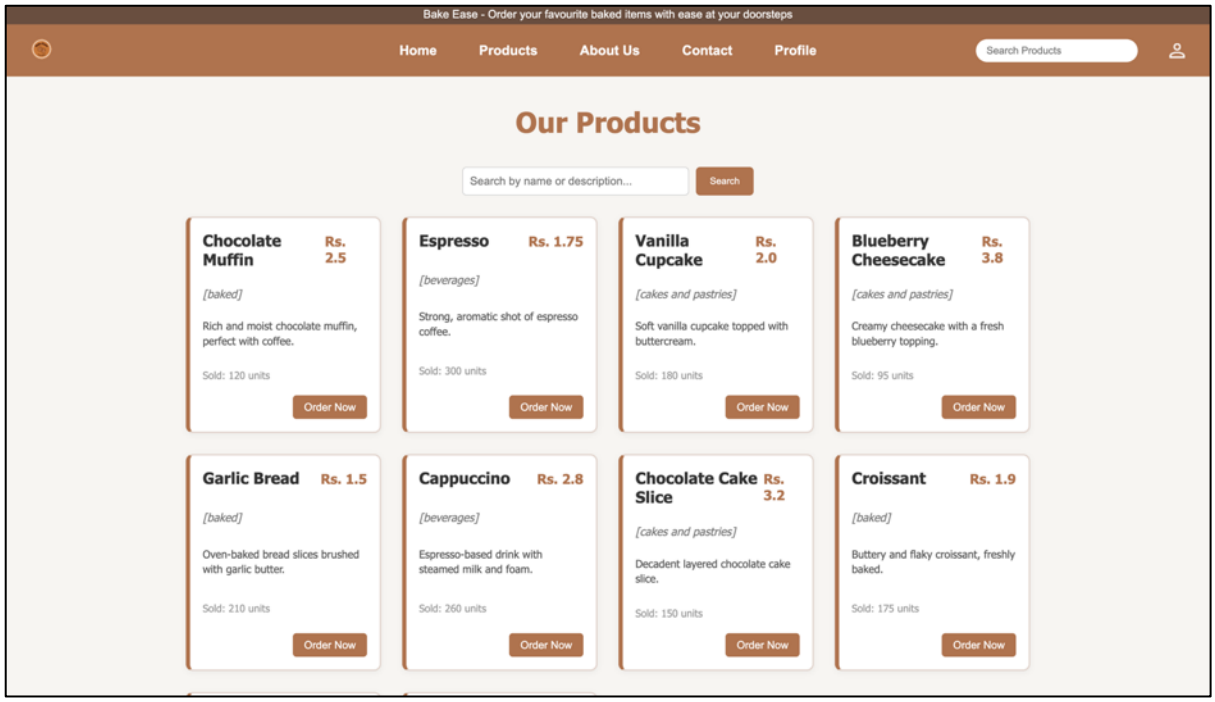


Figure 16: Actual product page design

3.4. Login/Register

Login/register page is used for authentication for user and admin as login page for user allows user to register to our website and for admin it allows perform main operations in our website. This is my simple project's login page which is built using react. It is a part of my personal project which is currently being built. (Tokari, n.d.)

3.4.1. Research Reference

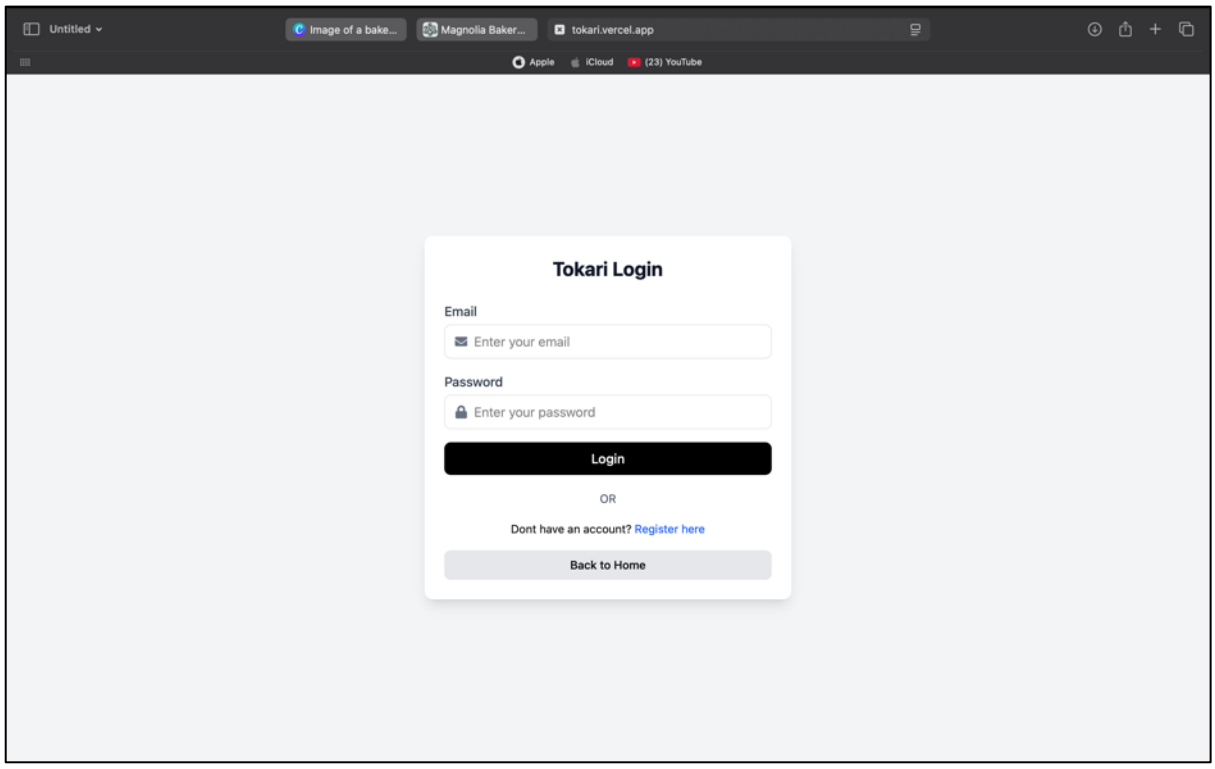


Figure 17: research reference for login/register

The reasons why I chose this as my reference are:

- Simple card design
- Color selection with the foreground and background
- Text fields and placeholders

3.4.2. Wireframe Design

Figure 18: wireframe for login/register

3.4.3. Design Development

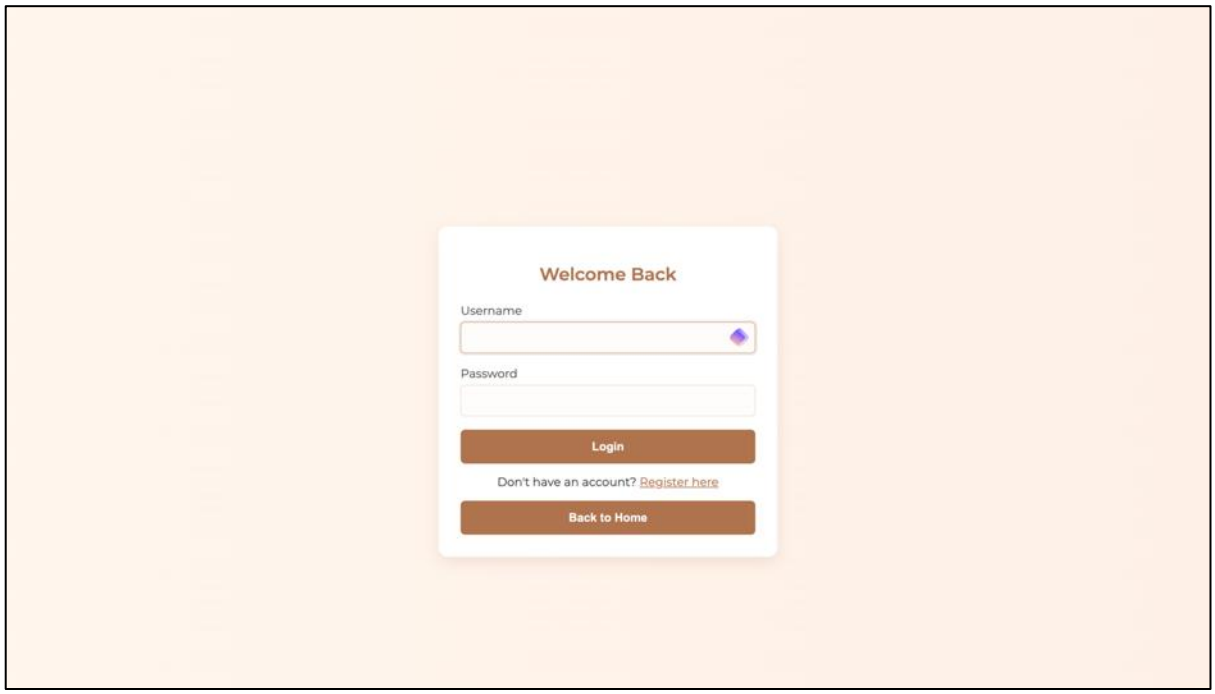


Figure 19: login page

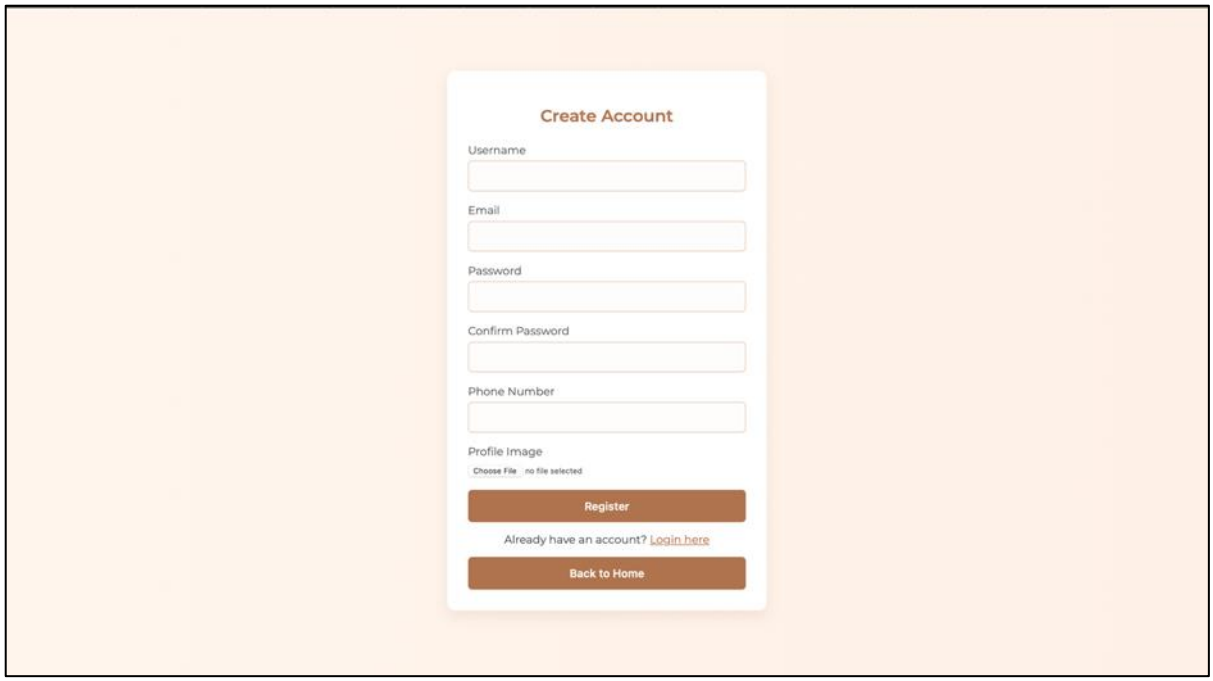


Figure 20: register page

3.5. Admin Page

Admin page is only accessible by admin to perform certain operations in the website which includes CRUD, changing banners etc.

3.5.1. Research Reference

For the research reference, I went to sideprojects website which had an image of prebuilt admin page which matches my requirements. (Projector, n.d.)

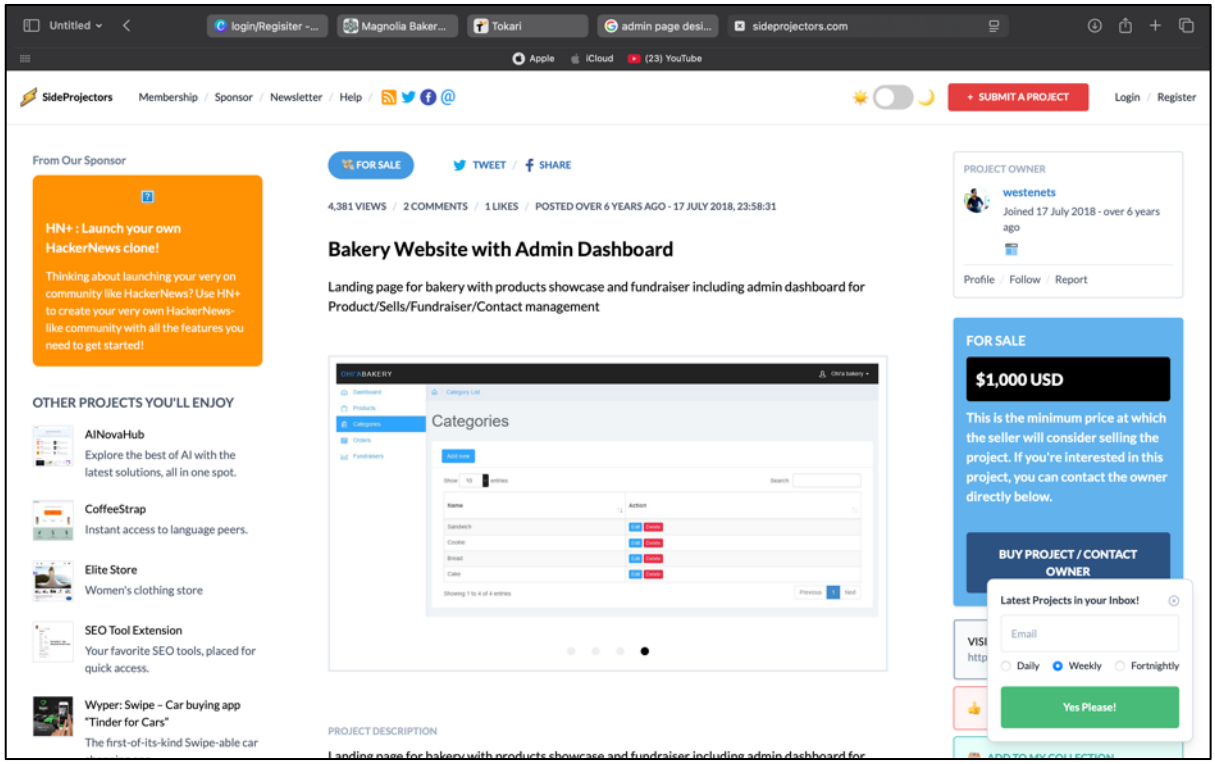


Figure 21: research reference of admin page

The reasons why I chose this as my reference are:

- Design Layout
- Options layout in the dashboard

3.5.2. Wireframe Design

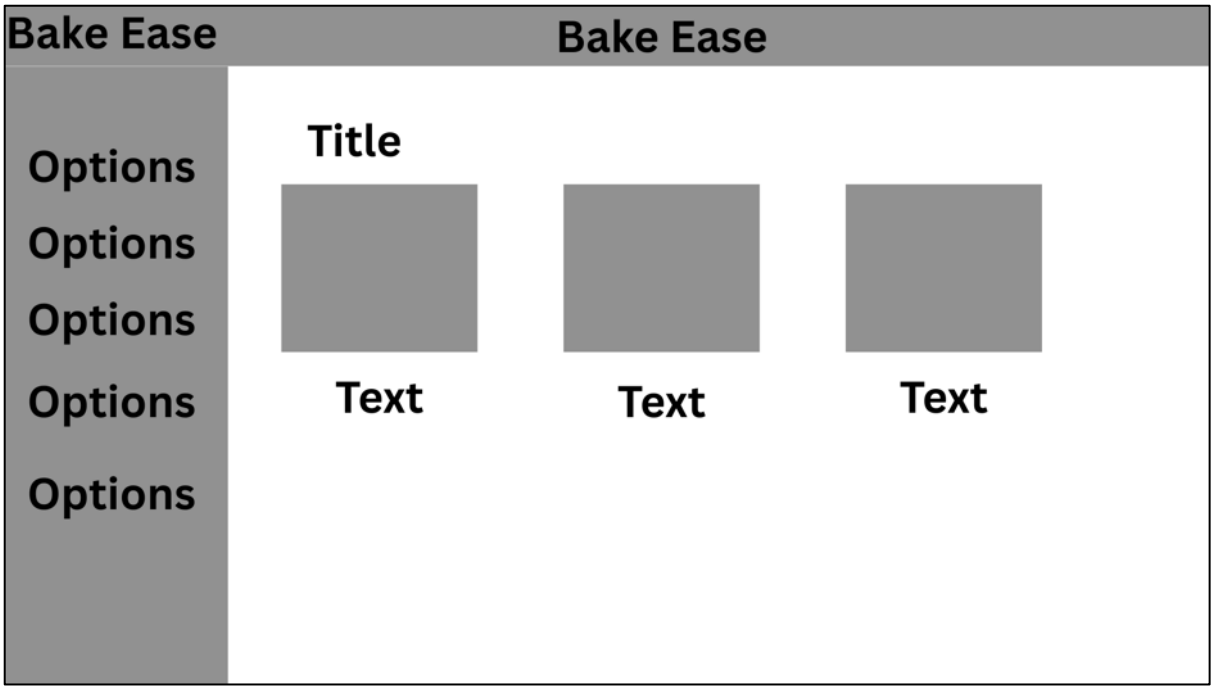


Figure 22: wireframe of admin page

3.5.3. Design Development

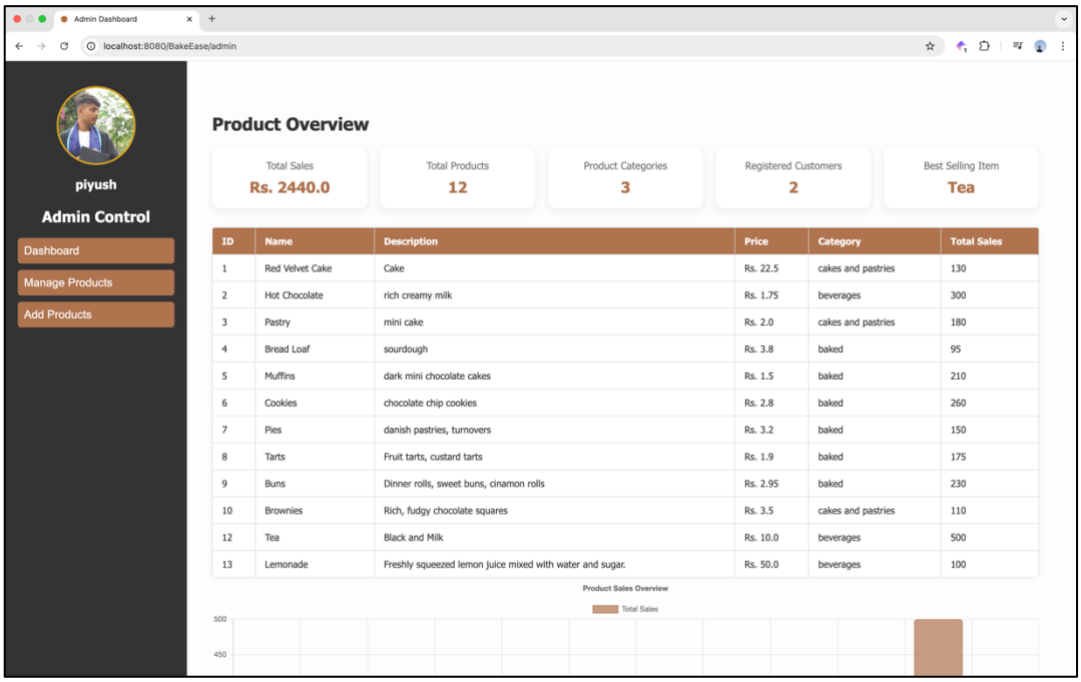


Figure 23: admin dashboard

3.6. Contact Page

The contact page is an essential element of a website where the user can contact the people involved in the project such as stakeholders, developers, owners etc. It establishes communication between the project team and the user.

3.6.1. Research Reference

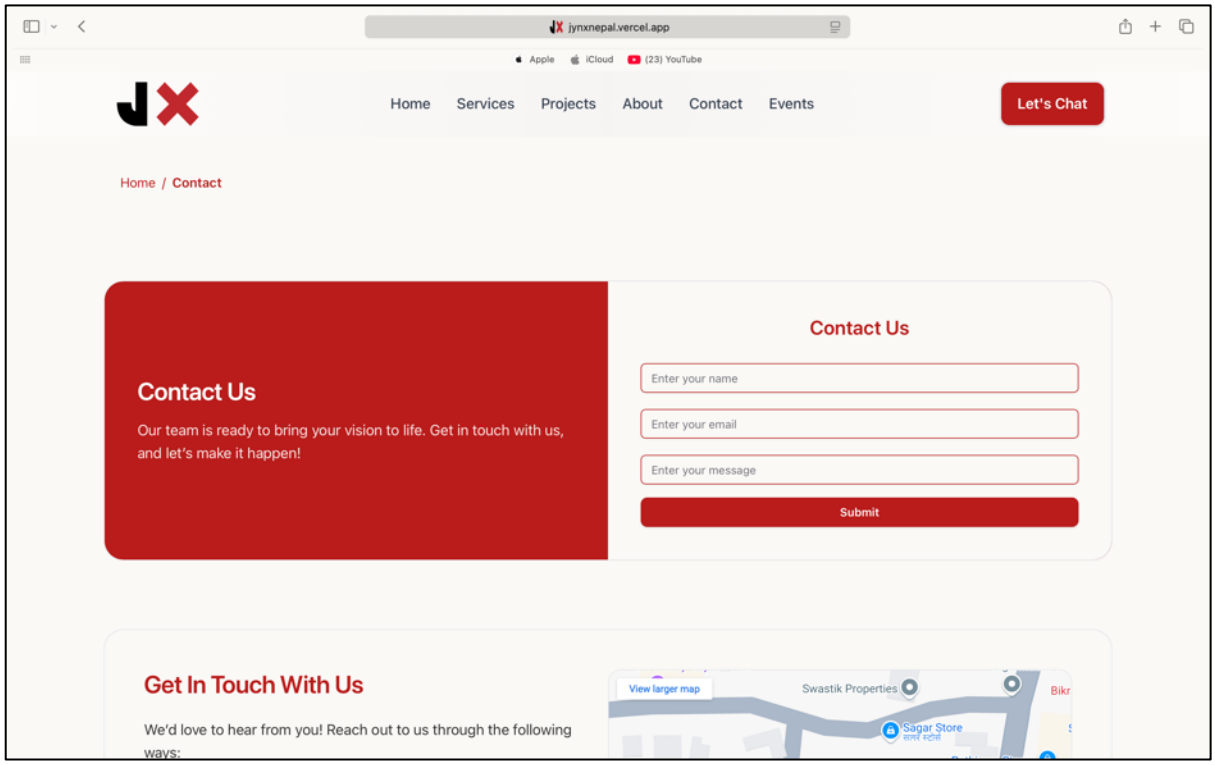


Figure 24: research reference of contact page

The reasons why I chose this as my reference are:

- Sidebar with color and following the same color pallet
- Specific design layout

3.6.2. Wireframe Design

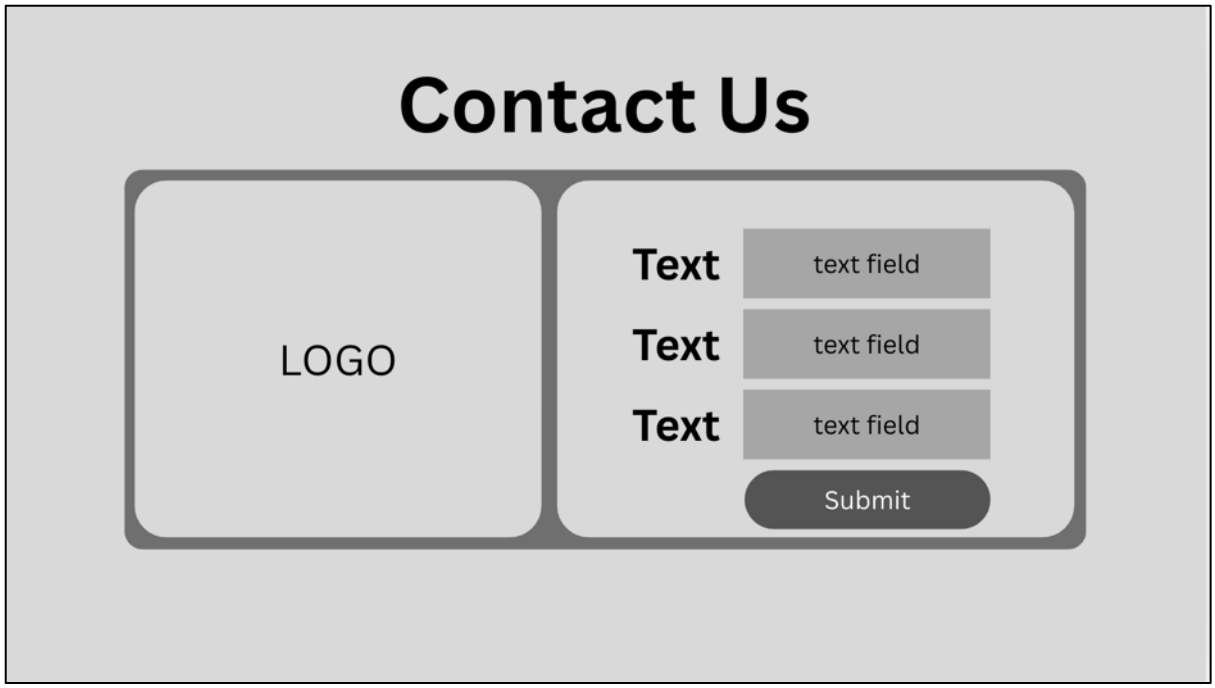


Figure 25: Wireframe of contact page

3.6.3. Design development

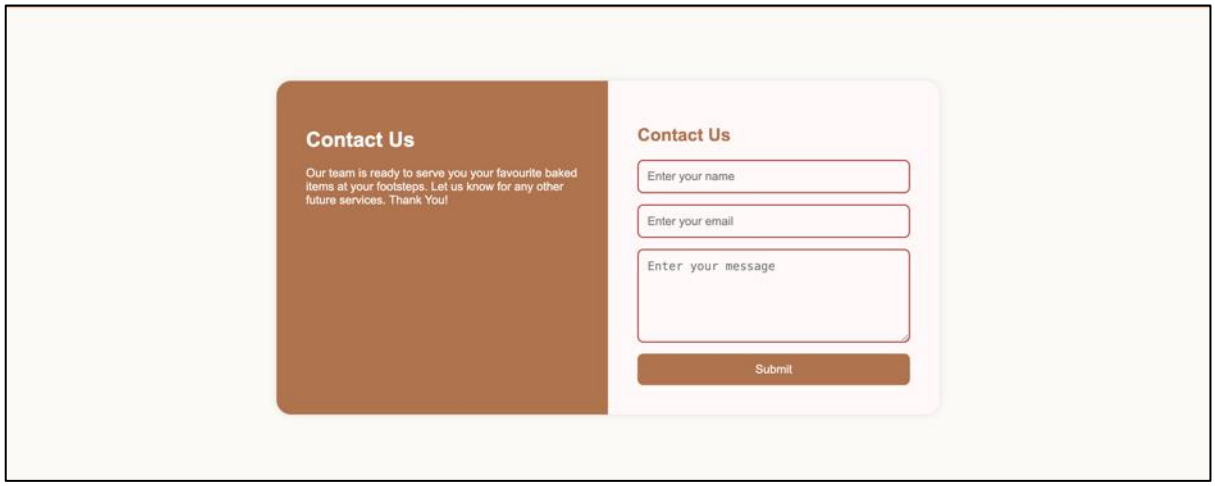


Figure 26: design of contact page

3.7. About us

The about us page contains information about the project and the team members working behind the scenes for the project.

3.7.1. Research reference

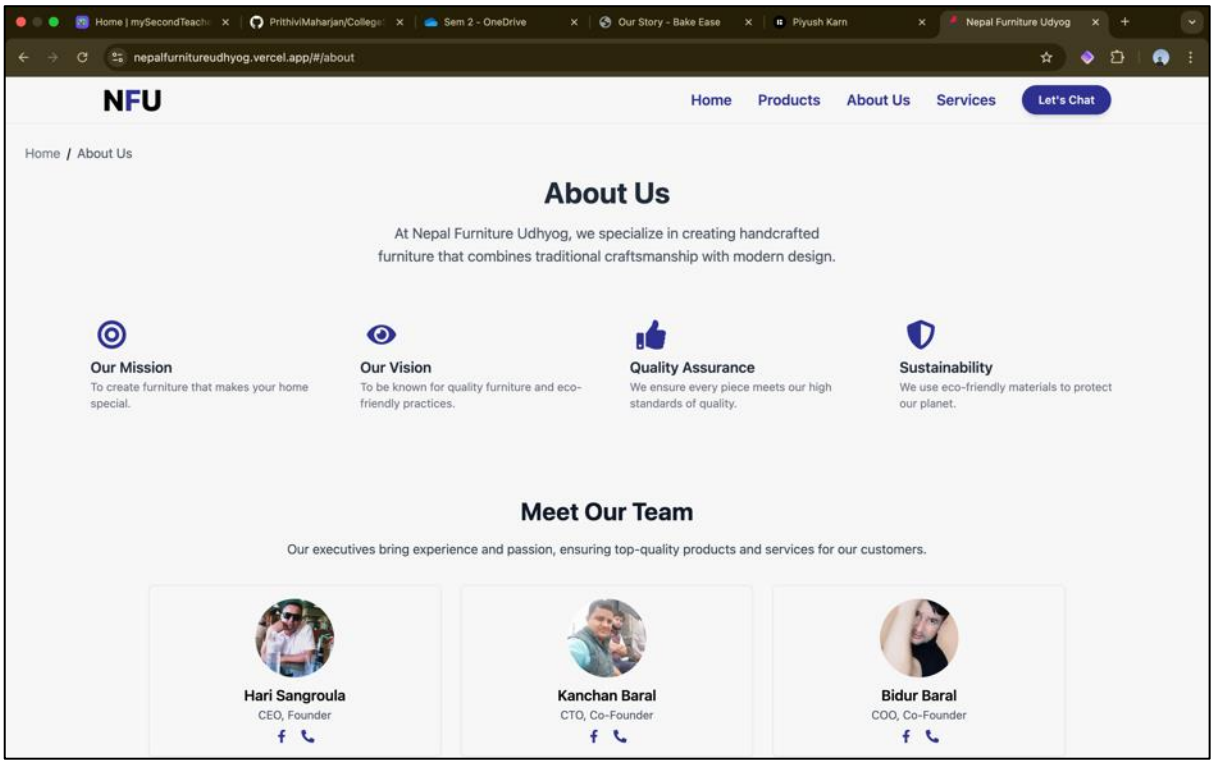


Figure 27: research reference of about page

The reasons why I chose this website as my reference are:

- Card layouts of team members elements
- Icons
- Color pallet selection

3.7.2. Wireframe Design

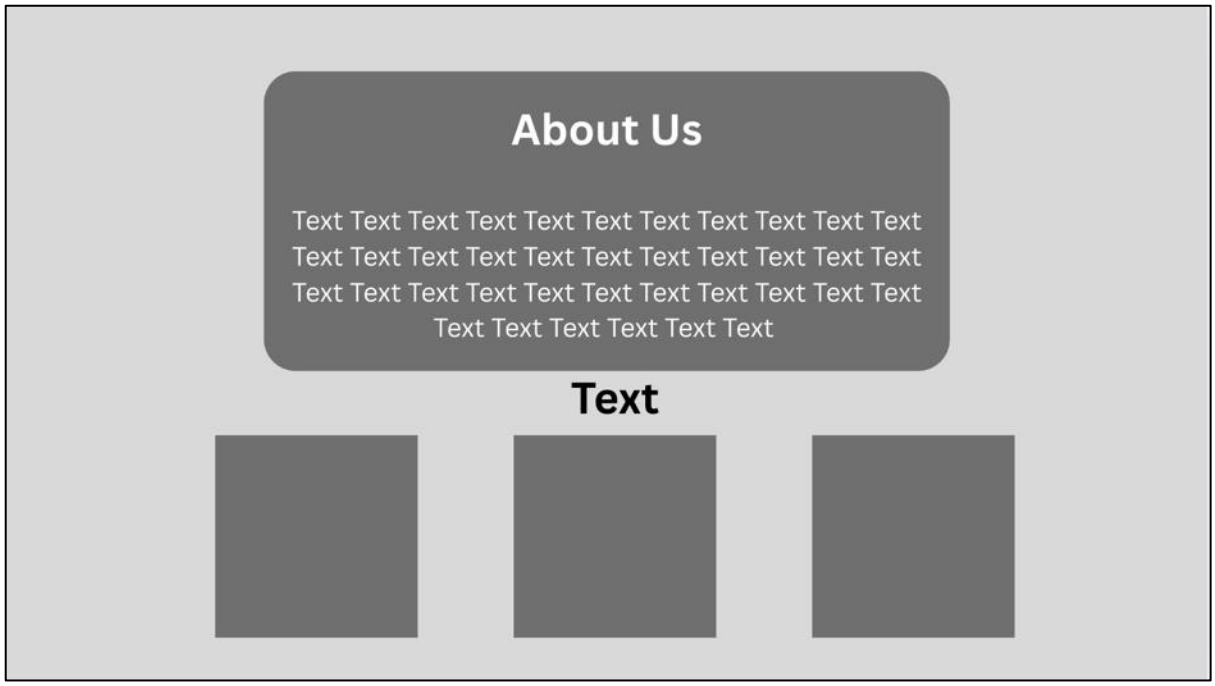


Figure 28: wireframe design of about us page

3.7.3. Design Development

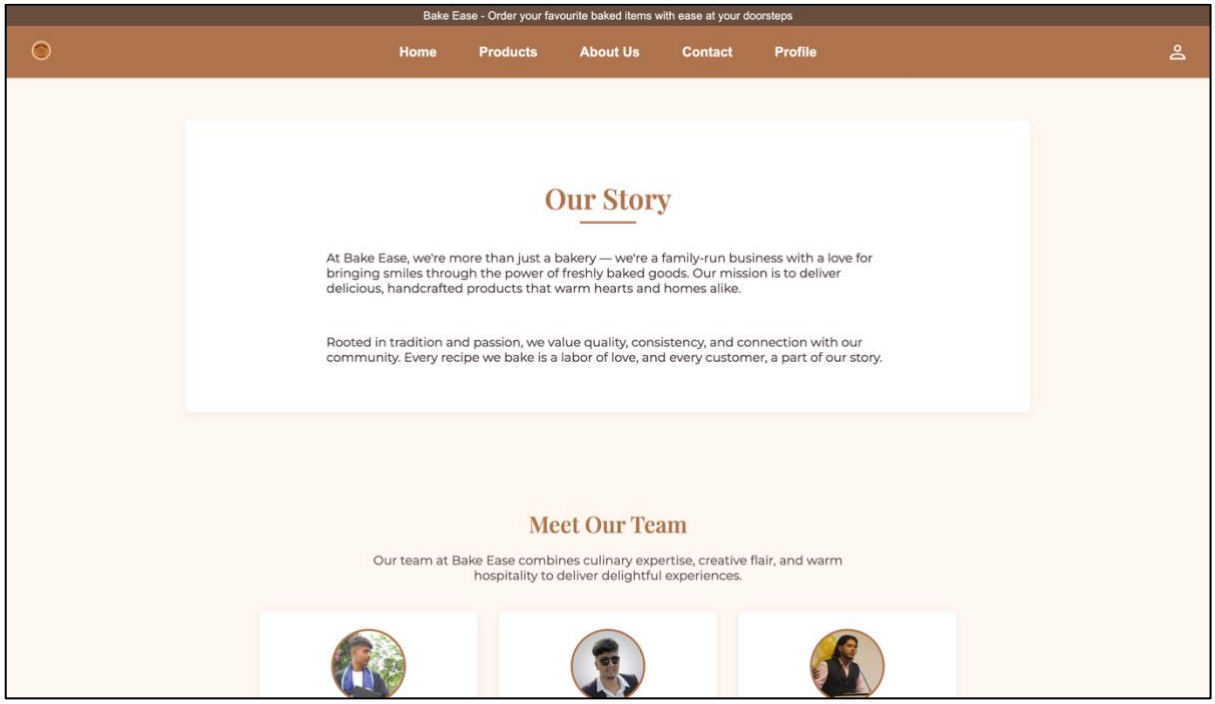


Figure 29: design of about page (i)

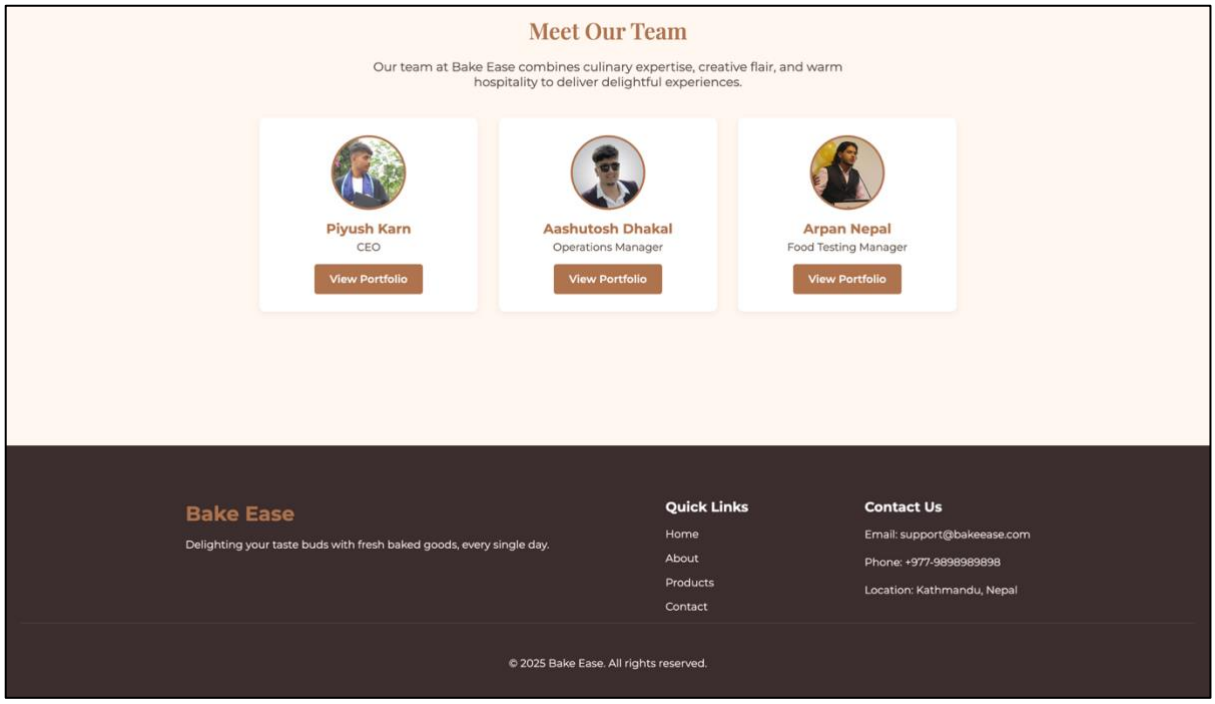


Figure 30: design of about page (ii)

3.8. Profile Page

The profile page is used to show the details of the user and also allows to update the user information through the webpage.

3.8.1. Research Reference

I found the research reference from a template and code editor website Sandbox.io (Sandbox.io, n.d.). Here, the layout and design are the things that impressed me to use this website as my reference.

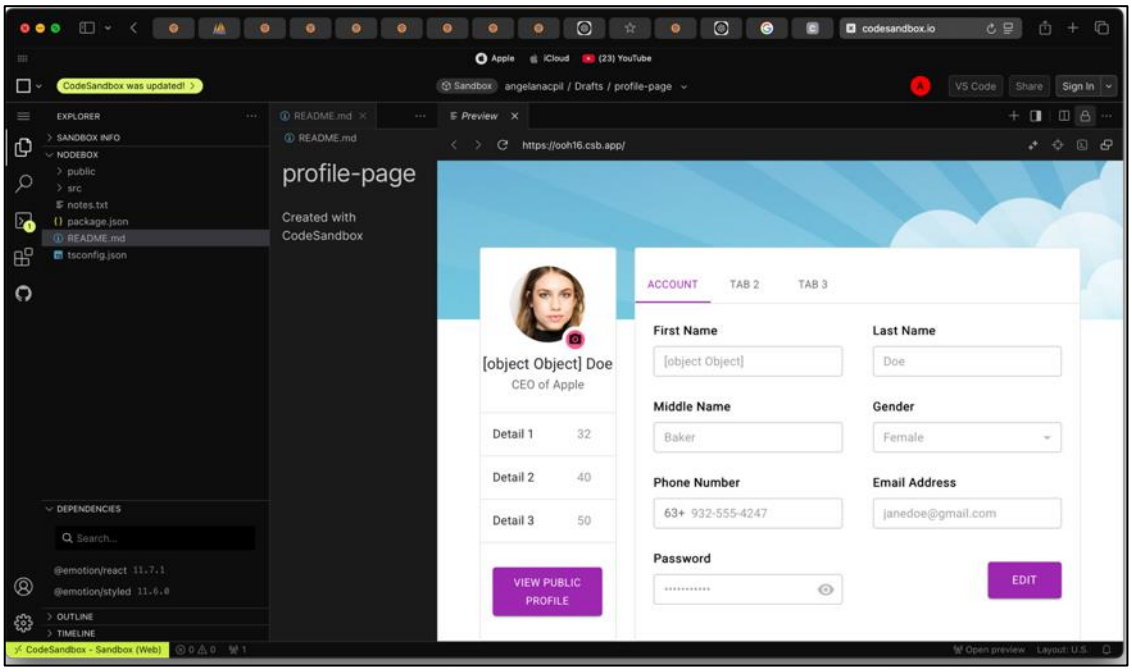


Figure 31: research reference for profile page

The elements that impressed me to chose this as my reference are:

- The left and right side bar elements
- Options management

3.8.2. Wireframe Design

Figure 32: wireframe of profile page

3.8.3. Design Development

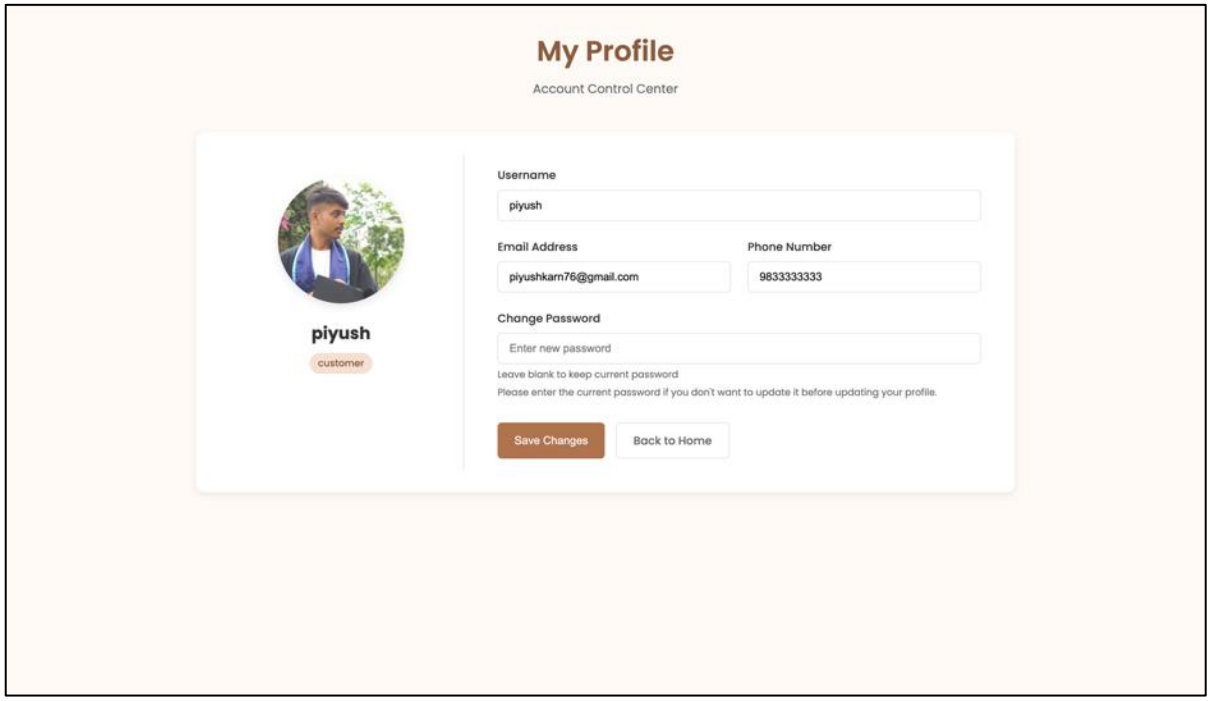


Figure 33: actual design of profile page

3.9. Footer

Footer contains some informational links and other information about the website such as the name of company, copyrights information etc.

3.9.1. Research reference

This is a website of digital marketing agency which contains a well-designed footer with links and copyrights information etc. The design and well-structured items are the things that impressed me to use this website as my reference. (Jynx, n.d.)

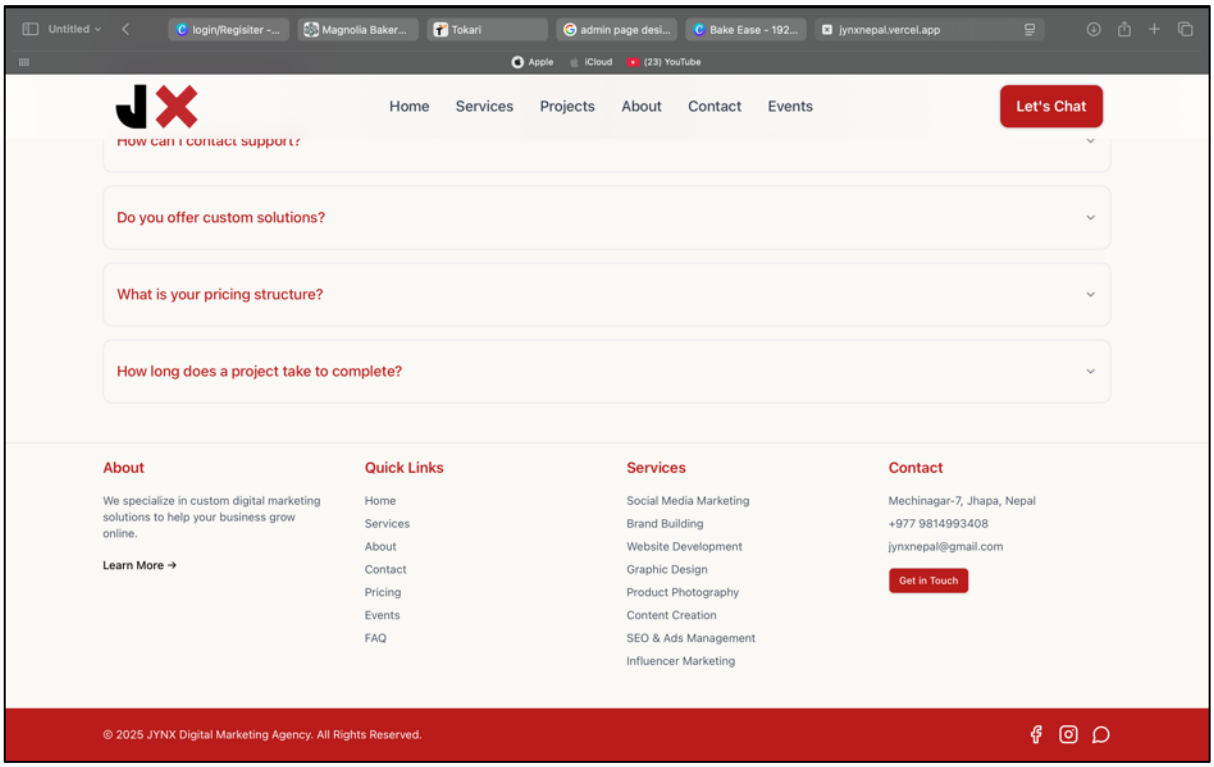


Figure 34: Research reference for footer

3.9.2. Wireframe Design

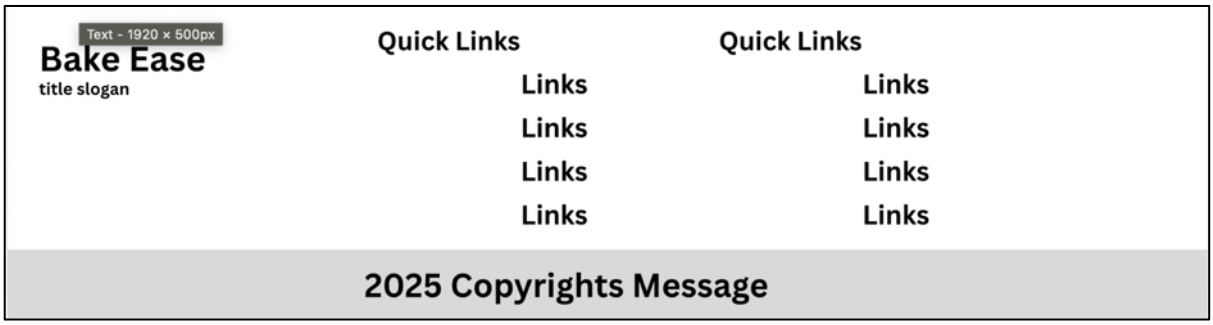


Figure 35: Wireframe design of footer

3.9.3. Design Development

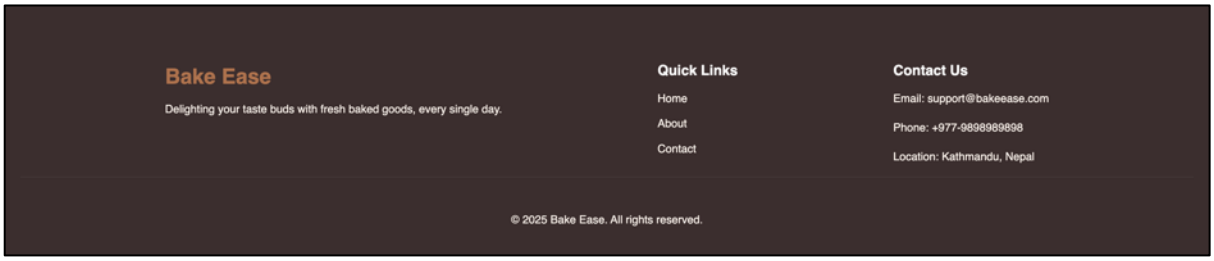


Figure 36: design of footer

4. Java Classes and Methods

4.1. Overall Class Diagram

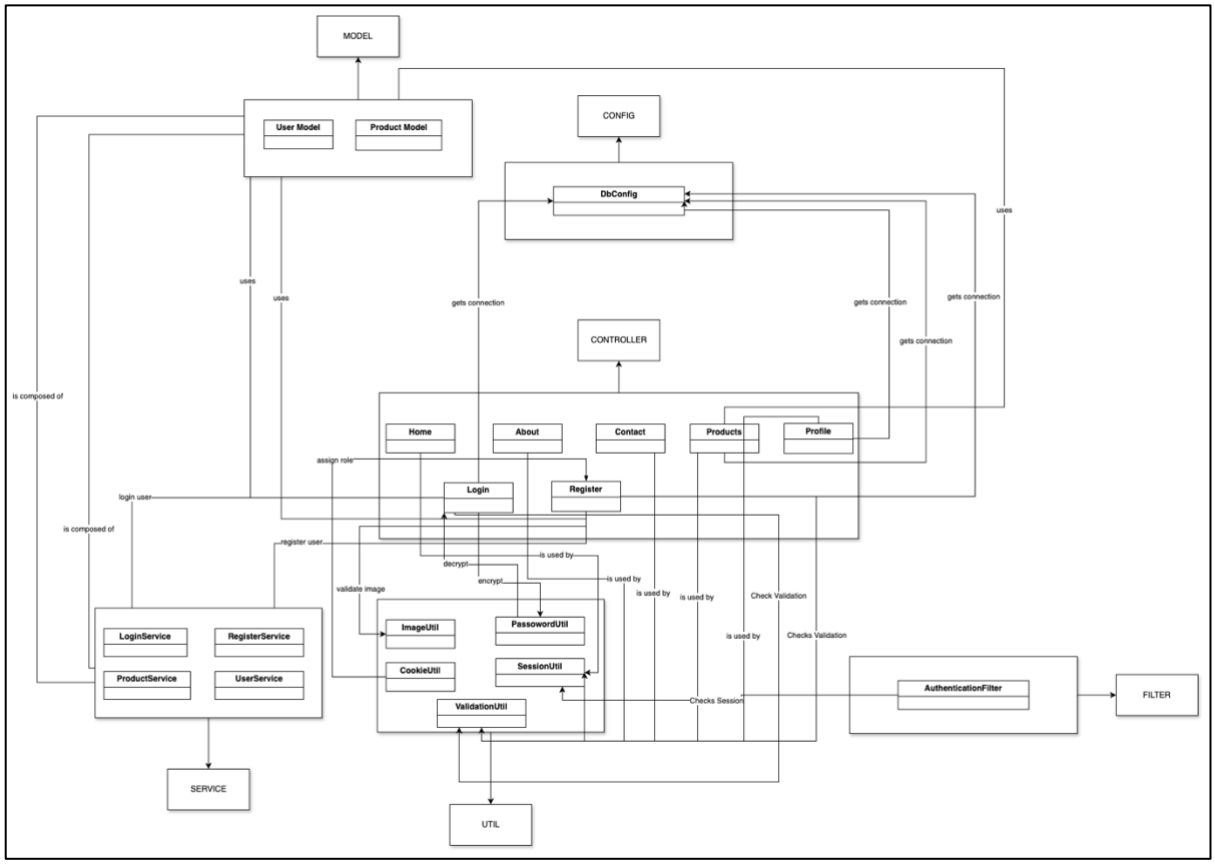


Figure 37: overall class diagram

4.2. Individual Class Diagram and Methods Description

4.2.1. DbConfig

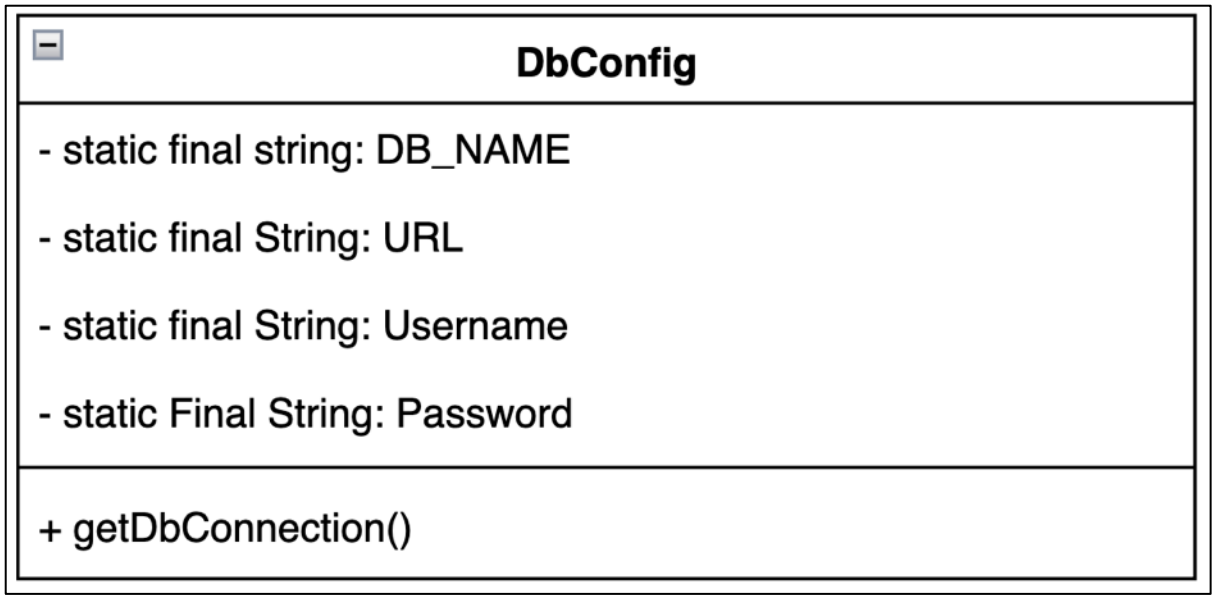


Figure 38: class diagram of DbConfig

This config file is responsible for establishing database connection to the MySql database with our BakeEase application. Its access modifier is public static and returns java sql connection. It loads the MySql driver Java Database Connectivity class and creates connection the connection through localhost server.

DbConfig.java

```
1  package com.bakeease.config;
2
3  import java.sql.Connection;
6
7  /**
8   * Database configuration class for BakeEase application.
9   * Provides method to establish a connection with the database.
10  */
11 public class DbConfig {
12
13     // Database configuration information
14     private static final String DB_NAME = "BakeEaseDb"; // Your database name
15     private static final String URL = "jdbc:mysql://localhost:3306/" + DB_NAME;
16     private static final String USERNAME = "root"; // Your DB username
17     private static final String PASSWORD = ""; // Your DB password
18
19     /**
20      * Establishes a connection to the database.
21      *
22      * @return Connection object for the database
23      * @throws SQLException if a database access error occurs
24      * @throws ClassNotFoundException if the JDBC driver class is not found
25      */
26     public static Connection getDbConnection() throws SQLException, ClassNotFoundException {
27         Class.forName("com.mysql.cj.jdbc.Driver");
28         return DriverManager.getConnection(URL, USERNAME, PASSWORD);
29     }
30 }
```

Figure 39: code for DbConfig file

4.2.2. About.java controller

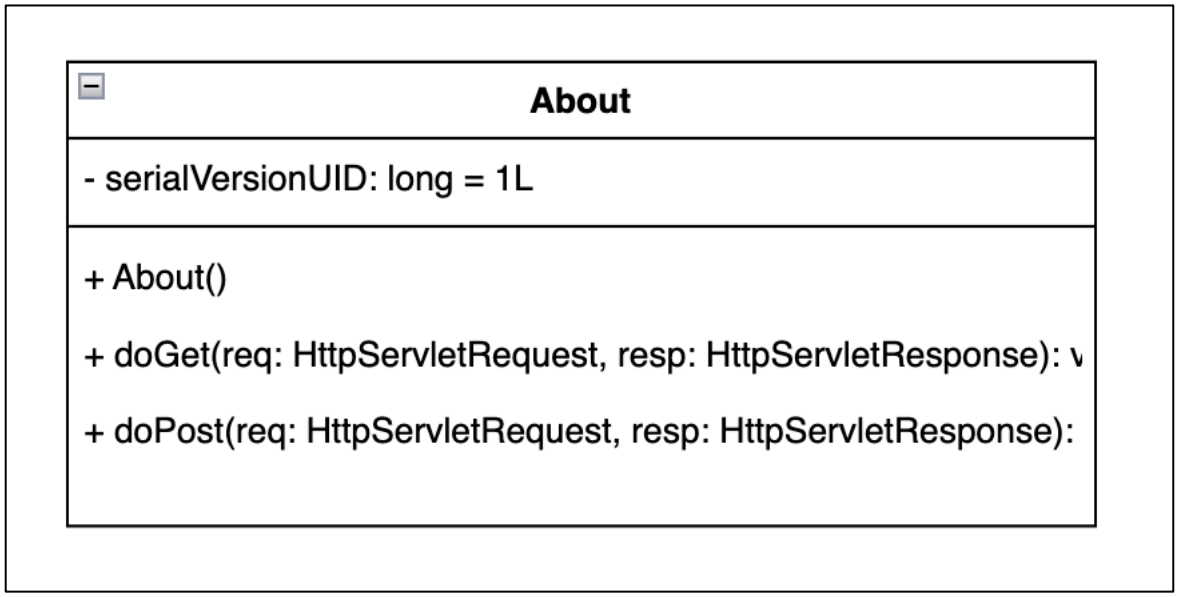
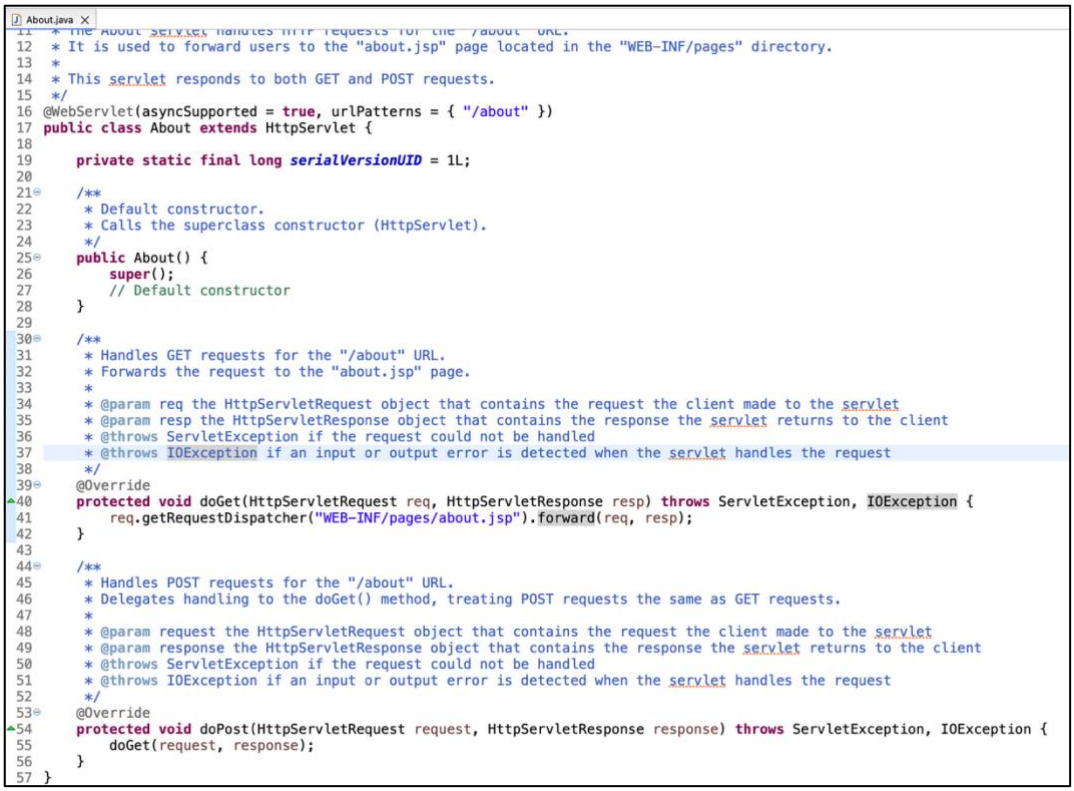


Figure 40: method description for about controller

This controller file contains two methods doGet and doPost. The doGet method is used to load the jsp page in the web page. Its access modifier is protected and it shows the about.jsp file when redirected to /about path in the web page where as the doPost method is used to handle http request only.



```java
About.java
 * It is used to forward users to the "about.jsp" page located in the "WEB-INF/pages" directory.
 *
 * This servlet responds to both GET and POST requests.
 */
@WebServlet(asyncSupported = true, urlPatterns = { "/about" })
public class About extends HttpServlet {

    private static final long serialVersionUID = 1L;

    /**
     * Default constructor.
     * Calls the superclass constructor (HttpServlet).
     */
    public About() {
        super();
        // Default constructor
    }

    /**
     * Handles GET requests for the "/about" URL.
     * Forwards the request to the "about.jsp" page.
     *
     * @param req the HttpServletRequest object that contains the request the client made to the servlet
     * @param resp the HttpServletResponse object that contains the response the servlet returns to the client
     * @throws ServletException if the request could not be handled
     * @throws IOException if an input or output error is detected when the servlet handles the request
     */
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.getRequestDispatcher("WEB-INF/pages/about.jsp").forward(req, resp);
    }

    /**
     * Handles POST requests for the "/about" URL.
     * Delegates handling to the doGet() method, treating POST requests the same as GET requests.
     *
     * @param request the HttpServletRequest object that contains the request the client made to the servlet
     * @param response the HttpServletResponse object that contains the response the servlet returns to the client
     * @throws ServletException if the request could not be handled
     * @throws IOException if an input or output error is detected when the servlet handles the request
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Figure 41: code for doGet and doPost method

4.2.3. Admin.java Controller

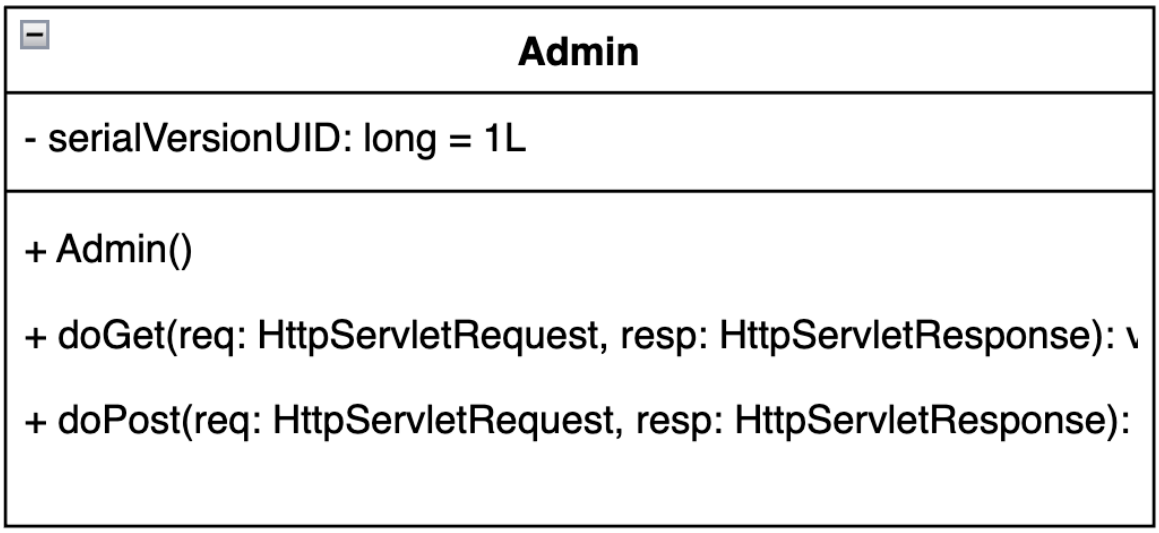


Figure 42: class diagram of about controller

The admin controller is used to handle admin operations in the web page. It contains two methods which are doGet and doPost. Its access modifier is public. The doGet method is used to check the route if the route is /admin and load the admin.jsp file. It fetches the user details of the logged in user, displays admin statistics and handles action based routing.

Admin.java

```
     * It also checks for a valid session and loads user-specific and product data.
     *
     * @param request the HttpServletRequest object that contains the request the client made to the servlet
     * @param response the HttpServletResponse object that contains the response the servlet returns to the client
     * @throws ServletException if an error occurs during request forwarding
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Check session for authentication
        HttpSession session = request.getSession(false);
        if (session == null || session.getAttribute("username") == null) {
            response.sendRedirect(request.getContextPath() + "/login");
            return;
        }

        // Retrieve logged-in user details
        String username = (String) session.getAttribute("username");
        UserModel user = userService.getUserByUsername(username);
        if (user != null) {
            request.setAttribute("user", user);
        }

        // Handle different actions: show addProduct or viewProducts page
        String action = request.getParameter("action");

        if ("addProduct".equals(action)) {
            request.getRequestDispatcher("/WEB-INF/pages/addProduct.jsp").forward(request, response);
        } else if ("viewProducts".equals(action)) {
            List<ProductModel> products = productService.getAllProducts();
            request.setAttribute("products", products);
            request.getRequestDispatcher("/WEB-INF/pages/viewProducts.jsp").forward(request, response);
        } else {
            // Load default admin dashboard data
            List<ProductModel> products = productService.getAllProducts();

            double totalSales = products.stream().mapToDouble(ProductModel::getTotalSales).sum();
            int items = products.size();
            int categories = 3; // Hardcoded; can be dynamically determined if needed
            int customers = userService.countCustomers();
            int orders = products.size(); // Assuming each product counts as one order
            String getBestSellingProductName = productService.getBestSellingProduct();

            // Set attributes for admin dashboard display
            request.setAttribute("totalSales", totalSales);
            request.setAttribute("items", items);
            request.setAttribute("categories", categories);
            request.setAttribute("customers", customers);
            request.setAttribute("orders", orders);
            request.setAttribute("products", products);
            request.setAttribute("bestSellingProduct", getBestSellingProductName);

            request.getRequestDispatcher("/WEB-INF/pages/admin.jsp").forward(request, response);
        }
    }
```

Figure 43: code for do get method

The doPost method contains the core logic for implementing adding product, updating product information and deleting product.

Admin.java

```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String action = request.getParameter("action");

    try {
        // Add a new product
        if ("add".equals(action)) {
            String name = request.getParameter("name");
            String description = request.getParameter("description");
            double price = Double.parseDouble(request.getParameter("price"));
            String category = request.getParameter("category");
            int totalSales = Integer.parseInt(request.getParameter("total_sales"));

            ProductModel product = new ProductModel();
            product.setName(name);
            product.setDescription(description);
            product.setPrice(price);
            product.setCategory(category);
            product.setTotalSales(totalSales);

            productService.addProduct(product);
            request.setAttribute("successMessage", "Product added successfully!");

        // Update an existing product
        } else if ("update".equals(action)) {
            int id = Integer.parseInt(request.getParameter("id"));
            String name = request.getParameter("name");
            String description = request.getParameter("description");
            double price = Double.parseDouble(request.getParameter("price"));
            String category = request.getParameter("category");
            int totalSales = Integer.parseInt(request.getParameter("total_sales"));

            ProductModel product = new ProductModel();
            product.setId(id);
            product.setName(name);
            product.setDescription(description);
            product.setPrice(price);
            product.setCategory(category);
            product.setTotalSales(totalSales);

            productService.updateProduct(product);
            request.setAttribute("successMessage", "Product updated successfully!");

        // Delete a product
        } else if ("delete".equals(action)) {
            int id = Integer.parseInt(request.getParameter("id"));
            productService.deleteProduct(id);
            request.setAttribute("successMessage", "Product deleted successfully!");
        }

    } catch (Exception e) {
        e.printStackTrace();
        request.setAttribute("successMessage", "An error occurred while processing the request.");
    }

```

Figure 44: code for doPost method

4.2.4. Contact.java controller

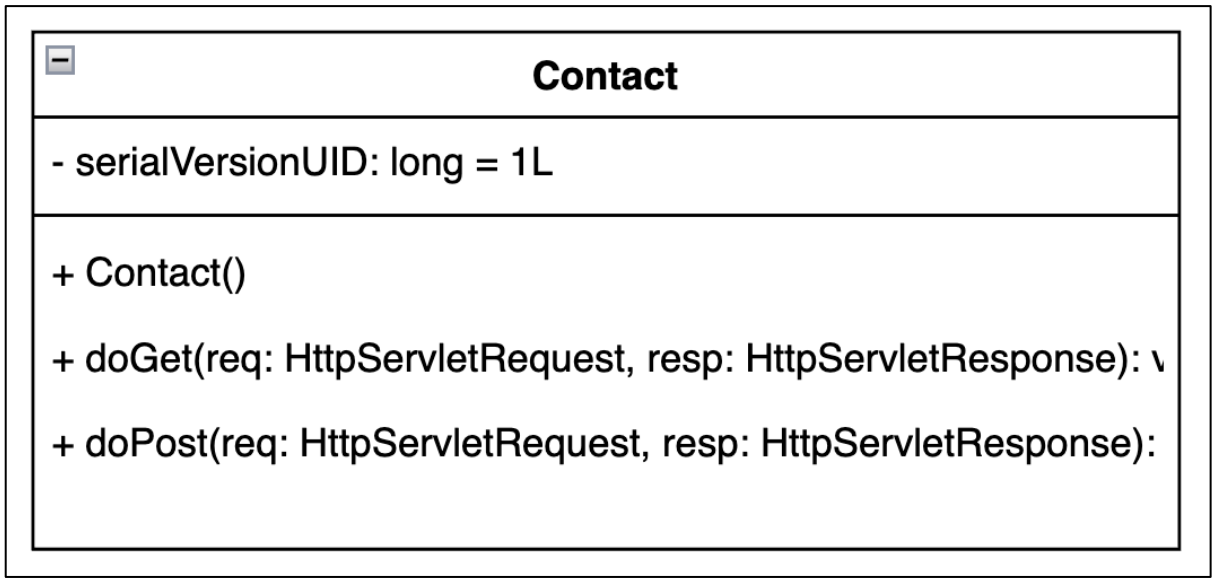


Figure 45: class diagram of contact controller

The contact controller is configured when /contact path is triggered in the web page. The doGet method is used to load the contact.jsp file in /contact path and the doPost method is used to handle the HTTP Get request.

Contact.java

```
1  package com.bakeease.controller;
2
3  import jakarta.servlet.ServletException;
9
10 /**
11  * Servlet implementation class Contact
12  */
13 @WebServlet(asyncSupported = true, urlPatterns = { "/contact" })
14 public class Contact extends HttpServlet {
15     private static final long serialVersionUID = 1L;
16
17     /**
18      * @see HttpServlet#HttpServlet()
19      */
20     public Contact() {
21         super();
22         // TODO Auto-generated constructor stub
23     }
24
25     /**
26      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
27      */
28     @Override
29     protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
30         req.getRequestDispatcher("WEB-INF/pages/contact.jsp").forward(req, resp);
31     }
32
33     /**
34      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
35      */
36     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
37         // TODO Auto-generated method stub
38         doGet(request, response);
39     }
40
41 }
42
```

Figure 46: code for doGet and doPost method

4.2.5. Home.java controller

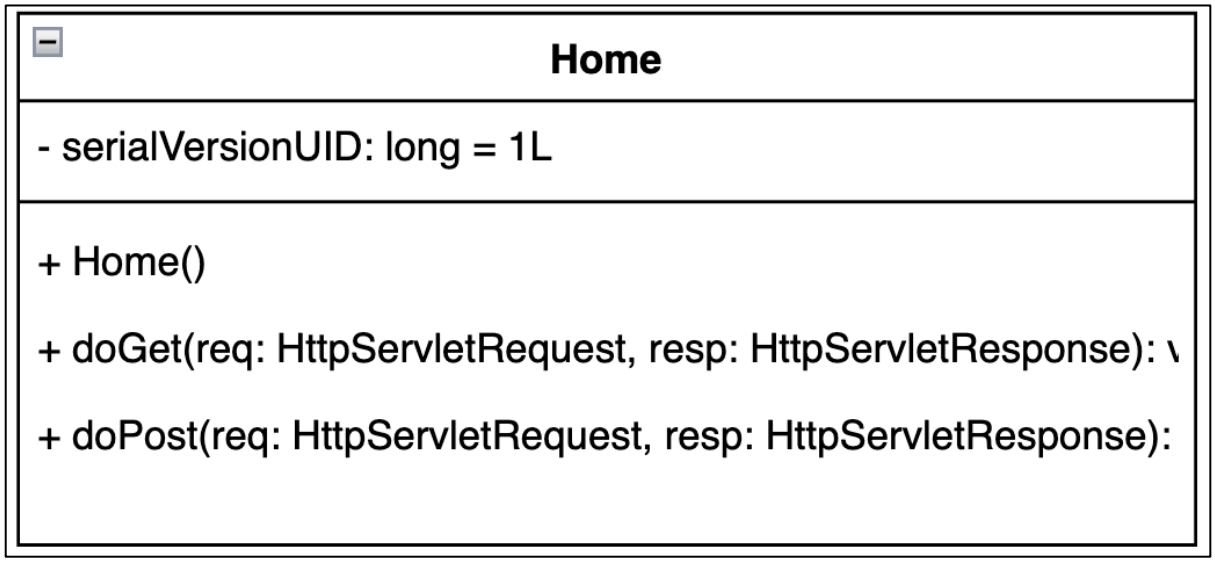


Figure 47: class diagram of home controller

The home controller is triggered when “/home” or “/” path is triggered in the web page. The doGet method is used to load the home.jsp file in the “/home” and “/” path where as the doPost method is used to handle the Get HTTP request of the home path.

```java
Home.java
package com.bakeease.controller;

import jakarta.servlet.ServletException;

/**
 * Servlet implementation class Home
 * Handles requests for the home page.
 *
 * @author
 * Piyush Karn
 */
@WebServlet(asyncSupported = true, urlPatterns = {"/home", "/"})
public class Home extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Handles GET requests and forwards to home.jsp.
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        request.getRequestDispatcher("WEB-INF/pages/home.jsp").forward(request, response);
    }

    /**
     * Handles POST requests by retrieving form data and forwarding to home.jsp.
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Example: Processing form inputs
        String username = request.getParameter("username");
        String message = request.getParameter("message");

        // Setting attributes to send back to JSP
        request.setAttribute("username", username);
        request.setAttribute("message", message);

        // Forwarding to home.jsp with attributes
        request.getRequestDispatcher("WEB-INF/pages/home.jsp").forward(request, response);
    }
}
```

Figure 48: code for doGet and doPost

4.2.6. Login.java controller

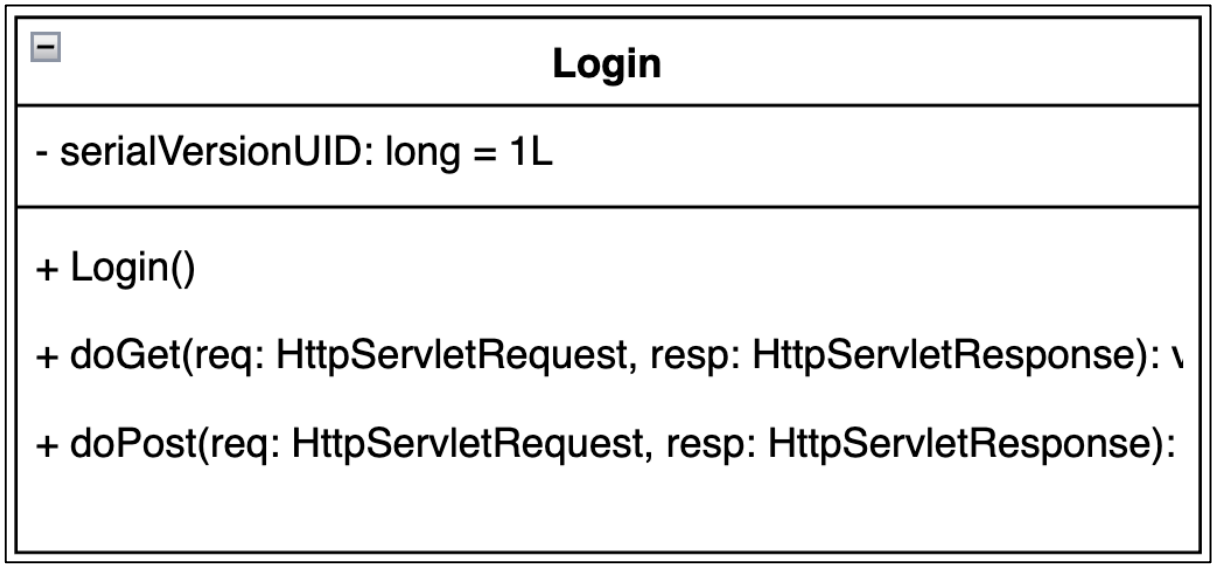


Figure 49: class diagram of login controller

The login controller class is annotated by /login path. It is a servlet file. The doGet method is used to load login.jsp file in the /login path and the doPost method is used to handle login operations. Its access modifier is protected and returns invalid response. The doPost method accepts username and password from the web page and authenticates using service package. Then returns successful redirection if the credentials match, else returns error message if invalid credentials are found. If the credentials match with the customer role, it redirects to home path else if the credentials match with the admin role, it redirects to admin path.

```java
package com.bakeease.controller;

import com.bakeease.service.LoginService;

@WebServlet("/login")
public class Login extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private final LoginService loginService = new LoginService();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");

        String username = request.getParameter("username");
        String inputPassword = request.getParameter("password");

        String role = loginService.authenticate(username, inputPassword);

        if (role != null) {
            // Login successful
            HttpSession session = request.getSession();
            session.setAttribute("username", username);

            // Set role cookie
            CookieUtil.addCookie(response, "role", role, 60 * 60); // 1 hour validity

            if ("admin".equalsIgnoreCase(role)) {
                response.sendRedirect(request.getContextPath() + "/admin");
            } else {
                response.sendRedirect(request.getContextPath() + "/home");
            }
            return;
        }

        // Login failed
        request.setAttribute("error", "Invalid username or password.");
        request.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(request, response);
    }
}
```

Figure 50: code for doGet and doPost

4.2.7. Product.java controller

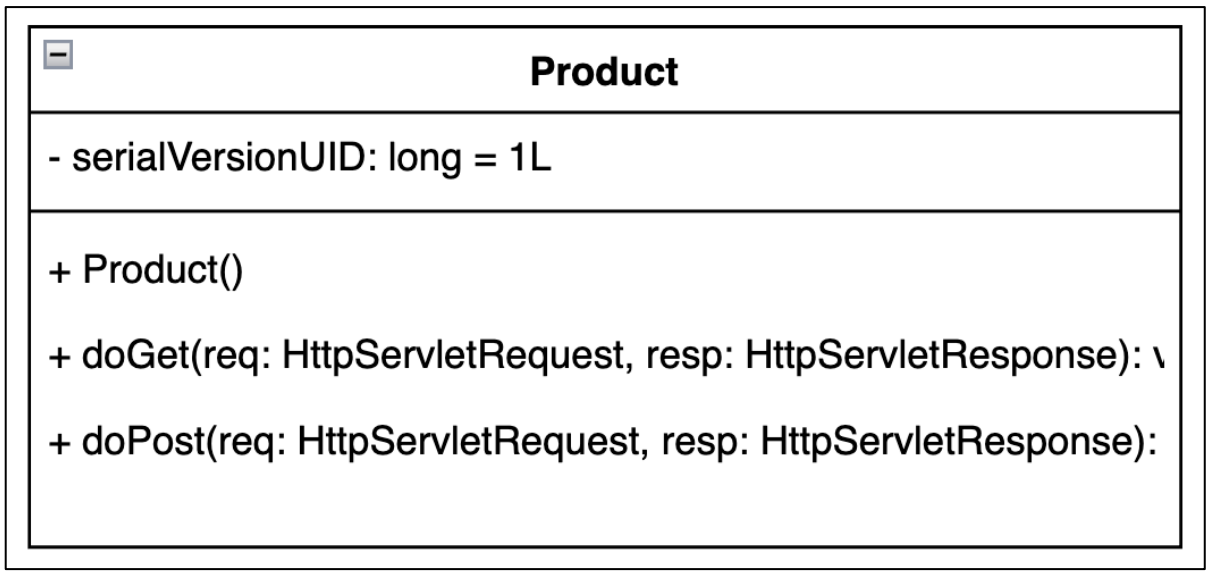


Figure 51: class diagram of product controller

The product controller file is triggered when the /product path is entered in the web page. The controller contains doGet and doPost methods. The doGet method is used to handle GET requests for displaying all the products which are in the database to be

displayed in the product page. It loads the product.jsp file in the web page where all the products from the database is displayed. It extracts the select query from service package and display the products in the web page.

Product.java

```
1  package com.bakeease.controller;
2
3  import com.bakeease.model.ProductModel;
13
14 @WebServlet("/product")
15 public class Product extends HttpServlet {
16     private static final long serialVersionUID = 1L;
17
18     private final ProductService productService = new ProductService();
19
20     /**
21      * Handles GET requests for displaying all or searched products.
22      */
23     @Override
24     protected void doGet(HttpServletRequest request, HttpServletResponse response)
25             throws ServletException, IOException {
26         String query = request.getParameter("query");
27         List<ProductModel> products;
28
29         if (query != null && !query.trim().isEmpty()) {
30             products = productService.searchProducts(query.trim());
31         } else {
32             products = productService.getAllProducts();
33         }
34
35         request.setAttribute("products", products);
36         request.setAttribute("query", query); // Optional: used in JSP for pre-filling the search box
37         request.getRequestDispatcher("/WEB-INF/pages/product.jsp").forward(request, response);
38     }
39
```

Figure 52: code for doGet method

The doPost method is used to handle POST requests in the web page. The doPost method contains operations for adding, updating and deleting the product inside try catch blocks where in try block it contains add, update and delete operation and inside catch block it contains exception which returns sendError function.

Product.java

```
36          request.setAttribute("query", query); // Optional: used in JSP for pre-filling the search box
37          request.getRequestDispatcher("/WEB-INF/pages/product.jsp").forward(request, response);
38      }
39
40      /**
41       * Handles POST requests for product operations: add, update, delete.
42       */
43      @Override
44      protected void doPost(HttpServletRequest request, HttpServletResponse response)
45              throws ServletException, IOException {
46          String action = request.getParameter("action");
47
48          try {
49              if ("add".equals(action)) {
50                  String name = request.getParameter("name");
51                  String description = request.getParameter("description");
52                  double price = Double.parseDouble(request.getParameter("price"));
53
54                  ProductModel newProduct = new ProductModel();
55                  newProduct.setName(name);
56                  newProduct.setDescription(description);
57                  newProduct.setPrice(price);
58
59                  productService.addProduct(newProduct);
60              } else if ("update".equals(action)) {
61                  int id = Integer.parseInt(request.getParameter("id"));
62                  String name = request.getParameter("name");
63                  String description = request.getParameter("description");
64                  double price = Double.parseDouble(request.getParameter("price"));
65
66                  ProductModel product = new ProductModel();
67                  product.setId(id);
68                  product.setName(name);
69                  product.setDescription(description);
70                  product.setPrice(price);
71
72                  productService.updateProduct(product);
73              } else if ("delete".equals(action)) {
74                  int id = Integer.parseInt(request.getParameter("id"));
75                  productService.deleteProduct(id);
76              }
77
78              // After operation, redirect back to product list
79              response.sendRedirect("product");
80          } catch (Exception e) {
81              e.printStackTrace();
82              response.sendError(HttpServletResponse.SC_BAD_REQUEST, "An error occurred while processing the product.");
83          }
84      }
85  }
```

Figure 53: code for doPost method

4.2.8. Profile.java controller

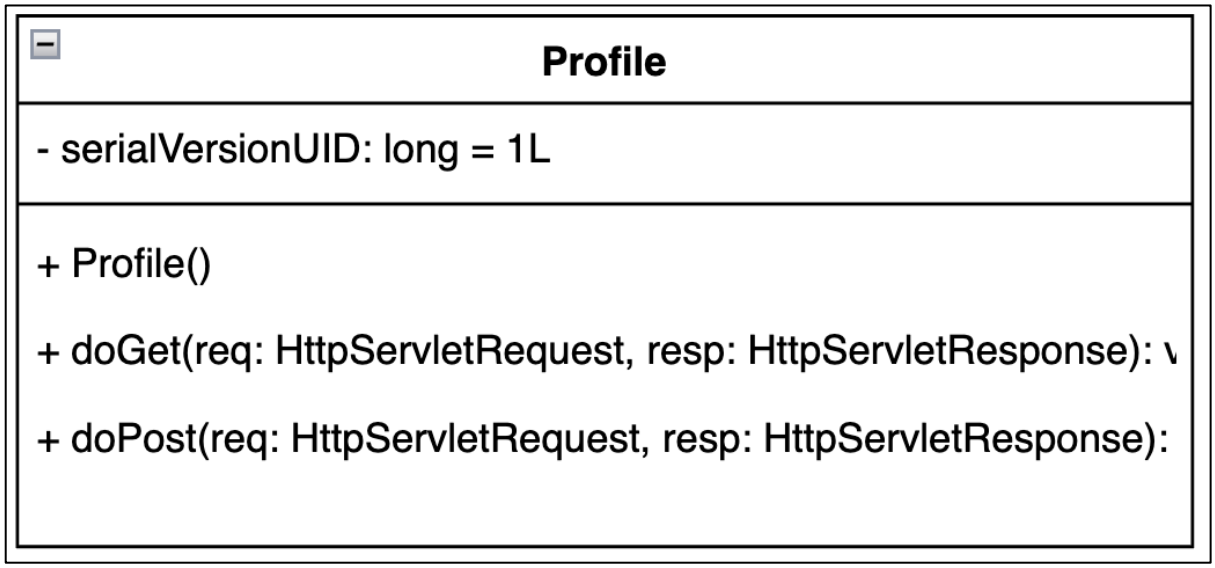


Figure 54: class diagram of profile controller

The profile controller is annotated by /profile path and also contains multi part config to handle image utility. The doGet method of this class is used to access session, determine role, and return null is user or session is not found in the profile.jsp file in the web page.

Profile.java

```
1  package com.bakeease.controller;
2
3  import com.bakeease.model.UserModel;
12
13 @WebServlet("/profile")
14 @MultipartConfig
15 public class Profile extends HttpServlet {
16     private static final long serialVersionUID = 1L;
17
18     private final UserService userService = new UserService();
19
20     @Override
21     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
22         HttpSession session = request.getSession(false);
23
24         if (session == null || session.getAttribute("username") == null) {
25             response.sendRedirect(request.getContextPath() + "/login");
26             return;
27         }
28
29         String username = (String) session.getAttribute("username");
30         UserModel user = userService.getUserByUsername(username);
31
32         if (user != null) {
33             request.setAttribute("user", user);
34             request.getRequestDispatcher("/WEB-INF/pages/profile.jsp").forward(request, response);
35         } else {
36             request.setAttribute("error", "User not found.");
37             request.getRequestDispatcher("/WEB-INF/pages/error.jsp").forward(request, response);
38         }
39     }
40
```

Figure 55: code for doGet method

The doPost method is used to handle POST requests from the web page. The method is used to update the user credentials which is fetched from the database and display successful or error message when profile is not updated.

```
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        try {
            boolean updated = UpdateProfile.updateUser(request);
            if (updated) {
                request.setAttribute("success", "Profile updated successfully.");
            } else {
                request.setAttribute("error", "Profile update failed or user not logged in.");
            }
        } catch (Exception e) {
            e.printStackTrace();
            request.setAttribute("error", "An error occurred: " + e.getMessage());
        }

        // Refresh updated user info for display
        doGet(request, response);
    }
}
```

Figure 56: code for doPost method

4.2.9. Register.java controller

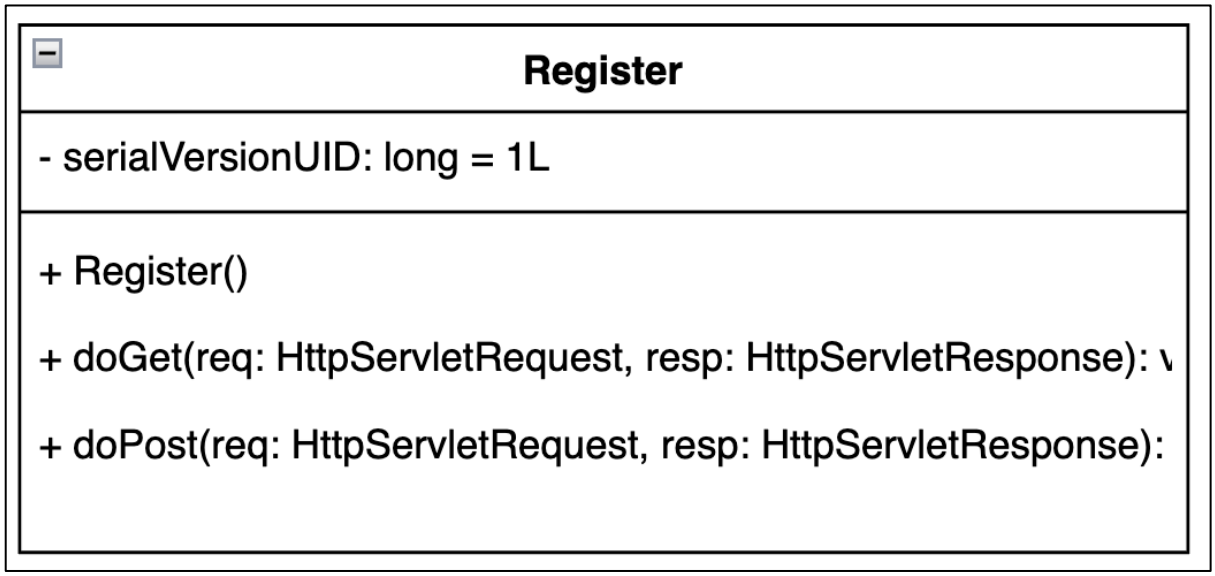


Figure 57: class diagram of register controller

The register controller is annotated by /register path and using the multipartconfig it the image size threshold is set to maximum 50 mb. The doGet method is used to handle GET requests which involves serving the register.jsp file to the web page. Its access modifier is protected and it returns servlet response.

```
Register.java
34
35      @Override
36      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
37          request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
38      }
39
```

Figure 58: code for doGet method

The doPost method is used to handle POST requests and contains validation as well as registration method for each text fields in the web page. The validationutil file is used to handle validation in the text fields. The controller also checks for duplicate entries through email or username. Upon successful user registration, it redirects to /login path else remains in the same page showing error messages to the user.

Register.java

```java
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    request.setCharacterEncoding("UTF-8");

    String username = request.getParameter("username");
    String email = request.getParameter("email");
    String password = request.getParameter("password");
    String confirmPassword = request.getParameter("confirmPassword");
    String phone = request.getParameter("phone");
    Part image = request.getPart("image");

    boolean isValid = true;

    // Validation checks
    if (Validationutil.isNullOrEmpty(username)) {
        request.setAttribute("usernameError", "Username is required.");
        isValid = false;
    } else if (!username.matches("^[a-zA-Z][a-zA-Z0-9_]{2,19}$")) {
        request.setAttribute("usernameError", "Username must start with a letter and be 3-20 characters long.");
        isValid = false;
    }

    if (Validationutil.isNullOrEmpty(email)) {
        request.setAttribute("emailError", "Email is required.");
        isValid = false;
    } else if (!email.matches("^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$")) {
        request.setAttribute("emailError", "Please enter a valid email address.");
        isValid = false;
    }

    if (Validationutil.isNullOrEmpty(password)) {
        request.setAttribute("passwordError", "Password is required.");
        isValid = false;
    } else if (password.length() < 6) {
        request.setAttribute("passwordError", "Password must be at least 6 characters long.");
        isValid = false;
    }

    if (Validationutil.isNullOrEmpty(confirmPassword)) {
        request.setAttribute("confirmPasswordError", "Confirm Password is required.");
        isValid = false;
    } else if (!password.equals(confirmPassword)) {
        request.setAttribute("confirmPasswordError", "Password and Confirm Password do not match.");
        isValid = false;
    }

    if (Validationutil.isNullOrEmpty(phone)) {
        request.setAttribute("phoneError", "Phone number is required.");
        isValid = false;
    } else if (!phone.matches("^98[0-9]{8}$")) {
        request.setAttribute("phoneError", "Phone number must start with '98' and be 10 digits.");
        isValid = false;
    }

    String fileName = null;
    if (image == null || image.getSize() == 0) {
        request.setAttribute("imageError", "Image field is empty. Please upload an image.");
```

Register.java

```
        if (Validationutil.isNullOrEmpty(phone)) {
            request.setAttribute("phoneError", "Phone number is required.");
            isValid = false;
        } else if (!phone.matches("^98[0-9]{8}$")) {
            request.setAttribute("phoneError", "Phone number must start with '98' and be 10 digits.");
            isValid = false;
        }

        String fileName = null;
        if (image == null || image.getSize() == 0) {
            request.setAttribute("imageError", "Image field is empty. Please upload an image.");
            isValid = false;
        } else if (!Validationutil.isValidImageExtension(image)) {
            request.setAttribute("imageError", "Only upload png, jpg and jpeg image.");
            isValid = false;
        } else {
            fileName = imageUtil.getImageNameFromPart(image);
            boolean uploadSuccess = imageUtil.uploadImage(image, getServletContext().getRealPath(""), "/customer-images");
            if (!uploadSuccess) {
                request.setAttribute("imageError", "Oops! Couldn't upload the image. Give it another shot.");
                isValid = false;
            }
        }

        // Check for duplicate entries
        if (isValid && registerService.isDuplicateUsernameOrEmail(username, email)) {
            request.setAttribute("duplicateError", "Duplicate entries found. Please choose another username and email.");
            isValid = false;
        }

        if (!isValid) {
            request.setAttribute("username", username);
            request.setAttribute("email", email);
            request.setAttribute("phone", phone);
            request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
            return;
        }

        String encryptedPassword = PasswordUtil.encrypt(username, password);
        if (encryptedPassword == null) {
            request.setAttribute("error", "Error encrypting password. Try again.");
            request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
            return;
        }

        String role = "customer";
        UserModel user = new UserModel(username, email, phone, encryptedPassword, role, fileName);

        boolean isRegistered = registerService.addUser(user);
        if (isRegistered) {
            response.sendRedirect(request.getContextPath() + "/login");
        } else {
            request.setAttribute("error", "Registration failed. Try again.");
            request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
        }
    }
}
```

Figure 59: code for doPost method

4.2.10. AuthenticationFilter.java Filter

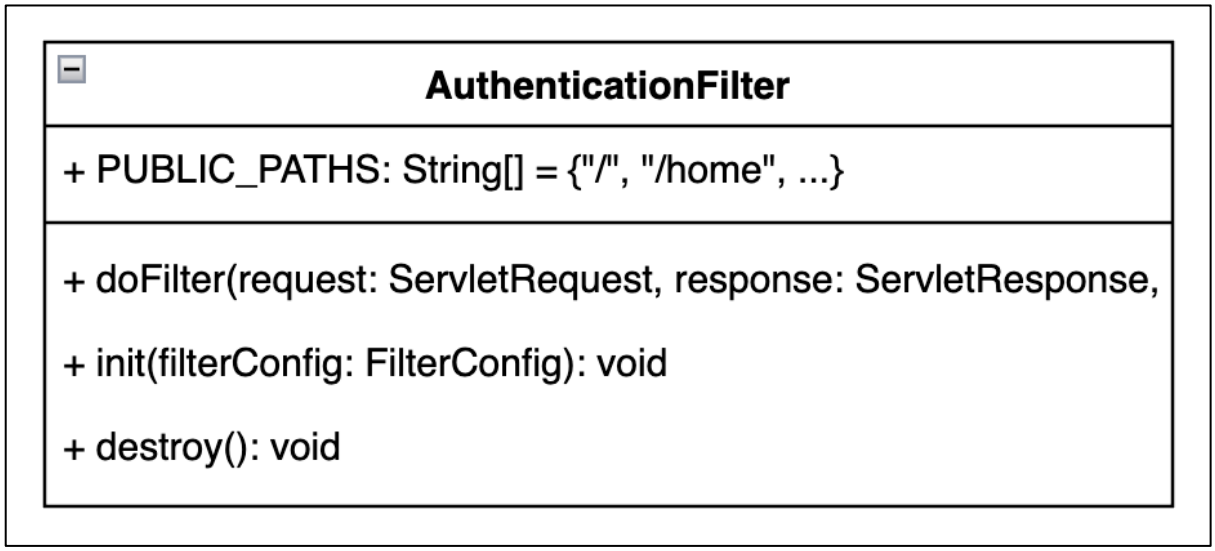


Figure 60: class diagram of authentication filter

The authentication filter file makes a gap difference between the admin, customer and non-logged in users. It allows the non-logged in users to access certain pages but doesn't allow the full access to the system. Similarly it allows certain access to the customer and allows full control of the system to the admin. It allows "/", "/home", "/about", "/login", "/register", "/contact", "/product" paths to non-logged in users, further the customer can access "/profile" path and the admin can access all the mentioned paths as well as the "/admin" path. Hence, it extracts the role from the cookie and provides access to the users.

AuthenticationFilter.java

```java
@WebFilter(asyncSupported = true, urlPatterns = "/*")
public class AuthenticationFilter implements Filter {

    private static final String[] PUBLIC_PATHS = {
        "/", "/home", "/about", "/login", "/register", "/contact", "/product"
    };

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        String uri = req.getRequestURI();

        boolean isLoggedIn = SessionUtil.getAttribute(req, "username") != null;
        String role = CookieUtil.getCookie(req, "role") != null
                ? CookieUtil.getCookie(req, "role").getValue()
                : null;

        // Allow static resources (CSS, JS, images, fonts, etc.)
        if (uri.matches(".*(\\.css|\\.js|\\.png|\\.jpg|\\.jpeg|\\.gif|\\.svg|\\.woff|\\.woff2|\\.ttf|\\.eot)$")) {
            chain.doFilter(request, response);
            return;
        }

        // Allow public paths to everyone
        for (String path : PUBLIC_PATHS) {
            if (uri.equals(req.getContextPath() + path) || uri.equals(path)) {
                chain.doFilter(request, response);
                return;
            }
        }

        // Admin-only path
        if (uri.equals(req.getContextPath() + "/admin")) {
            if (isLoggedIn && "admin".equals(role)) {
                chain.doFilter(request, response);
            } else {
                res.sendRedirect(req.getContextPath() + "/login");
            }
            return;
        }

        // Profile page – allow both admin and customer
        if (uri.equals(req.getContextPath() + "/profile")) {
            if (isLoggedIn && ("admin".equals(role) || "customer".equals(role))) {
                chain.doFilter(request, response);
            } else {
                res.sendRedirect(req.getContextPath() + "/login");
            }
            return;
        }

        // Default case – block and redirect to login
        res.sendRedirect(req.getContextPath() + "/login");
    }
```

Figure 61: doFilter code for authentication filter

4.2.11. ProductModel.java Model

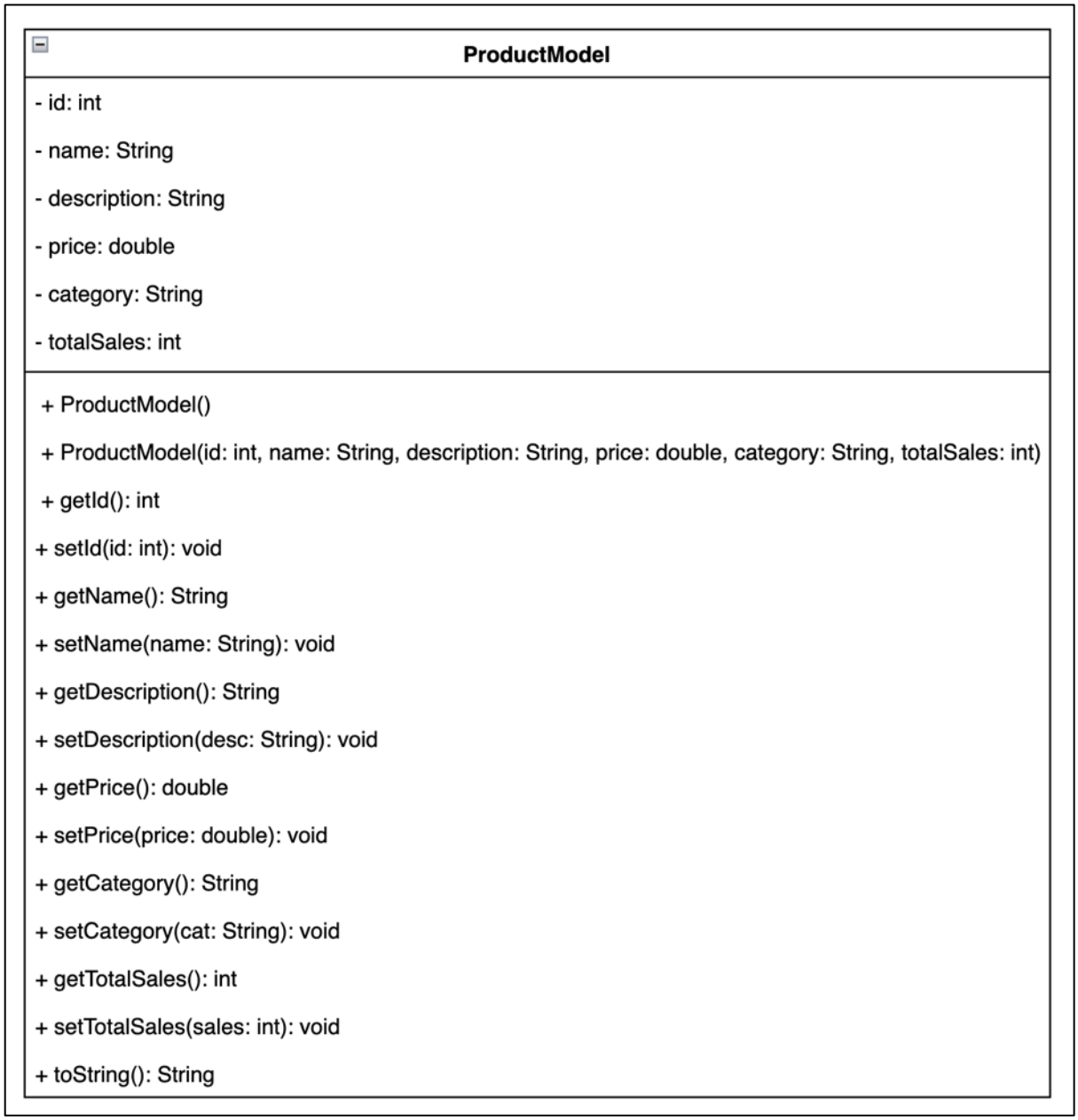


Figure 62: class diagram of product model

The product model class is a model class which contains the getter, setter and constructor method for the products including product id, product name, description, price, category and total sales.

ProductModel.java

```java
package com.bakeease.model;

public class ProductModel {
    private int id;
    private String name;
    private String description;
    private double price;
    private String category;
    private int totalSales;

    // Default Constructor
    public ProductModel() {}

    // Constructor with all parameters
    public ProductModel(int id, String name, String description, double price, String category, int totalSales) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.price = price;
        this.category = category;
        this.totalSales = totalSales;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }

    public String getCategory() { return category; }
    public void setCategory(String category) { this.category = category; }

    public int getTotalSales() { return totalSales; }
    public void setTotalSales(int totalSales) { this.totalSales = totalSales; }

    @Override
    public String toString() {
        return "ProductModel{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", description='" + description + '\'' +
                ", price=" + price +
                ", category='" + category + '\'' +
                ", totalSales=" + totalSales +
                '}';
    }
}
```

Figure 63: code of product model

4.2.12. UserModel.java Model

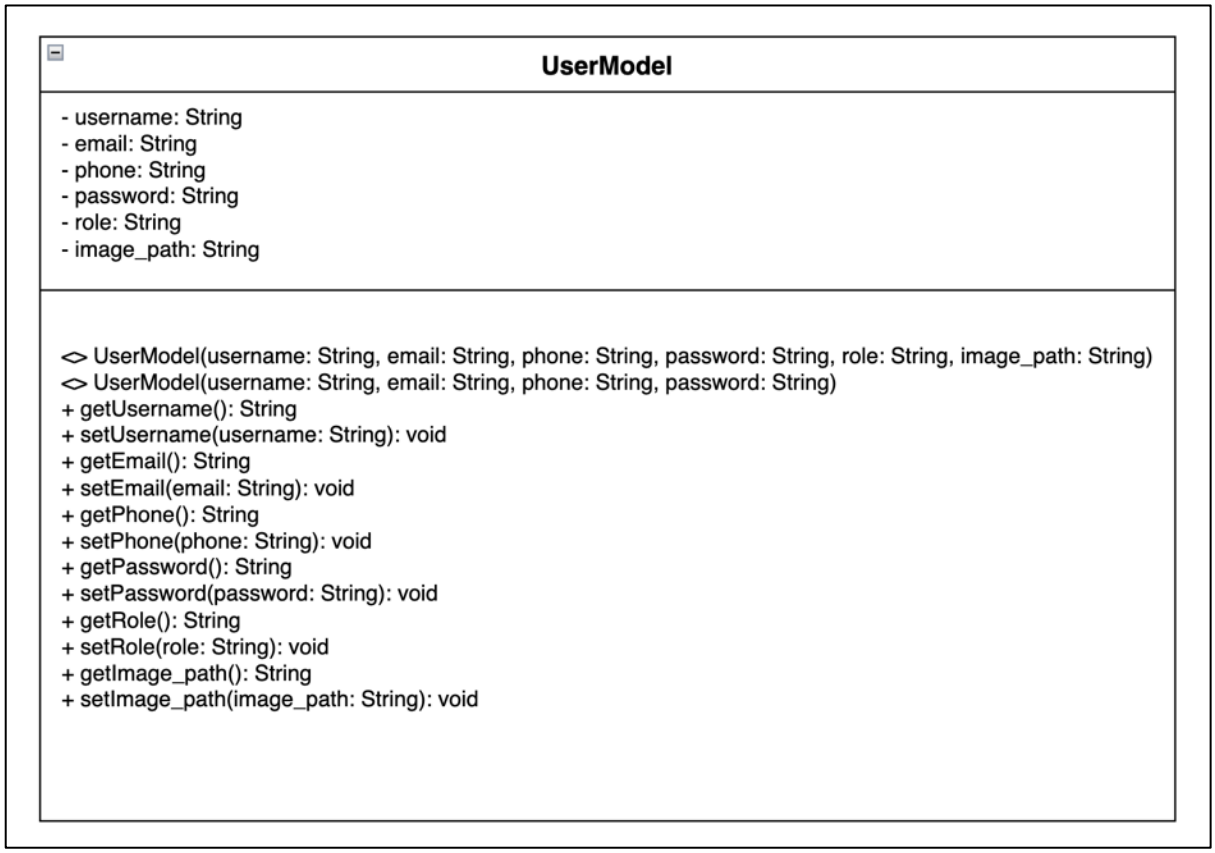


Figure 64: class diagram for user model

The user model is a model class file which contains getter, setter and constructors method for the user register, login and profile implementation. It contains methods for username, email, phone, password, role and image.

UserModel.java

```java
package com.bakeease.model;

/**
 * UserModel represents a user record that matches the database schema.
 */
public class UserModel {
    private String username;
    private String email;
    private String phone;
    private String password;
    private String role;
    private String image_path; // exact match with the DB column

    // Constructor with all fields
    public UserModel(String username, String email, String phone, String password, String role, String image_path) {
        this.username = username;
        this.email = email;
        this.phone = phone;
        this.password = password;
        this.role = role;
        this.image_path = image_path;
    }

    // Constructor without image_path and role (default role can be set later)
    public UserModel(String username, String email, String phone, String password) {
        this.username = username;
        this.email = email;
        this.phone = phone;
        this.password = password;
        this.role = "customer"; // default role
    }

    // Getters and Setters
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

```

Figure 65: code for user model

4.2.13. LoginService.java Service

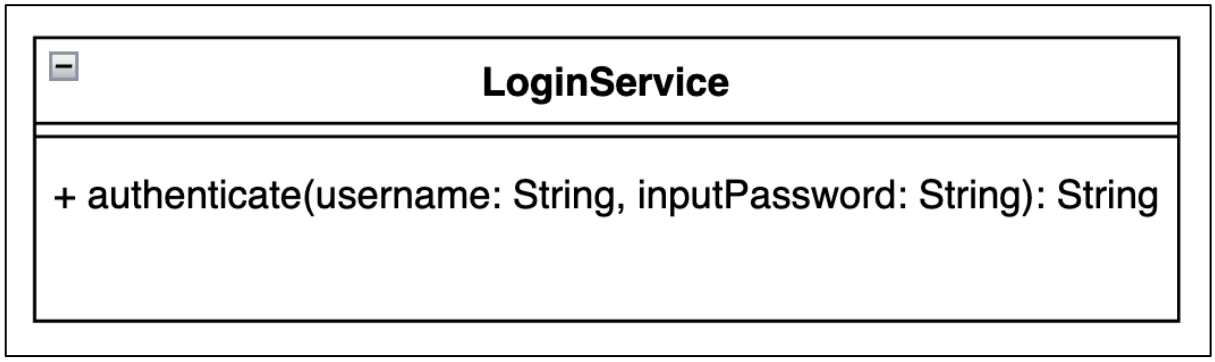


Figure 66: class diagram of login service

The login service class is responsible for authenticating user based on their username and role. The method authenticate is used to authenticate user based on their

username using select query in the database's user table. Its access modifier is public. The method is used to login the user into the system using their credentials and identify the role of the user and for logging in, it is required to decrypt the password and then proceed to logging in.

LoginService.java

```java
package com.bakeease.service;

import com.bakeease.config.DbConfig;
import com.bakeease.util.PasswordUtil;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class LoginService {

    public String authenticate(String username, String inputPassword) {
        String sql = "SELECT password, role FROM users WHERE username = ?";

        try (Connection conn = DbConfig.getDbConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, username);

            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    String encryptedPassword = rs.getString("password");
                    String role = rs.getString("role");

                    String decryptedPassword = PasswordUtil.decrypt(encryptedPassword, username);

                    if (decryptedPassword != null && decryptedPassword.equals(inputPassword)) {
                        return role; // Return role directly if authentication successful
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return null; // Authentication failed
    }
}
```

Figure 67: code for login service

4.2.14. ProductService.java Service

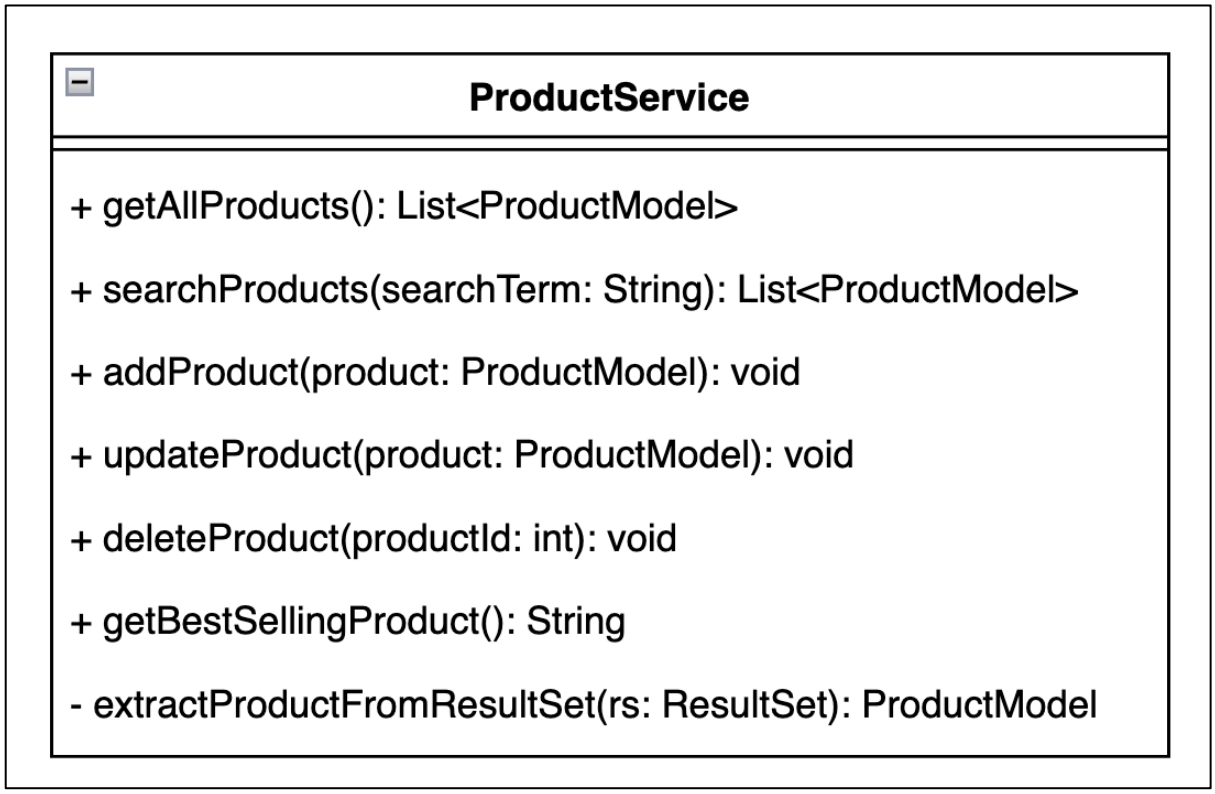


Figure 68: class diagram for product service

ProductService.java

```java
package com.bakeease.service;

import com.bakeease.model.ProductModel;

public class ProductService {

    // Method to get all products
    public List<ProductModel> getAllProducts() {
        List<ProductModel> products = new ArrayList<>();
        String query = "SELECT * FROM products";

        try (Connection conn = DbConfig.getDbConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {

            while (rs.next()) {
                ProductModel product = extractProductFromResultSet(rs);
                products.add(product);
            }

        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        return products;
    }
```

Figure 69: code for getAllProduct method

This method gets all product records from the database. It simply carries out a SELECT * FROM products SQL query statement, then which loops through the result set and converts each row into a ProductModel object by calling a helper method before adding it to a list that is returned. Normally, the method is used to retrieve all information on products.

ProductService.java

```java
    // Method to search products based on query
    public List<ProductModel> searchProducts(String searchTerm) {
        List<ProductModel> products = new ArrayList<>();
        String query = "SELECT * FROM products WHERE LOWER(name) LIKE ? OR LOWER(description) LIKE ? OR LOWER(category) LIKE ?";

        try (Connection conn = DbConfig.getDbConnection();
             PreparedStatement pstmt = conn.prepareStatement(query)) {

            String keyword = "%" + searchTerm.toLowerCase() + "%";
            pstmt.setString(1, keyword);
            pstmt.setString(2, keyword);
            pstmt.setString(3, keyword);

            try (ResultSet rs = pstmt.executeQuery()) {
                while (rs.next()) {
                    ProductModel product = extractProductFromResultSet(rs);
                    products.add(product);
                }
            }

        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        return products;
    }
```

Figure 70: code for search products method

This method performs case sensitive operation to search product. It converts the name of the product to lower case and then execute the select query in the database which selects the product name, description and category and returns the extracted data. Then the data is set to lower case to track the product information.

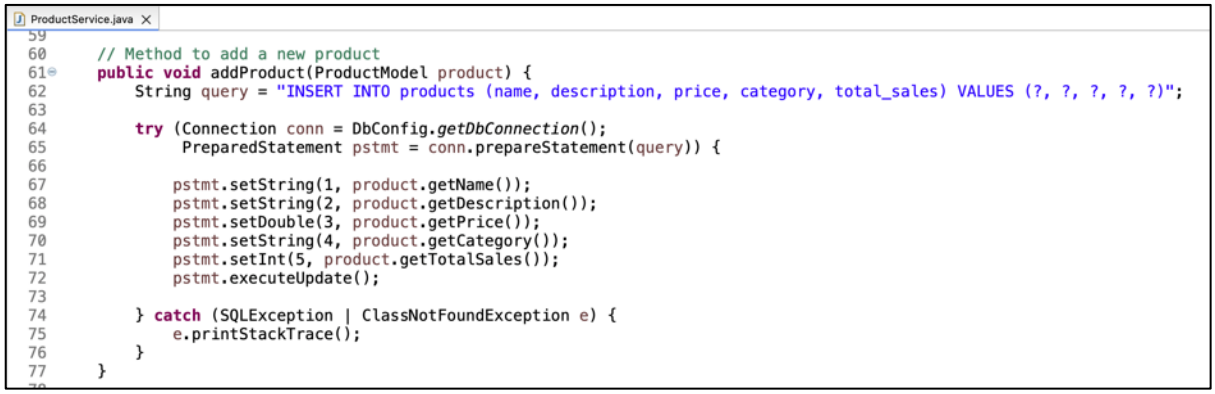


Figure 71: code for add products

This method is used to add a product to the database. It uses insert query to add the product to the database and it takes the object of model class as parameter and requires the getter method of name, description, price, category and total sales from the model class.

Figure 72: code for update product

This method updates the information of the existing product in the database. It also requires object of model class as parameter. This method also requires the getter method of name, description, price, category and total sales from the model class to get product information and then uses update query in the database to update product information.

```java
// Method to delete a product by ID
public void deleteProduct(int productId) {
    String query = "DELETE FROM products WHERE id = ?";

    try (Connection conn = DbConfig.getDbConnection();
         PreparedStatement pstmt = conn.prepareStatement(query)) {

        pstmt.setInt(1, productId);
        pstmt.executeUpdate();

    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

Figure 73: code to delete product

This method is used to delete the product from the database using product id. It executes delete query to delete product from the database.

```java
        PreparedStatement pstmt = conn.prepareStatement(query)) {

        pstmt.setInt(1, productId);
        pstmt.executeUpdate();

    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}

// Method to get the name of the best-selling product
public String getBestSellingProduct() {
    String query = "SELECT name FROM products ORDER BY total_sales DESC LIMIT 1";

    try (Connection conn = DbConfig.getDbConnection();
         PreparedStatement pstmt = conn.prepareStatement(query);
         ResultSet rs = pstmt.executeQuery()) {

        if (rs.next()) {
            return rs.getString("name");
        }

    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    return null;
}
```

Figure 74: method to identify the best selling product

This method is used to identify the best-selling product by ordering the total sales which means identifying the product which has the highest number of sales then declares that product as highest selling product.

4.2.15. RegisterService.java Service

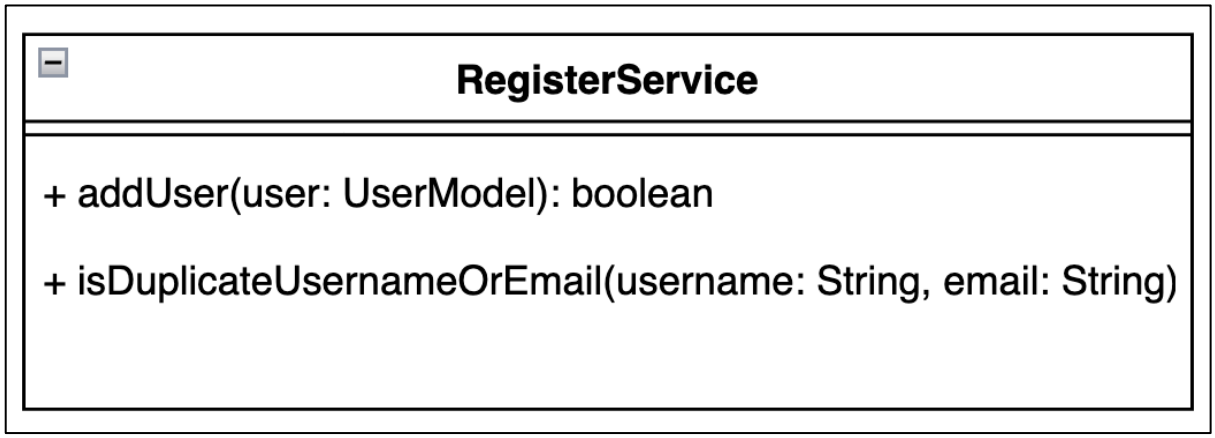


Figure 75: class diagram of register service

This class contains two methods addUser and isDuplicateUsernameOrEmail. Futher the methods are discussed below.

```
RegisterService.java
2
3  import com.bakeease.model.UserModel;
9
10 public class RegisterService {
11
12     public boolean addUser(UserModel user) {
13         boolean isInserted = false;
14
15         String sql = "INSERT INTO users (username, email, phone, password, role, image_path) VALUES (?, ?, ?, ?, ?, ?)";
16
17         try (Connection conn = DbConfig.getDbConnection();
18              PreparedStatement ps = conn.prepareStatement(sql)) {
19
20             ps.setString(1, user.getUsername());
21             ps.setString(2, user.getEmail());
22             ps.setString(3, user.getPhone());
23             ps.setString(4, user.getPassword());
24             ps.setString(5, user.getRole());
25             ps.setString(6, user.getImage_path());
26
27             int rows = ps.executeUpdate();
28             isInserted = rows > 0;
29
30             System.out.println("Rows inserted: " + rows);
31
32         } catch (Exception e) {
33             e.printStackTrace();
34         }
35
36         return isInserted;
37     }
```

Figure 76: code for add user

This method is used to add a new user by entering their information into the users table of the database. It receives a UserModel object, retrieves its attributes which are username, email, phone, password, role, and image_path and adds them through a insert query. It returns true when the insertion succeeds otherwise it returns false.

```
    // METHOD to check duplicate username/email
    public boolean isDuplicateUsernameOrEmail(String username, String email) {
        String sql = "SELECT COUNT(*) FROM users WHERE username = ? OR email = ?";
        try (Connection conn = DbConfig.getDbConnection();
             PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, username);
            ps.setString(2, email);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt(1) > 0;
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }
```

Figure 77: code for checking duplicate entries

The method is used to check in the database whether a user with the given username or email already exists. Running select count query with the given username and email as parameters, if the count is greater than zero, then it will return true, implying a duplicate entry of the user, else false. It ensures uniqueness in user registration.

4.2.16. UserService.java Service

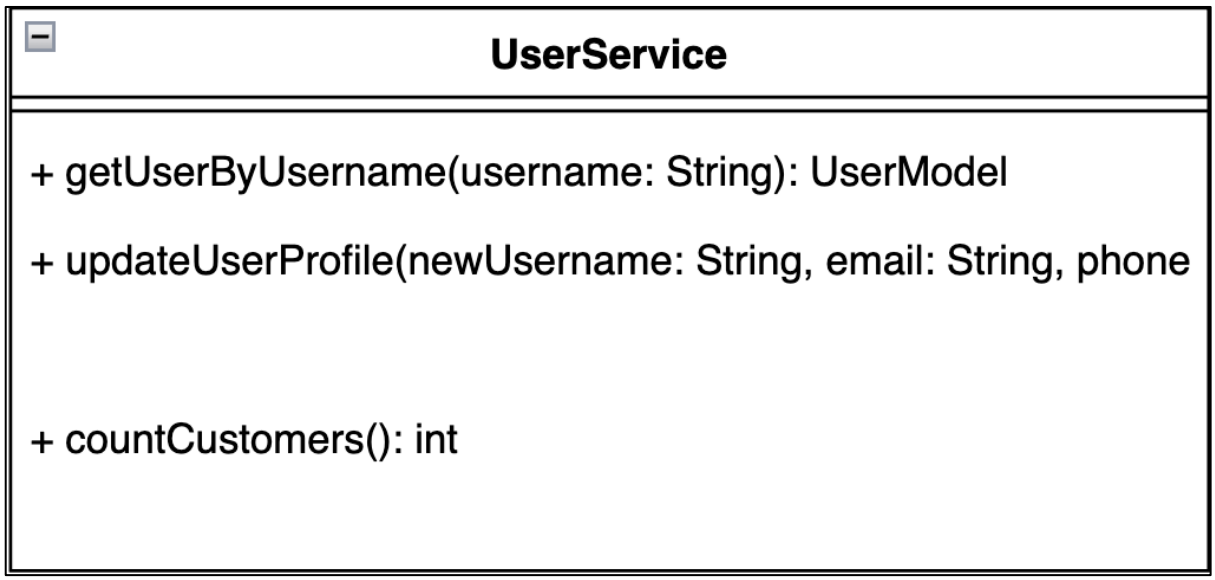


Figure 78: class diagram of user service

This class contains 3 methods which are further discussed below.

UserService.java

```
1  package com.bakeease.service;
2
3  import com.bakeease.config.DbConfig;
16
17 public class UserService {
18
19     // Get user by username
20     public UserModel getUserByUsername(String username) {
21         UserModel user = null;
22         String query = "SELECT * FROM users WHERE username = ?";
23
24         try (Connection conn = DbConfig.getDbConnection();
25              PreparedStatement stmt = conn.prepareStatement(query)) {
26
27             stmt.setString(1, username);
28             ResultSet rs = stmt.executeQuery();
29
30             if (rs.next()) {
31                 user = new UserModel(
32                         rs.getString("username"),
33                         rs.getString("email"),
34                         rs.getString("phone"),
35                         rs.getString("password"),
36                         rs.getString("role"),
37                         rs.getString("image_path")
38                 );
39             }
40         } catch (Exception e) {
41             e.printStackTrace();
42         }
43
44         return user;
45     }
```

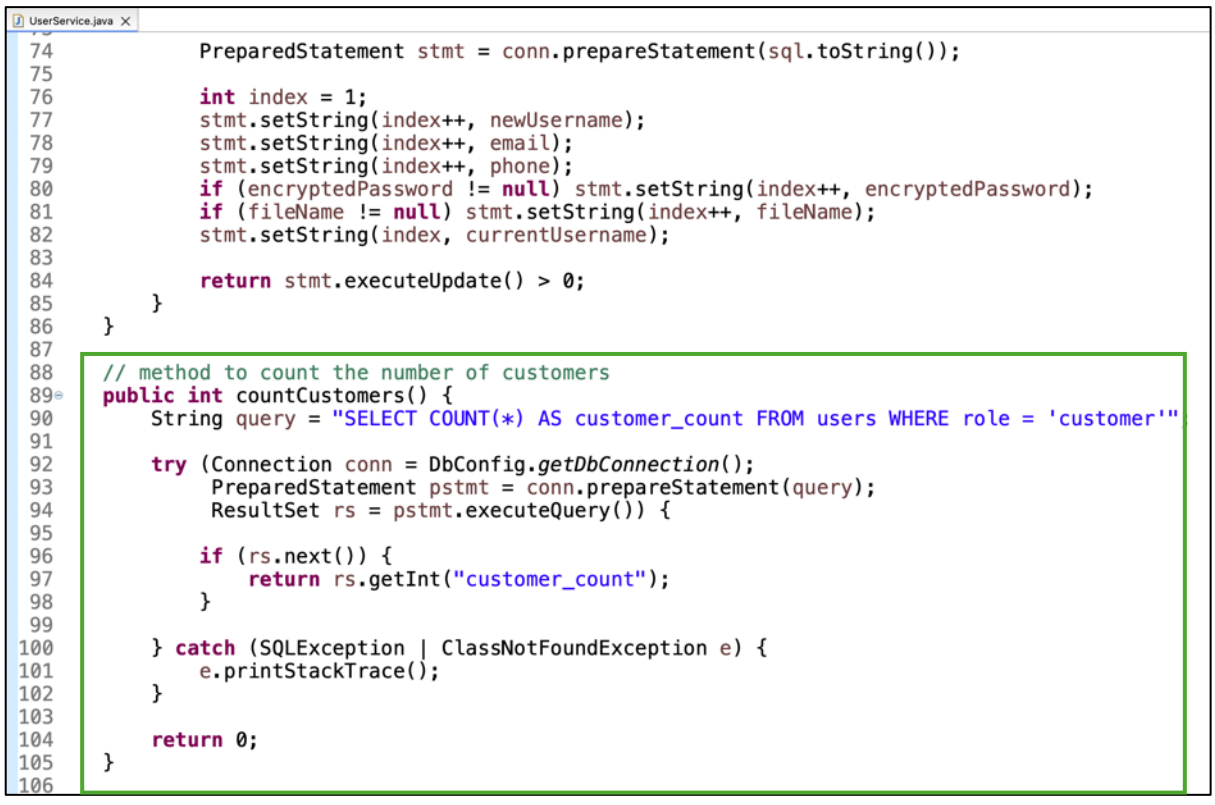
Figure 79: code for get user by username

This method is used to retrieve user information by their username. It requires username as parameter and select query is executed in the database and then user credentials involving username, email, password, phone number, role and image is retrieved.

```java
// Update user profile
public boolean updateUserProfile(String newUsername, String email, String phone, String password, Part imagePart, String currentUsername, String uploadPath)
        throws SQLException, IOException, ClassNotFoundException {

    String encryptedPassword = null;
    if (password != null && !password.trim().isEmpty()) {
        encryptedPassword = PasswordUtil.encrypt(newUsername, password);
    }

    String fileName = null;
    if (imagePart != null && imagePart.getSize() > 0) {
        fileName = Paths.get(imagePart.getSubmittedFileName()).getFileName().toString();
        File uploadsDir = new File(uploadPath);
        if (!uploadsDir.exists()) uploadsDir.mkdirs();

        File file = new File(uploadsDir, fileName);
        try (InputStream input = imagePart.getInputStream()) {
            Files.copy(input, file.toPath(), StandardCopyOption.REPLACE_EXISTING);
        }
    }

    try (Connection conn = DbConfig.getDbConnection()) {
        StringBuilder sql = new StringBuilder("UPDATE users SET username = ?, email = ?, phone = ?");
        if (encryptedPassword != null) sql.append(", password = ?");
        if (fileName != null) sql.append(", image_path = ?");
        sql.append(" WHERE username = ?");

        PreparedStatement stmt = conn.prepareStatement(sql.toString());

        int index = 1;
        stmt.setString(index++, newUsername);
        stmt.setString(index++, email);
        stmt.setString(index++, phone);
        if (encryptedPassword != null) stmt.setString(index++, encryptedPassword);
        if (fileName != null) stmt.setString(index++, fileName);
        stmt.setString(index, currentUsername);

        return stmt.executeUpdate() > 0;
    }
}
```

Figure 80: code for update user

This method is used to update user information including the username, email, password, phone number and profile image. It also encrypts the password using the password utility and executes the update query in the database to finalize the updated information to the database.



```java
        PreparedStatement stmt = conn.prepareStatement(sql.toString());

        int index = 1;
        stmt.setString(index++, newUsername);
        stmt.setString(index++, email);
        stmt.setString(index++, phone);
        if (encryptedPassword != null) stmt.setString(index++, encryptedPassword);
        if (fileName != null) stmt.setString(index++, fileName);
        stmt.setString(index, currentUsername);

        return stmt.executeUpdate() > 0;
    }
}

// method to count the number of customers
public int countCustomers() {
    String query = "SELECT COUNT(*) AS customer_count FROM users WHERE role = 'customer'";

    try (Connection conn = DbConfig.getDbConnection();
         PreparedStatement pstmt = conn.prepareStatement(query);
         ResultSet rs = pstmt.executeQuery()) {

        if (rs.next()) {
            return rs.getInt("customer_count");
        }

    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    return 0;
}
```

Figure 81: code to calculate the number of customers

This method is used to count the number of customers which are registered in the system. It uses select count query and selects only the users with role as “customer” and if the user is found then the counter is increased by 1. Lastly, the final number of customers are retrieved which is used by admins for their system analysis.

4.2.17. CookieUtil.java Util

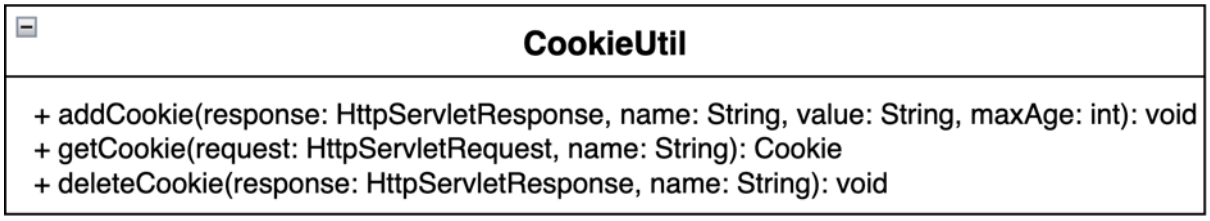


Figure 82: class diagram for cookie util

This class contains 3 methods which are discussed below.

```java
package com.bakeease.util;

import jakarta.servlet.http.Cookie;

public class CookieUtil {

    public static void addCookie(HttpServletResponse response, String name, String value, int maxAge) {
        Cookie cookie = new Cookie(name, value);
        cookie.setMaxAge(maxAge);
        cookie.setPath("/");
        response.addCookie(cookie);
    }
```

Figure 83: code for add cookie

This method adds a new cookie to the HTTP response with the specified name, value, and maximum age. The cookie is set to be available across the entire domain “/” path which is the home path of the system. It is further used to allocate role to the user.

```java
    public static Cookie getCookie(HttpServletRequest request, String name) {
        if (request.getCookies() != null) {
            return Arrays.stream(request.getCookies())
                    .filter(cookie -> name.equals(cookie.getName()))
                    .findFirst()
                    .orElse(null);
        }
        return null;
    }
```

Figure 84: code for get cookie

This method is used to get the cookie by its name and returns null if not found.

```
public static void deleteCookie(HttpServletResponse response, String name) {
    Cookie cookie = new Cookie(name, null);
    cookie.setMaxAge(0);
    cookie.setPath("/");
    response.addCookie(cookie);
}
```

Figure 85: code for delete cookie

This method is used to delete the existing cookie in "/" path. It executes when the system implements logout method but in our system it is not used but will be used in the further operation.

4.2.18. ImageUtil.java Util

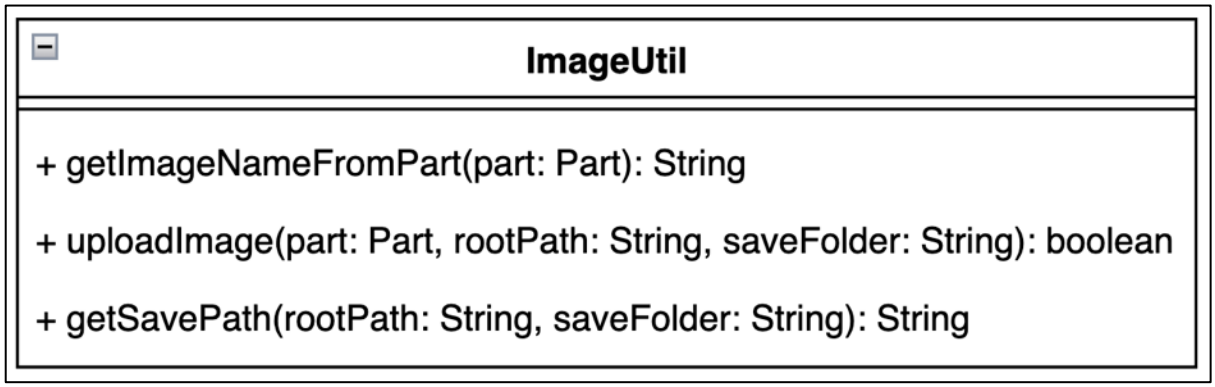


Figure 86: class diagram for image util

This class contains 3 methods which are used in extracting file name from a part. Further the methods are discussed below.

ImageUtil.java

```java
package com.bakeease.util;

import java.io.File;

/**
 * Utility class for handling image file uploads.
 * This class provides methods for extracting the file name from a Part
 * object and uploading the image file to a specified directory on the server.
 */
public class ImageUtil {

    /**
     * Extracts the file name from the given Part object based on the
     * "content-disposition" header.
     *
     * @param part the Part object representing the uploaded file.
     * @return the extracted file name. If no filename is found, returns a default
     *         name "download.png".
     */
    public String getImageNameFromPart(Part part) {
        // Retrieve the content-disposition header from the part
        String contentDisp = part.getHeader("content-disposition");

        // Split the header by semicolons to isolate key-value pairs
        String[] items = contentDisp.split(";");

        // Initialize imageName variable to store the extracted file name
        String imageName = null;

        // Iterate through the items to find the filename
        for (String s : items) {
            if (s.trim().startsWith("filename")) {
                // Extract the file name from the header value
                imageName = s.substring(s.indexOf("=") + 2, s.length() - 1);
            }
        }

        // Check if the filename was not found or is empty
        if (imageName == null || imageName.isEmpty()) {
            // Assign a default file name if none was provided
            imageName = "download.png";
        }

        // Return the extracted or default file name
        return imageName;
    }

}
```

Figure 87: code for get image name

This method is used to extract file name of the given part of the image and returns the default name “download.png” if the name is not found.

ImageUtil.java

```
49
50      /**
51       * Uploads the image file from the given Part object to a specified
52       * directory on the server.
53       *
54       * @param part the Part object representing the uploaded image file.
55       * @param rootPath the root path of the project (usually comes from the servlet context).
56       * @param saveFolder the folder within the rootPath where the image will be saved.
57       * @return true if the file was successfully uploaded, false otherwise.
58       */
59      public boolean uploadImage(Part part, String rootPath, String saveFolder) {
60          String savePath = getSavePath(rootPath, saveFolder);
61          File fileSaveDir = new File(savePath);
62
63          // Ensure the directory exists
64          if (!fileSaveDir.exists()) {
65              if (!fileSaveDir.mkdirs()) {
66                  return false; // Failed to create the directory
67              }
68          }
69
70          try {
71              // Get the image name
72              String imageName = getImageNameFromPart(part);
73              // Create the file path
74              String filePath = savePath + imageName;
75              // Write the file to the server
76              part.write(filePath);
77              return true; // Upload successful
78          } catch (IOException e) {
79              e.printStackTrace(); // Log the exception
80              return false; // Upload failed
81          }
82      }
```

Figure 88: code for upload image

This method us used to initialize update image in our system. It takes image part, root path and save folder name as parameters. It returns true if image is successfully uploaded else returns false.

```
    /**
     * Returns the path to save the image based on the provided folder name.
     *
     * @param rootPath the root path of the application.
     * @param saveFolder the folder where images will be saved (e.g., "customer").
     * @return the full save path.
     */
    public String getSavePath(String rootPath, String saveFolder) {
        // Construct the save path by combining the root path and the folder
        return "/Users/piyushkarn/eclipse-workspace/BakeEase/src/main/webapp/resources/assets/"+saveFolder+"/";
    }
}
```

Figure 89: code for get save path

This method is used to return the uploaded image in the system specified path. In our context the image is returned inside “/Users/piyushkarn/eclipse-workspace/BakeEase/src/main/webapp/resources/assets/” path.

4.2.19. PasswordUtil.java Util

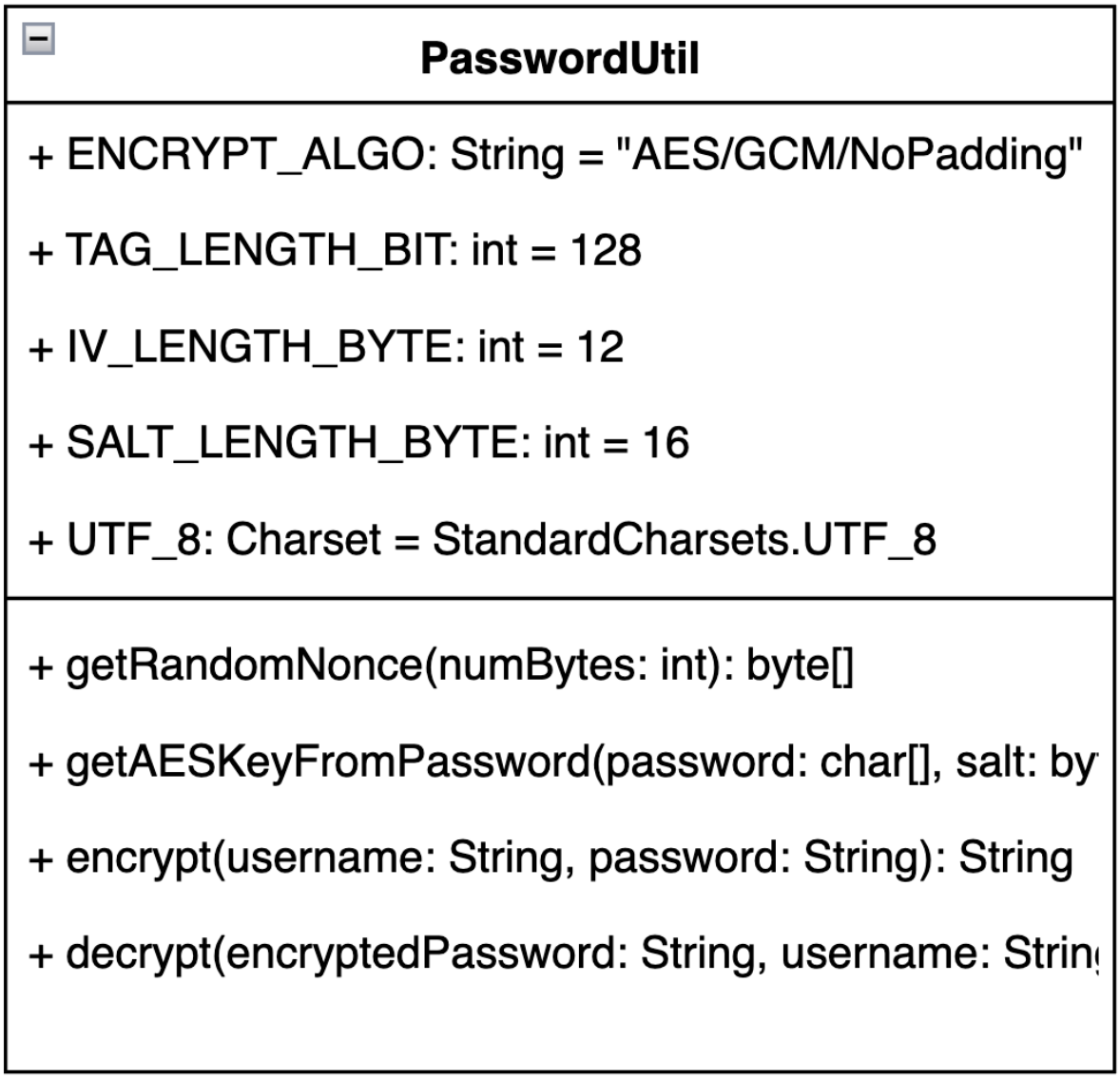


Figure 90: class diagram for password util

This class contains 4 major methods which are discussed below.

PasswordUtil.java

```
28      // Generates a random nonce (IV or salt)
29      public static byte[] getRandomNonce(int numBytes) {
30          byte[] nonce = new byte[numBytes];
31          new SecureRandom().nextBytes(nonce);
32          return nonce;
33      }
34
35      // Derives a SecretKey using PBKDF2 with HmacSHA256
36      public static SecretKey getAESKeyFromPassword(char[] password, byte[] salt) {
37          try {
38              SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
39              KeySpec spec = new PBEKeySpec(password, salt, 65536, 256);
40              return new SecretKeySpec(factory.generateSecret(spec).getEncoded(), "AES");
41          } catch (NoSuchAlgorithmException | InvalidKeySpecException ex) {
42              Logger.getLogger(PasswordUtil.class.getName()).log(Level.SEVERE, null, ex);
43              return null;
44          }
45      }
46
```

Figure 91: code for get random nonce and get AES key from password

The getRandomNonce method is used to generate and return a cryptographically secure random byte array (used as IV or salt) of the specified length and the method getAESKeyFromPassword is used to return an AES key and hash the password.

PasswordUtil.java

```
46
47      // Encrypts a password using a username-derived key and returns a base64 string
48      public static String encrypt(String username, String password) {
49          try {
50              byte[] salt = getRandomNonce(SALT_LENGTH_BYTE);
51              byte[] iv = getRandomNonce(IV_LENGTH_BYTE);
52
53              SecretKey aesKey = getAESKeyFromPassword(username.toCharArray(), salt);
54              Cipher cipher = Cipher.getInstance(ENCRYPT_ALGO);
55              cipher.init(Cipher.ENCRYPT_MODE, aesKey, new GCMParameterSpec(TAG_LENGTH_BIT, iv));
56
57              byte[] cipherText = cipher.doFinal(password.getBytes(UTF_8));
58              byte[] cipherTextWithIvSalt = ByteBuffer.allocate(iv.length + salt.length + cipherText.length)
59                      .put(iv)
60                      .put(salt)
61                      .put(cipherText)
62                      .array();
63
64              return Base64.getEncoder().encodeToString(cipherTextWithIvSalt);
65          } catch (Exception ex) {
66              Logger.getLogger(PasswordUtil.class.getName()).log(Level.SEVERE, null, ex);
67              return null;
68          }
69      }
70
```

Figure 92: code for encrypt password

This method is used to encrypt the user entered password using AES-GCM key derived from the username. The output looks like a bunch of characters placed randomly.

```
    // Decrypts an encrypted password using the original username as key source
    public static String decrypt(String encryptedPassword, String username) {
        try {
            byte[] decode = Base64.getDecoder().decode(encryptedPassword.getBytes(UTF_8));
            ByteBuffer bb = ByteBuffer.wrap(decode);

            byte[] iv = new byte[IV_LENGTH_BYTE];
            bb.get(iv);

            byte[] salt = new byte[SALT_LENGTH_BYTE];
            bb.get(salt);

            byte[] cipherText = new byte[bb.remaining()];
            bb.get(cipherText);

            SecretKey aesKey = getAESKeyFromPassword(username.toCharArray(), salt);
            Cipher cipher = Cipher.getInstance(ENCRYPT_ALGO);
            cipher.init(Cipher.DECRYPT_MODE, aesKey, new GCMParameterSpec(TAG_LENGTH_BIT, iv));

            byte[] plainText = cipher.doFinal(cipherText);
            return new String(plainText, UTF_8);
        } catch (Exception ex) {
            Logger.getLogger(PasswordUtil.class.getName()).log(Level.SEVERE, null, ex);
            return null;
        }
    }
```

Figure 93: code for decrypt password

This method is used to decrypt the encrypted password. It is used in login page and login password text field to decrypt the password of the user from the database. It takes encrypted password and username as parameters.

4.2.20. SessionUtil.java Util

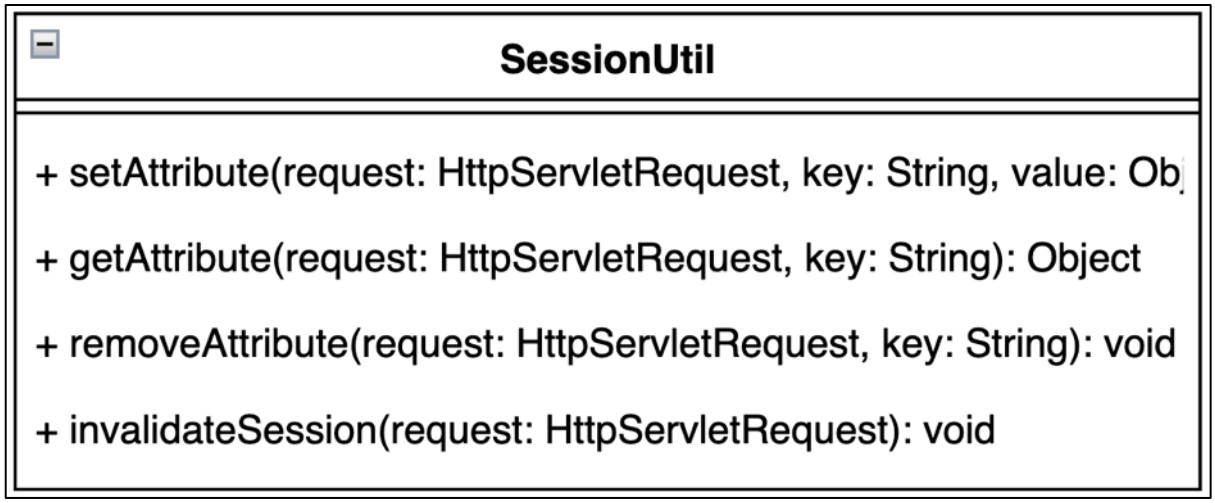


Figure 94: class diagram for session util

This class contains 4 methods setAttribute, getAttribute, removeAttribute and invalidateSession. Further the methods are discussed below.

SessionUtil.java

```java
package com.bakeease.util;

import jakarta.servlet.http.HttpServletRequest;

public class SessionUtil {

    public static void setAttribute(HttpServletRequest request, String key, Object value) {
        HttpSession session = request.getSession();
        session.setAttribute(key, value);
    }

```

Figure 95: code for set attribute method

This method setAttribute is used to set a attribute to the session with specified key and value and creates a new session if the session doesn't exist.

```java
public static Object getAttribute(HttpServletRequest request, String key) {
    HttpSession session = request.getSession(false);
    if (session != null) {
        return session.getAttribute(key);
    }
    return null;
}
```

Figure 96: code for get attribute

This method is used to retrieve the value of the specified session attributes or returns null if the session doesn't exist.

```java
public static void removeAttribute(HttpServletRequest request, String key) {
    HttpSession session = request.getSession(false);
    if (session != null) {
        session.removeAttribute(key);
    }
}
```

Figure 97: code for remove attribute

This method is used to remove the specified attribute from the current session if the current session exists.

```java
public static void invalidateSession(HttpServletRequest request) {
    HttpSession session = request.getSession(false);
    if (session != null) {
        session.invalidate();
    }
}
```

Figure 98: code for invalidate session

This method is used to invalidate the current session used in logging out the user and clearing user data.

4.2.21. UpdateProfile.java Util

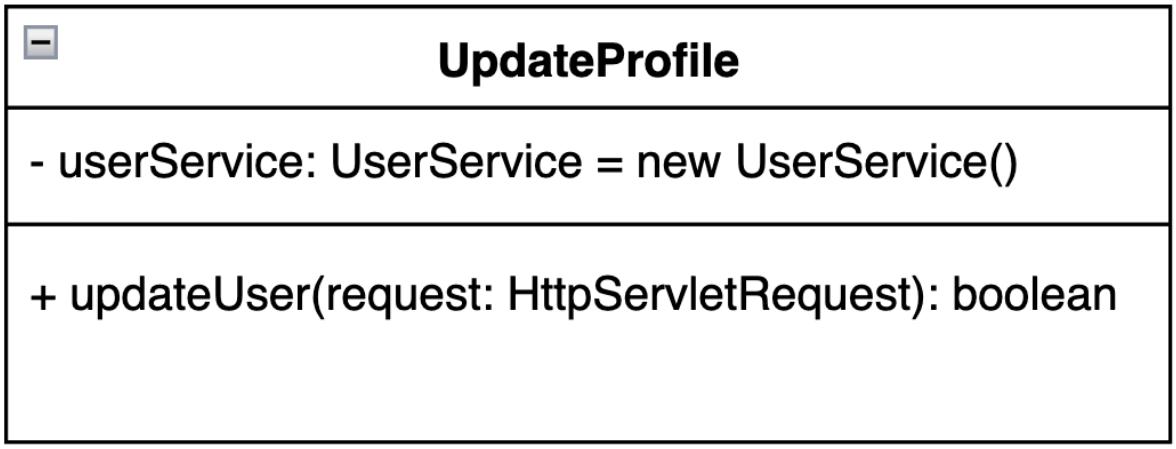


Figure 99: class diagram of update profile

The class contains only 1 method which is used to update the user credentials from the profile page. This method updates the username, email, password, phone number and profile image of the user from the profile page. It requires the getter methods from the model class and if the changes are successfully done, it sets the session attribute as the username of the user.

```java
package com.bakeease.util;

import com.bakeease.service.UserService;

public class UpdateProfile {

    private static final UserService userService = new UserService();

    public static boolean updateUser(HttpServletRequest request) throws Exception {
        String currentUsername = (String) SessionUtil.getAttribute(request, "username");

        if (currentUsername == null) {
            return false;
        }

        String newUsername = request.getParameter("username");
        String email = request.getParameter("email");
        String phone = request.getParameter("phone");
        String password = request.getParameter("password");
        Part imagePart = request.getPart("profileImage");

        String uploadPath = request.getServletContext().getRealPath("/") + File.separator + "resources" +
                File.separator + "assets" + File.separator + "customer-images";

        boolean success = userService.updateUserProfile(newUsername, email, phone, password, imagePart, currentUsername, uploadPath);

        if (success) {
            SessionUtil.setAttribute(request, "username", newUsername);
        }

        return success;
    }
}
```

Figure 100: code for update profile

4.2.22. ValidationUtil.java Util

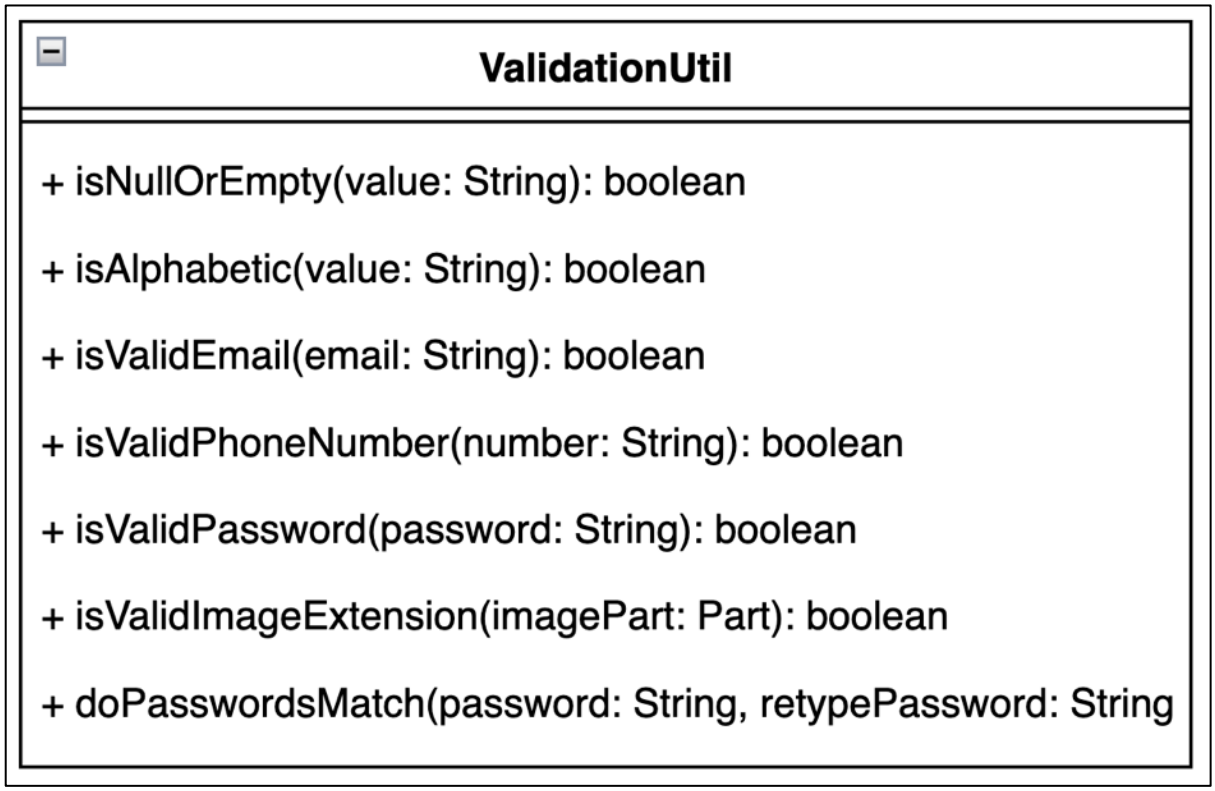


Figure 101: class diagram of validation util

The validation util file contains methods to carryout validation in our forms in login, register and contact pages. There are plenty of other methods but we will be discussing only the used methods in our system.

```
// 1. Validate if a field is null or empty
public static boolean isNullOrEmpty(String value) {
    return value == null || value.trim().isEmpty();
}
```

Figure 102: code for null or empty

This method is used to check the null or empty fields in our system.

```
// 2. Validate if a string contains only letters
public static boolean isAlphabetic(String value) {
    return value != null && value.matches("^[a-zA-Z]+$");
}
```

Figure 103: code for isAlphabetic

This method is used to check the alphabetic characters in our system.

```
// 3. Validate if a string starts with a letter and is composed of letters and numbers
public static boolean isAlphanumericStartingWithLetter(String value) {
    return value != null && value.matches("^[a-zA-Z][a-zA-Z0-9]*$");
}
```

Figure 104: code for isAlphanymericString

This method is used to check if the string starts with the letter and is composed with letters and numbers which is mainly used in password validation.

```
// 5. Validate if a string is a valid email address
public static boolean isValidEmail(String email) {
    String emailRegex = "^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$";
    return email != null && Pattern.matches(emailRegex, email);
}
```

Figure 105: code for isValidEmail

This method is used to validate if the email address is in the correct format.

```
// 6. Validate if a number is of 10 digits and starts with 98
public static boolean isValidPhoneNumber(String number) {
    return number != null && number.matches("^98\\d{8}$");
}
```

Figure 106: code for isValidPhoneNumber

This method checks if the phone number is in the correct format and if the phone number starts with 98.

```
// 8. Validate if a Part's file extension matches with image extensions (jpg, jpeg, png, gif)
public static boolean isValidImageExtension(Part imagePart) {
    if (imagePart == null || isNullOrEmpty(imagePart.getSubmittedFileName())) {
        return false;
    }
    String fileName = imagePart.getSubmittedFileName().toLowerCase();
    return fileName.endsWith(".jpg") || fileName.endsWith(".jpeg") || fileName.endsWith(".png") || fileName.endsWith(".gif");
}
```

Figure 107: code for valid image

This method is used to check the valid image type which the user is submitting in the form.

```
// 9. Validate if password and retype password match
public static boolean doPasswordsMatch(String password, String retypePassword) {
    return password != null && password.equals(retypePassword);
}
```

Figure 108: code for password matching

This method is used to check of the password and confirm password text field contain the same password to avoid password conflict in the database.

5. Test Cases

5.1. Planning Test Cases

Table 1: Test cases planning

Test Number	Test Case	Test Description
1	User Register	This feature should allow user to register into the system by entering correct details from the web page and add that user to the database.
2	User Login	This feature should allow user to login into the system to access the pages and as a proof of logging in, the user's credentials should be displayed in the profile page.
3	Profile Update	This feature should allow user to update their information from the profile page such as username, email, password, phone and profile image.
4	Add Product	This is the feature for the admins to add a new product to the database and show that product to the products page in the main web page.
5	Update and Delete Product	This feature is also for the admins to update the information of the already existing products and delete the already added product and show the changes in the database as well as in the product page of the web page as well.
6	Product Search	This feature should allow the user to search any products and if the product is found then display its product card else show appropriate message to let user

		know that the requested product doesn't exist.
7	Cookie Testing	The cookie allocates the role to the user who is trying to access the system which creates two different sides of the system i.e. admin and customer then the system allows the certain features to be used as a customer or admin.
8	Filter Testing	The filter takes the role from the cookie and then creates a barrier between the customer and admin which means there are certain features that only the admin can operate such as admin dashboard.
9	Profile Data Fetch	This feature is supposed to fetch the data of the user credentials and display it in the profile page.
10	Profile Update	This feature is supposed to update the user credentials and also reflect those changes in the database.

5.2. Test 1: User register

Table 2: Test 1: User register

Test Name	User Register
Test Description	This feature should allow user to register into the system by entering correct details from the web page and add that user to the database.
Actions Performed	Empty fields are validated first and wrong inputs are validated and then after entering correct details the test is completed.
Expected Result	After successful registration, the login page should be seen, and registered user should be seen in the database.
Actual Result	After successful registration, the login page is seen and the registered user is seen in the database.
Remarks	The test was successful.

- **Empty Fields Validation**

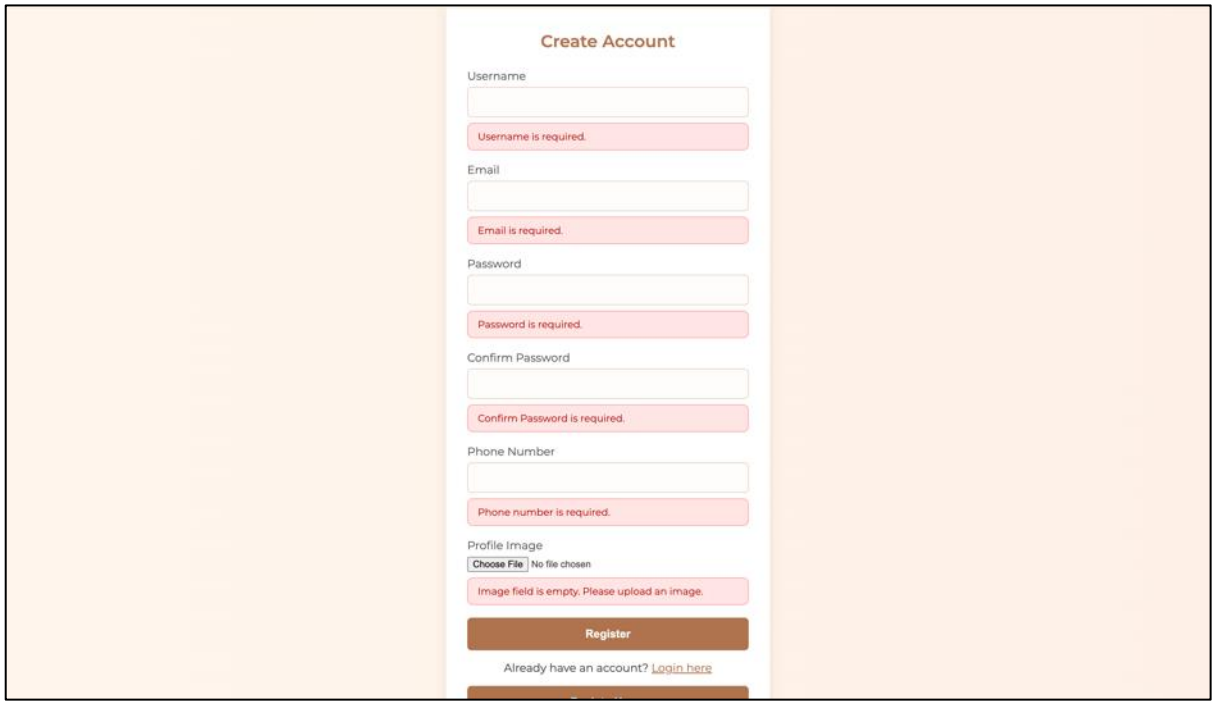


Figure 109: empty fields validation in register page

Here, when user directly clicks the register button without entering all the details, the above scrrenshot shows the result.

- **Incorrect details validation**

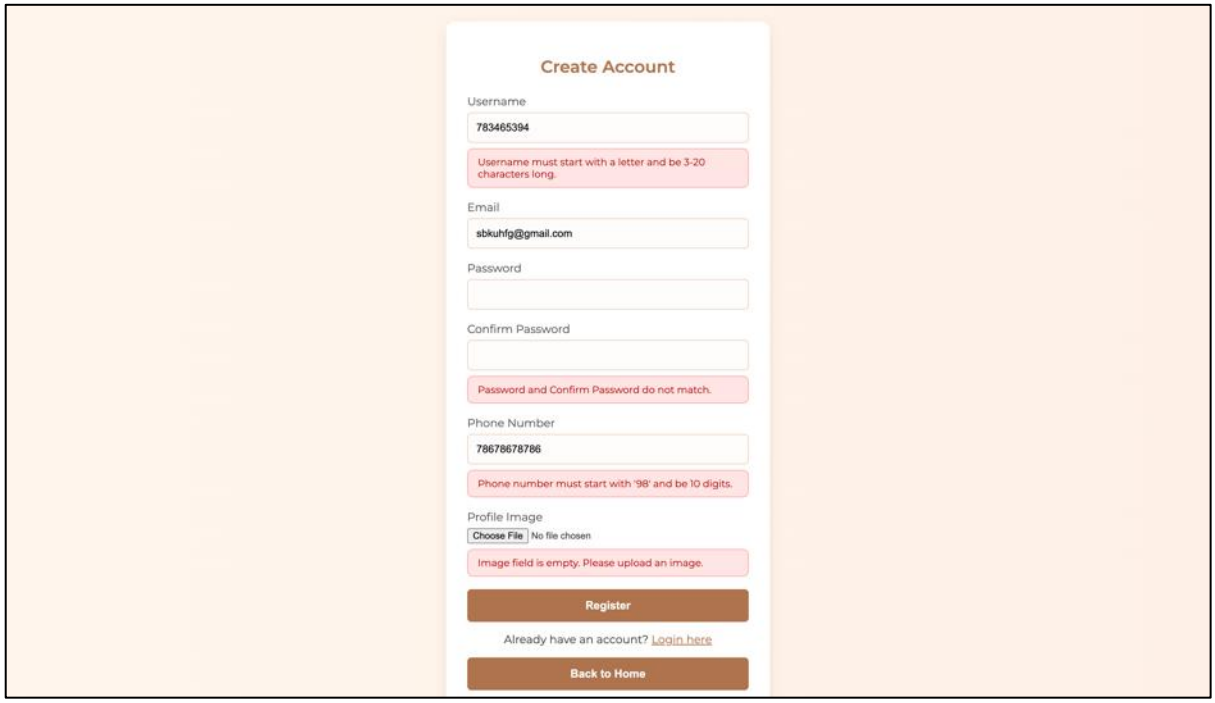


Figure 110: incorrect details validation in register page

Here we can see that the text fields are validated with their respective input types and if the user enters the wrong input and tries to register, the respective error messages are seen, and password text fields are cleared to maintain the security of the web page.

- **Correct Details**

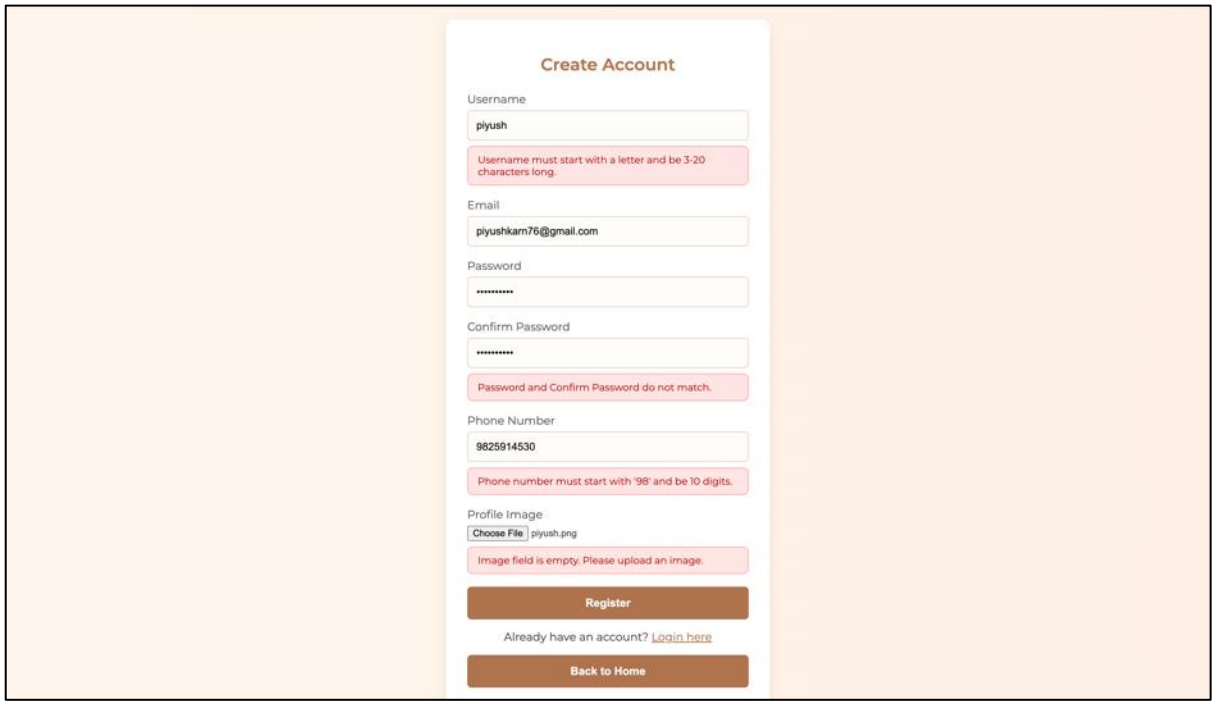


Figure 111: entering correct details in the register page

After entering the correct details the page was navigated to login page and the database was updated with the newly registered user and password encryption and image utility is also performed.

- **Registered user**

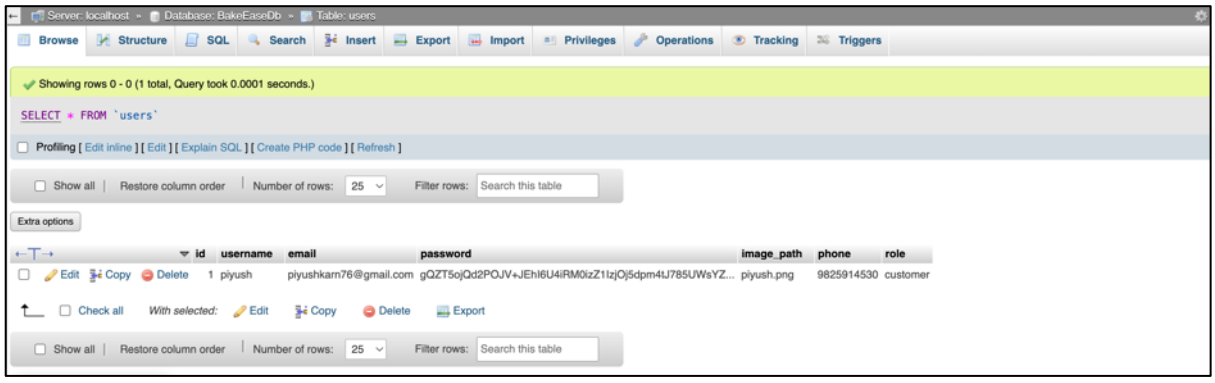


Figure 112: registered user in the database

Here the above registered user is registered in the database and can be accessed for login.

5.3. Test 2: User Login

Table 3: Test 2: user login

Test Name	User login
Test Description	This feature should allow user to login into the system to access the pages and as a proof of logging in, the user's credentials should be displayed in the profile page.
Actions Performed	Empty fields are validated first and then after entering correct details the test is completed when the web page is navigated to home page.
Expected Result	After successful registration, the home page should be seen, and logged in user should be seen in the profile page.
Actual Result	After successful registration, the home page is seen and the logged in user is seen in the profile page.
Remarks	The test was successful.

- **Wrong Details Validation**

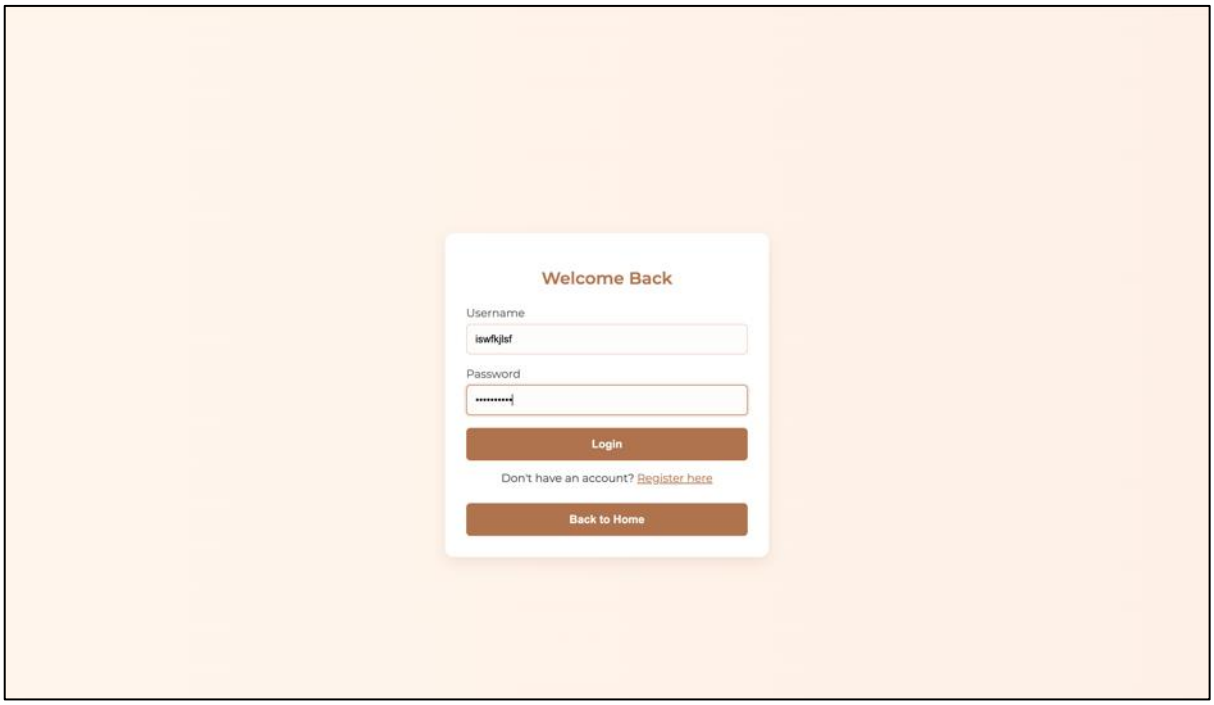


Figure 113: before entering wrong details in the login page

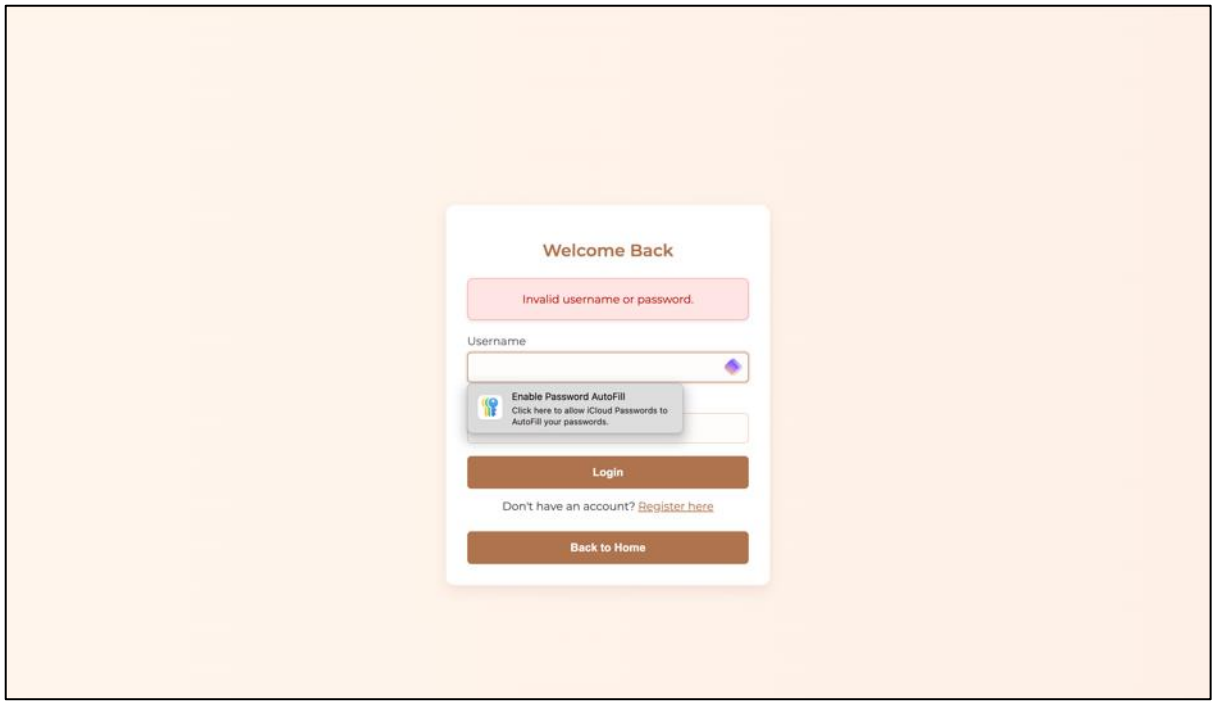


Figure 114: after entering wrong details in the login page

Here the error message was seen because the user with the entered page in the web page is not found in the database and hence the login feature cannot continue until the user is logged in with the correct credentials.

- **Entering Correct Details**

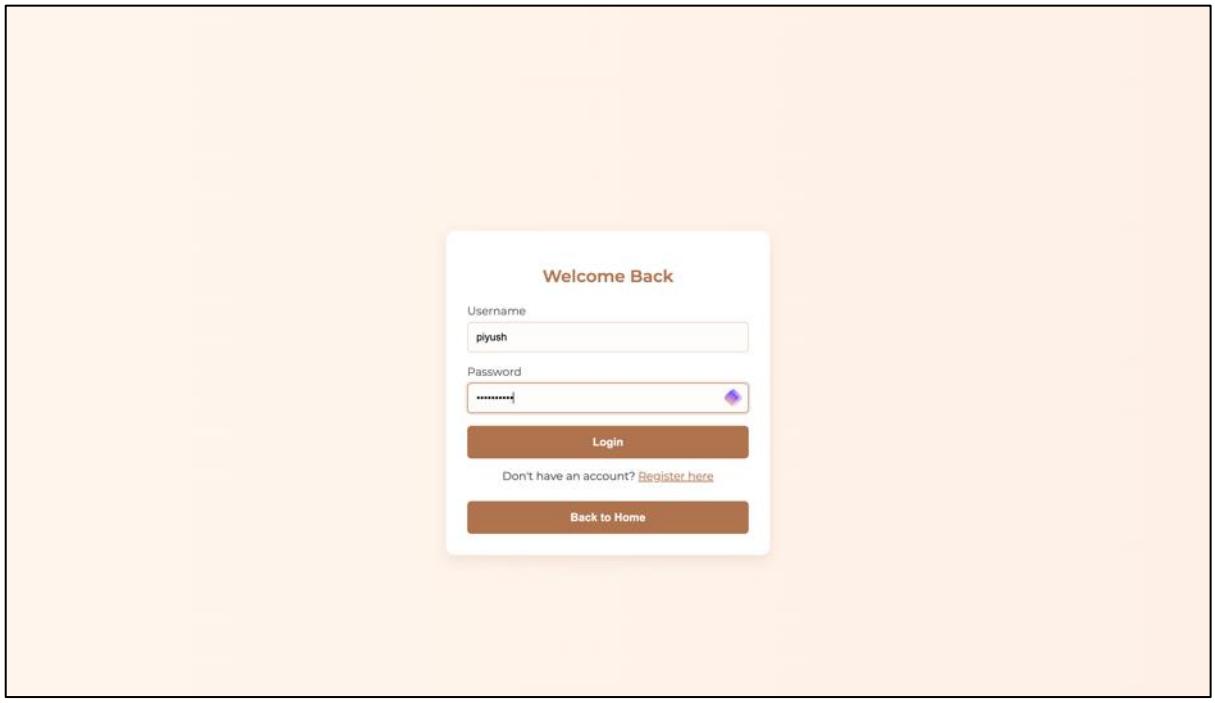


Figure 115: before entering correct details in the web page

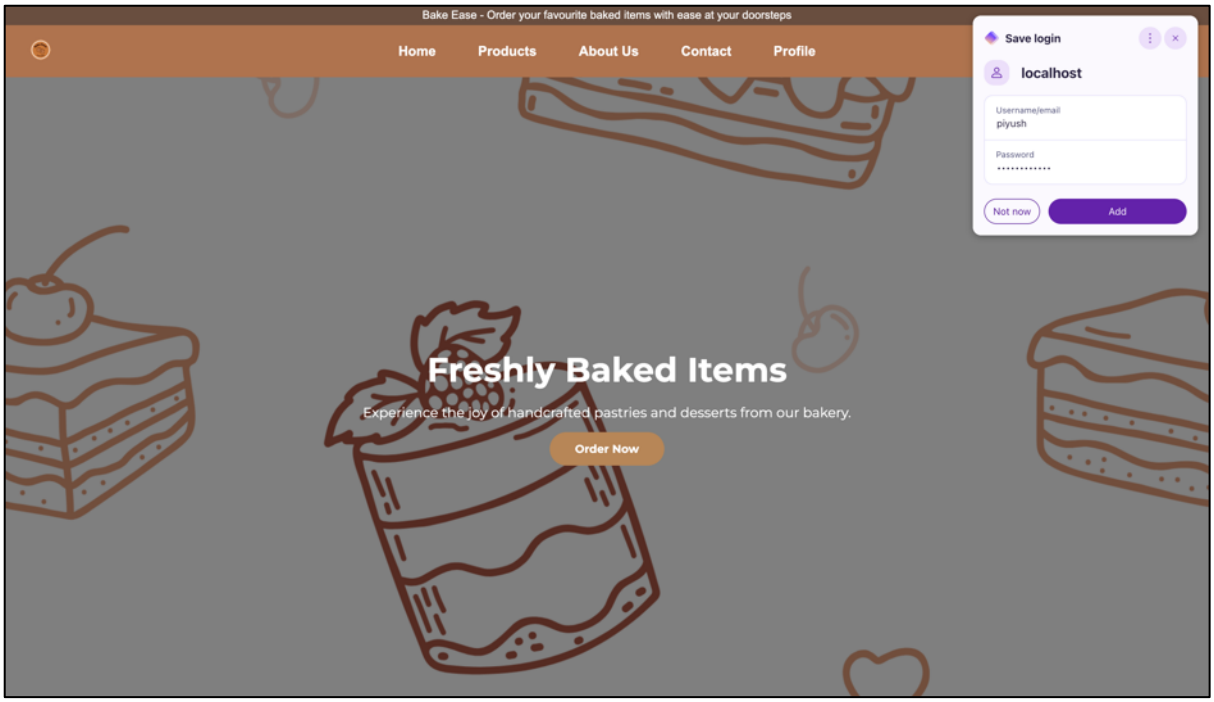


Figure 116: after entering the correct details in the web page

After entering the matching username and password from the database, the home page is seen, and the profile image of the user is seen in the project resources folder and profile page can be seen now which contains the information of the user.

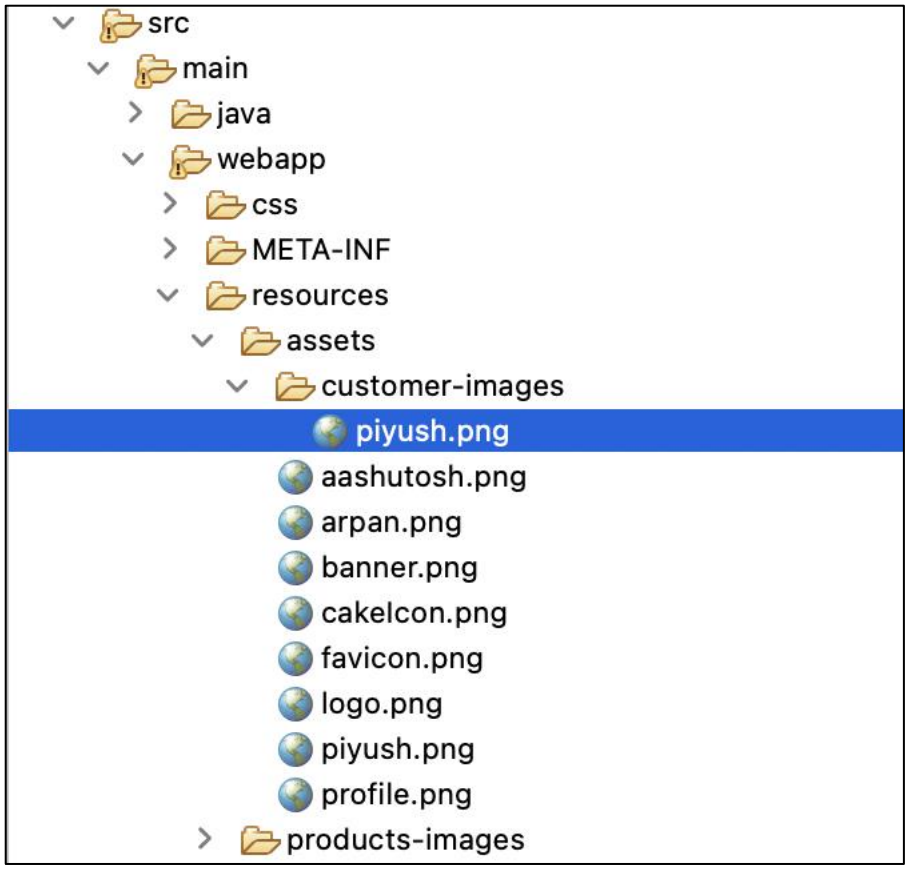


Figure 117: image of the logged in user in the system

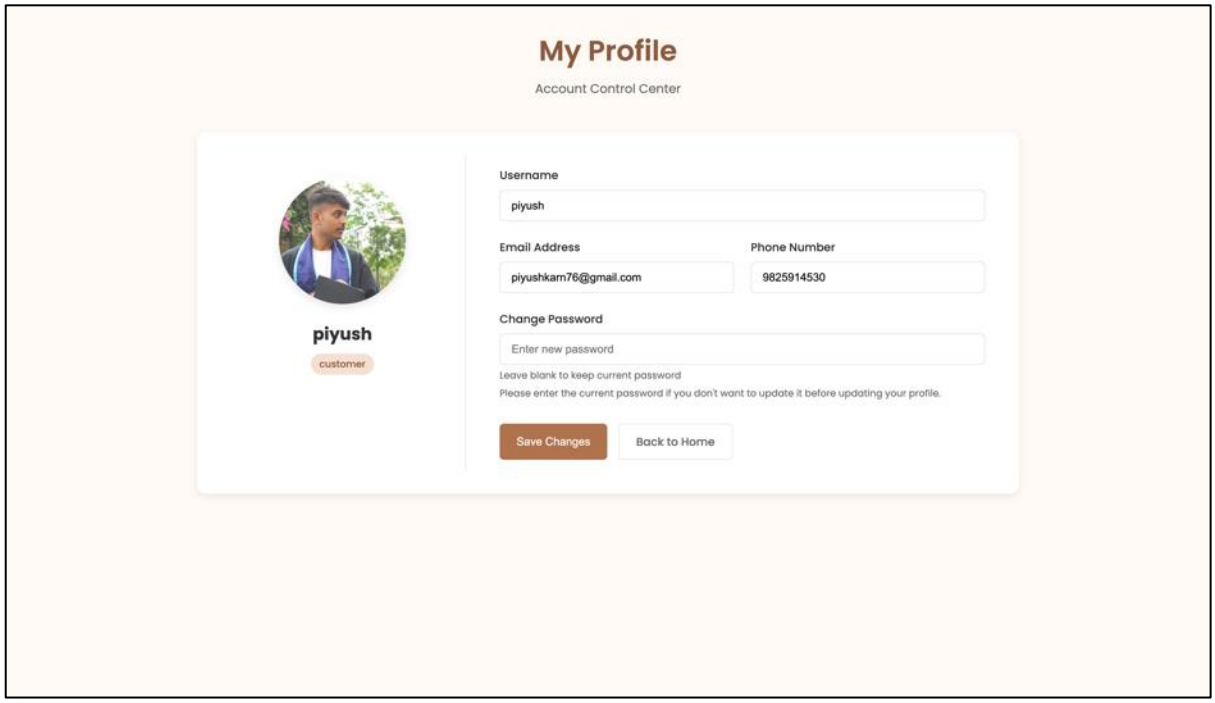


Figure 118: user profile of the currently logged in user

Here we can see the profile image of the logged in user is seen in the resources folder of the project and the profile page contains the details of the user fetched from the database.

5.4. Test 3: Profile Update

Table 4: Test 3: Profile update

Test Name	Profile Update
Test Description	This feature should allow user to update their information from the profile page such as username, email, password, phone and profile image.
Actions Performed	The text fields are changed with the details to be changed.
Expected Result	The profile should be updated, and the changes should also be reflected in the database.
Actual Result	The profile is updated and the changes should can be seen in the database.
Remarks	The test was successful.

- **Before Updating**

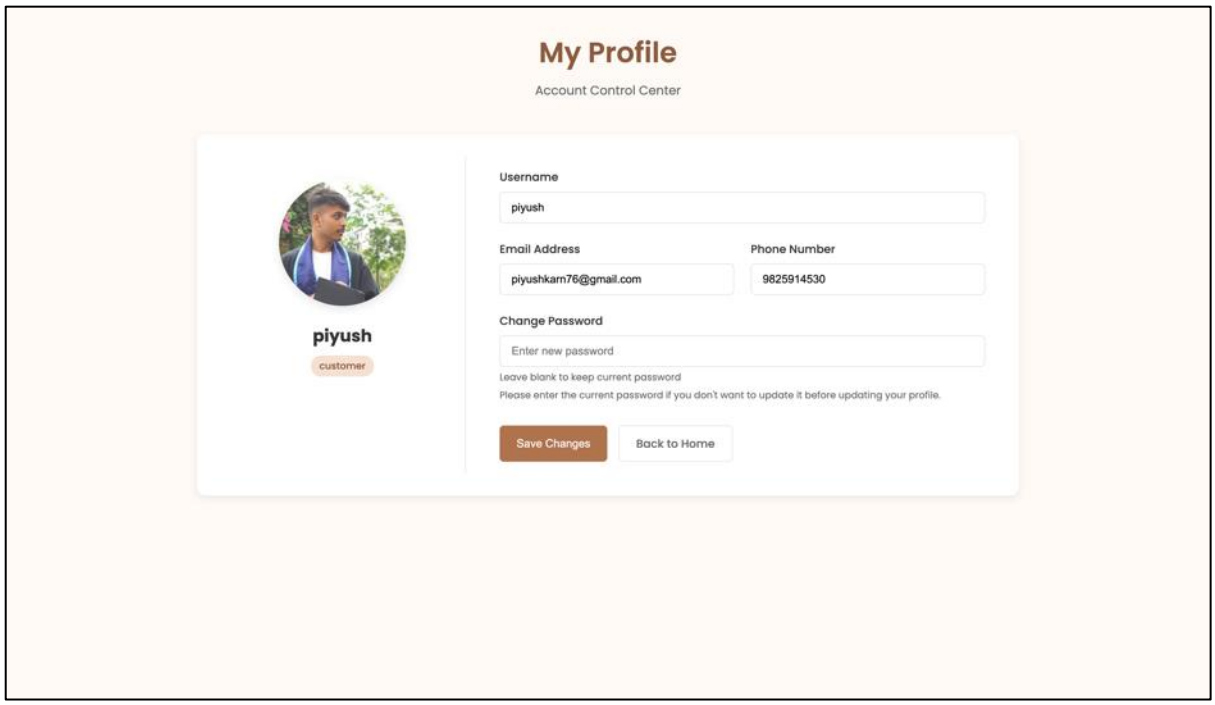


Figure 119: before updating user profile page

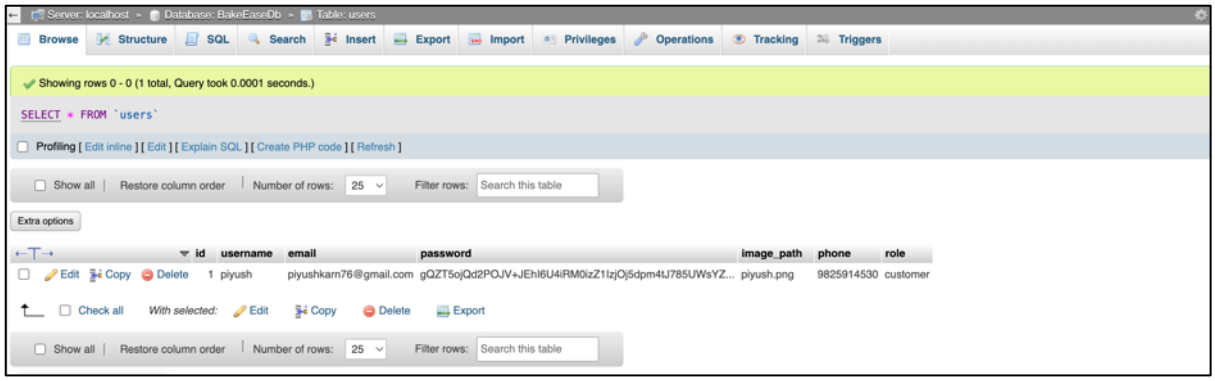


Figure 120: database before updating user

- **After Updating User Profile**

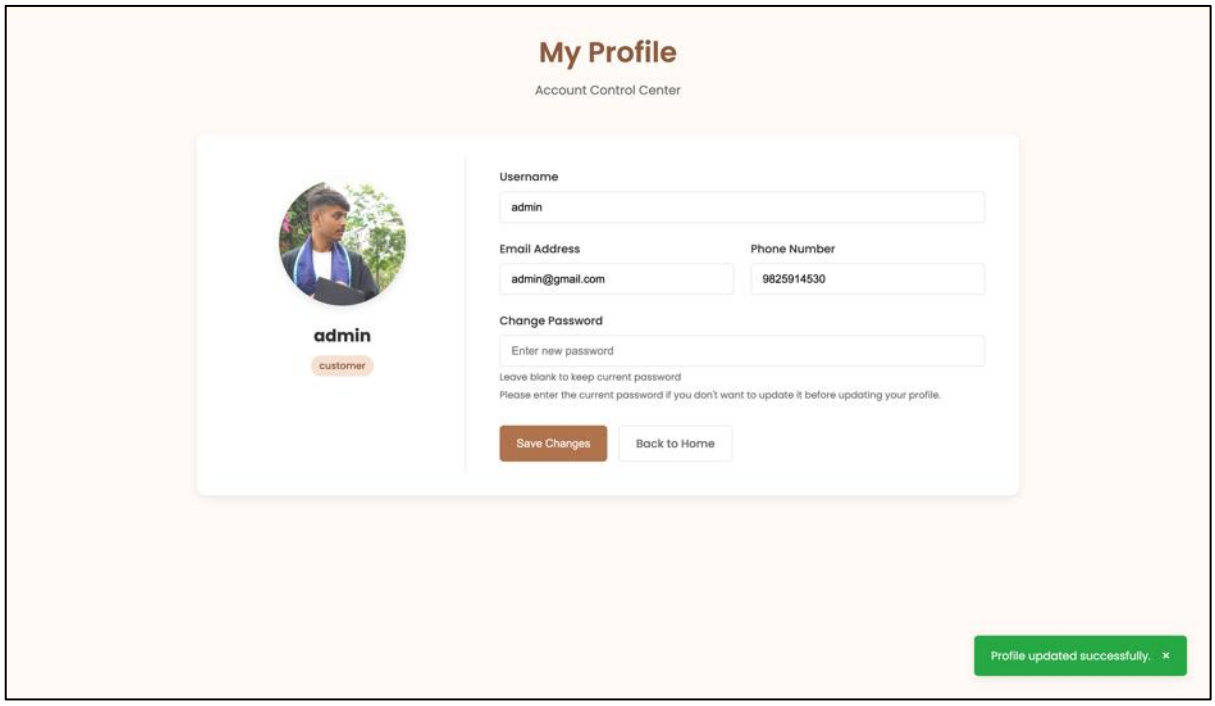


Figure 121: After updating user profile

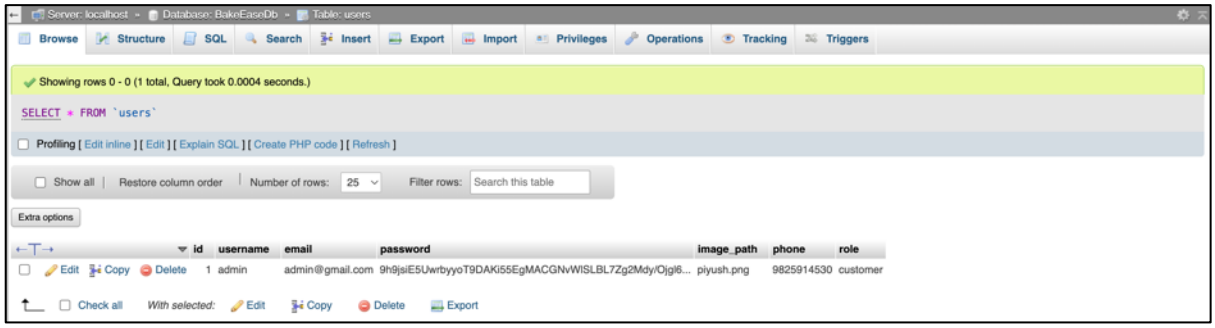


Figure 122: Database after updating table

5.5. Test 4: Add Product

Table 5: Test 4: adding product

Test Name	Add Product
Test Description	This is the feature for the admins to add a new product to the database and show that product to the products page in the main web page.
Actions Performed	The admin should enter new product's information from the dashboard and add product.
Expected Result	The new product should be added and seen in the admin dashboard and also in the product table.
Actual Result	The new product is added and seen in the admin dashboard and also in the product table.
Remarks	The test was successful.

To perform this operation, first we need to change the role of the user from customer to admin. The role of the user can be changed from the database itself only, there is no other way to do it so as to prevent admin violation.

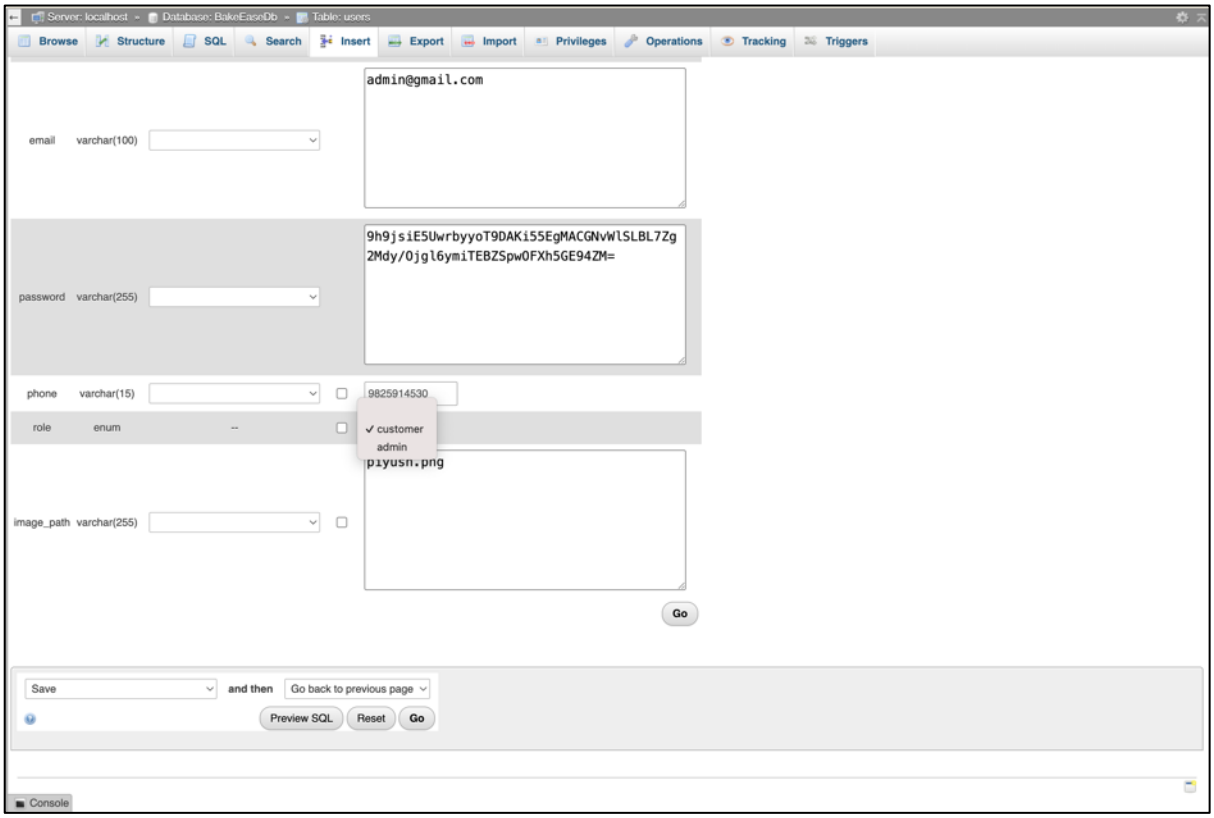


Figure 123: updating the user role from the database

After the role is updated, the changes is also seen in the profile page.

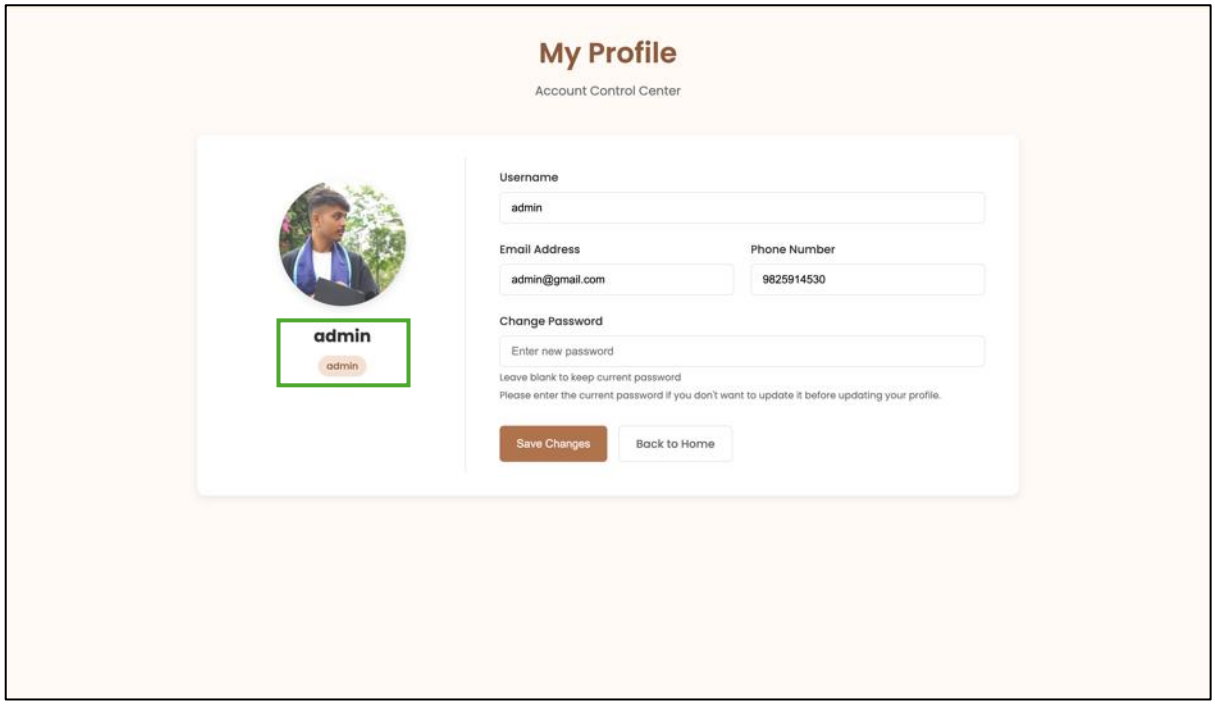


Figure 124: changed role information in profile page

- **Before adding product**

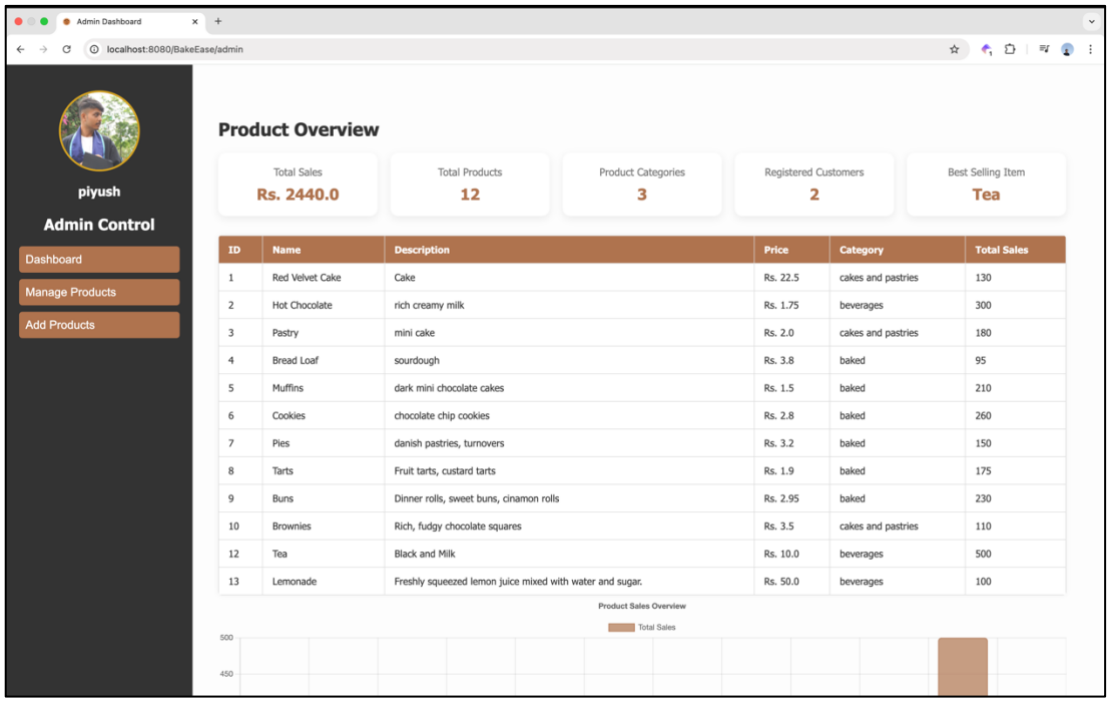
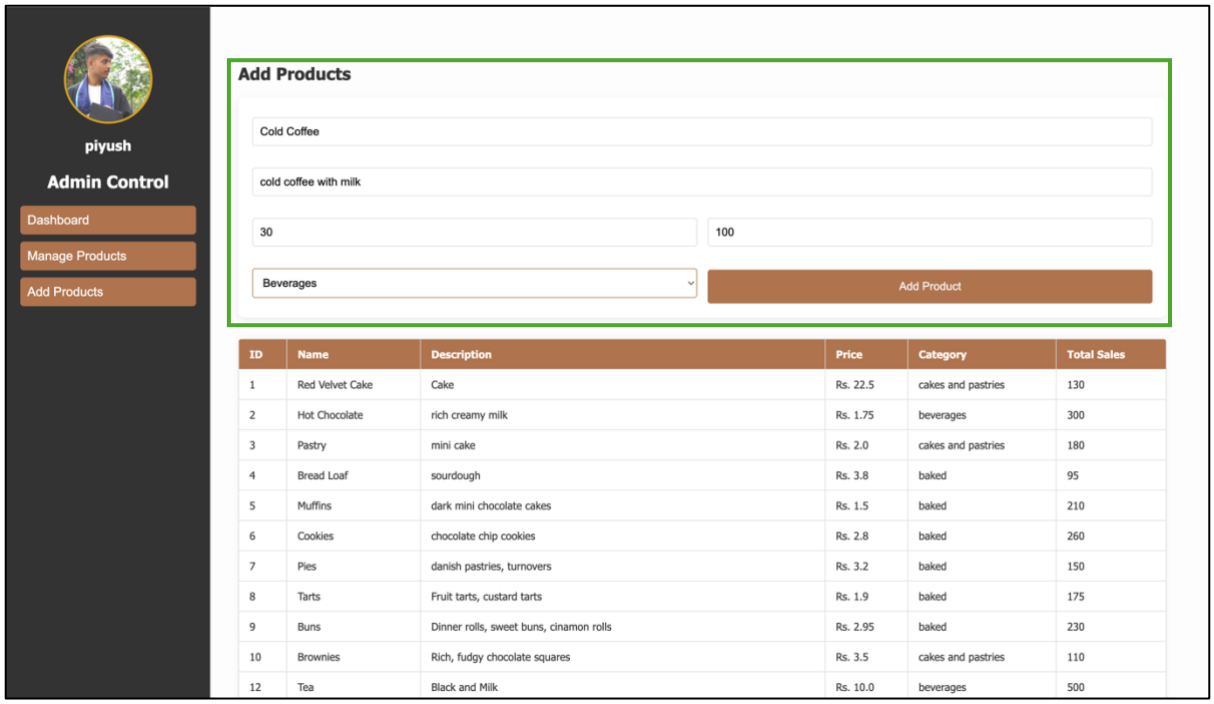


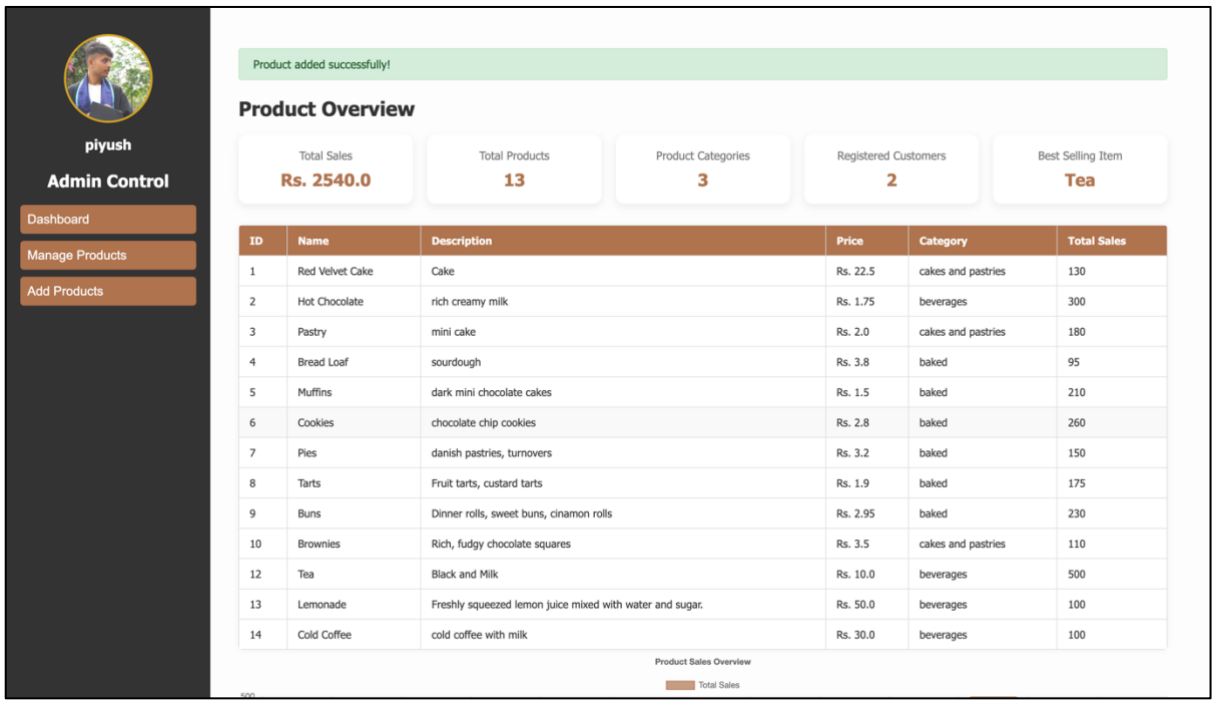
Figure 125: Before adding product

- **Adding Product**



ID	Name	Description	Price	Category	Total Sales
1	Red Velvet Cake	Cake	Rs. 22.5	cakes and pastries	130
2	Hot Chocolate	rich creamy milk	Rs. 1.75	beverages	300
3	Pastry	mini cake	Rs. 2.0	cakes and pastries	180
4	Bread Loaf	sourdough	Rs. 3.8	baked	95
5	Muffins	dark mini chocolate cakes	Rs. 1.5	baked	210
6	Cookies	chocolate chip cookies	Rs. 2.8	baked	260
7	Pies	danish pastries, turnovers	Rs. 3.2	baked	150
8	Tarts	Fruit tarts, custard tarts	Rs. 1.9	baked	175
9	Buns	Dinner rolls, sweet buns, cinamon rolls	Rs. 2.95	baked	230
10	Brownies	Rich, fudgy chocolate squares	Rs. 3.5	cakes and pastries	110
12	Tea	Black and Milk	Rs. 10.0	beverages	500

Figure 126: adding product



ID	Name	Description	Price	Category	Total Sales
1	Red Velvet Cake	Cake	Rs. 22.5	cakes and pastries	130
2	Hot Chocolate	rich creamy milk	Rs. 1.75	beverages	300
3	Pastry	mini cake	Rs. 2.0	cakes and pastries	180
4	Bread Loaf	sourdough	Rs. 3.8	baked	95
5	Muffins	dark mini chocolate cakes	Rs. 1.5	baked	210
6	Cookies	chocolate chip cookies	Rs. 2.8	baked	260
7	Pies	danish pastries, turnovers	Rs. 3.2	baked	150
8	Tarts	Fruit tarts, custard tarts	Rs. 1.9	baked	175
9	Buns	Dinner rolls, sweet buns, cinamon rolls	Rs. 2.95	baked	230
10	Brownies	Rich, fudgy chocolate squares	Rs. 3.5	cakes and pastries	110
12	Tea	Black and Milk	Rs. 10.0	beverages	500
13	Lemonade	Freshly squeezed lemon juice mixed with water and sugar.	Rs. 50.0	beverages	100
14	Cold Coffee	cold coffee with milk	Rs. 30.0	beverages	100

Figure 127: product added

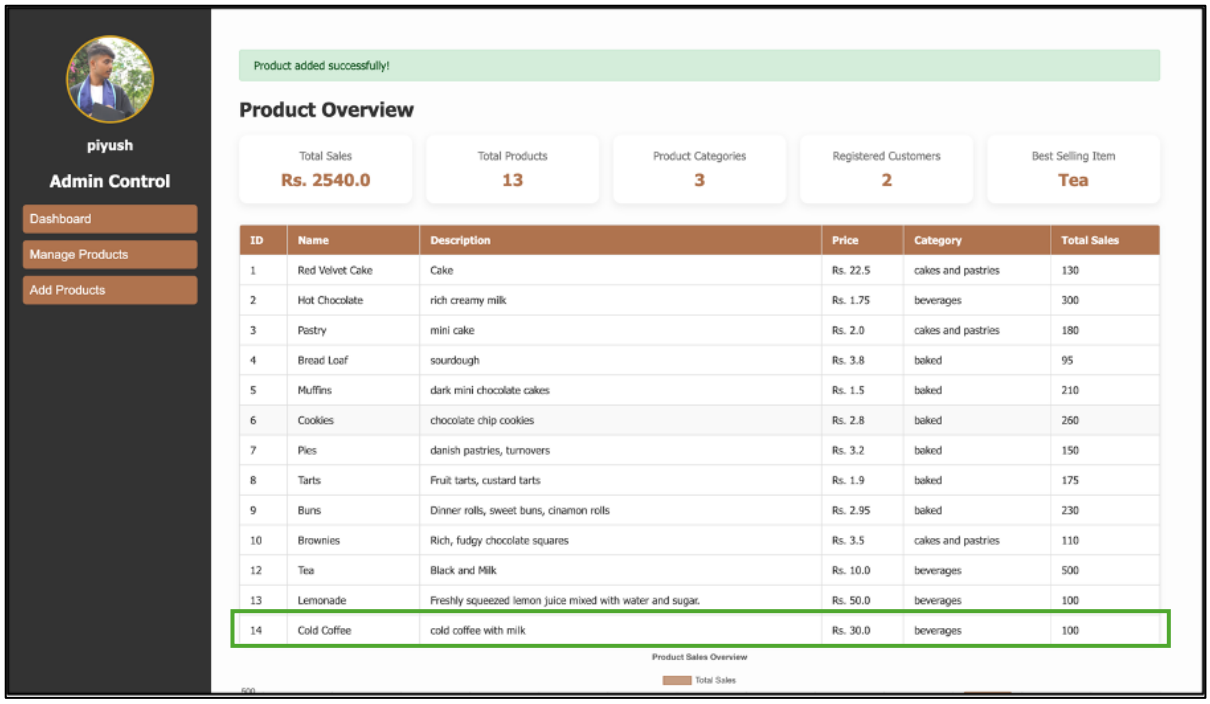


Figure 128: displaying the added product in the dashboard table

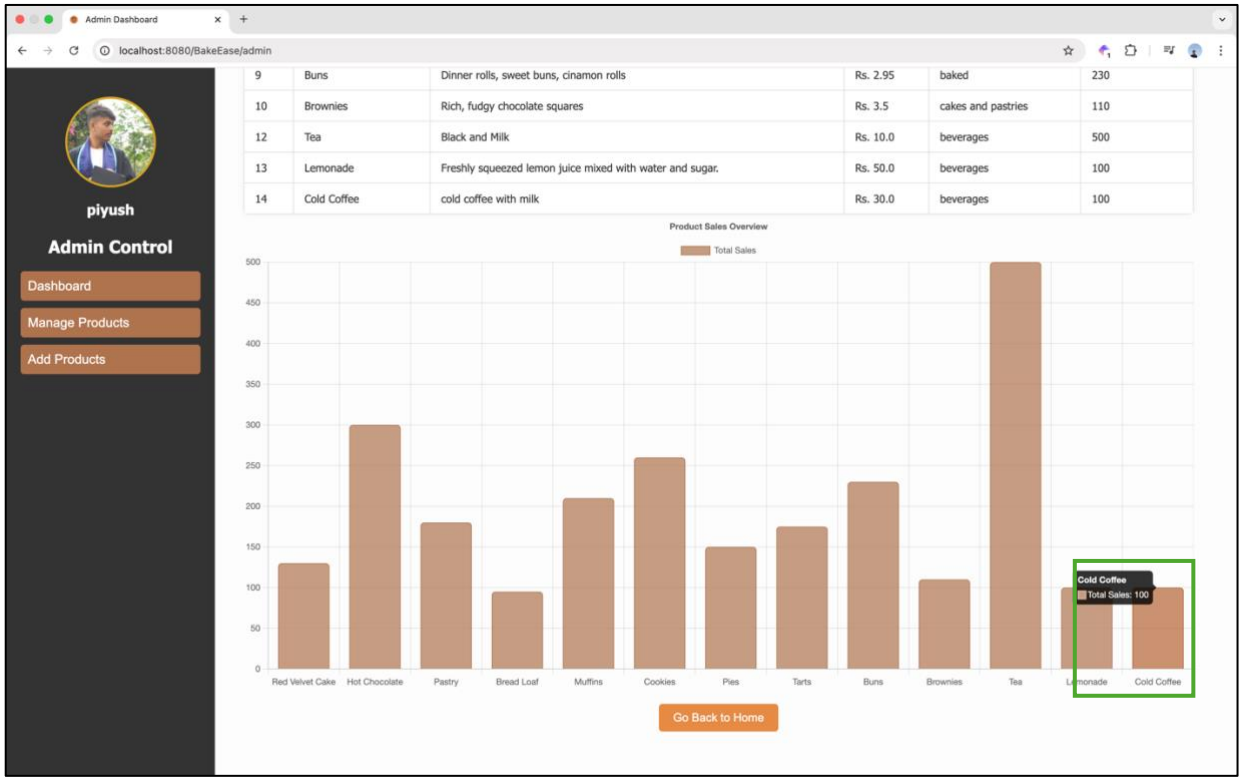


Figure 129: displaying the added product in the dashboard graph

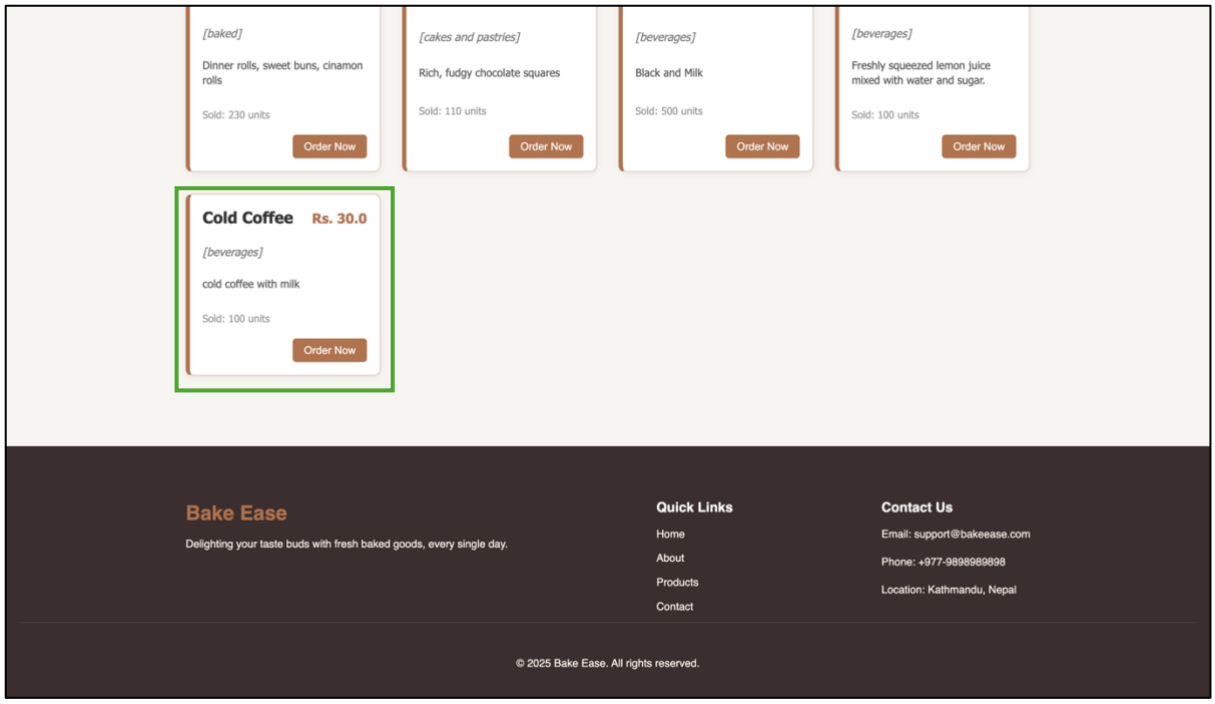


Figure 130: displaying the added product in the product page

5.6. Test 5: Updating and Deleting Product

Table 6: Test 5: Update and Delete product

Test Name	Updating and Deleting product
Test Description	This feature is also for the admins to update the information of the already existing products and also delete the already added product and show the changes in the database as well as in the product page of the web page as well.
Actions Performed	The admin should first update the product's information and after the update is completed delete the targeted product.
Expected Result	After updating the product information, the changes should be reflected in dashboard table and product page and after deleting the product the product should be deleted from the dashboard table, products page and all these changes should be reflected in the database as well.
Actual Result	After updating the product information, the changes are reflected in dashboard table and product page and after deleting the product the product is also deleted from the dashboard table, products page and all these changes are reflected in the database as well.
Remarks	The test was successful.

- **Target Product**

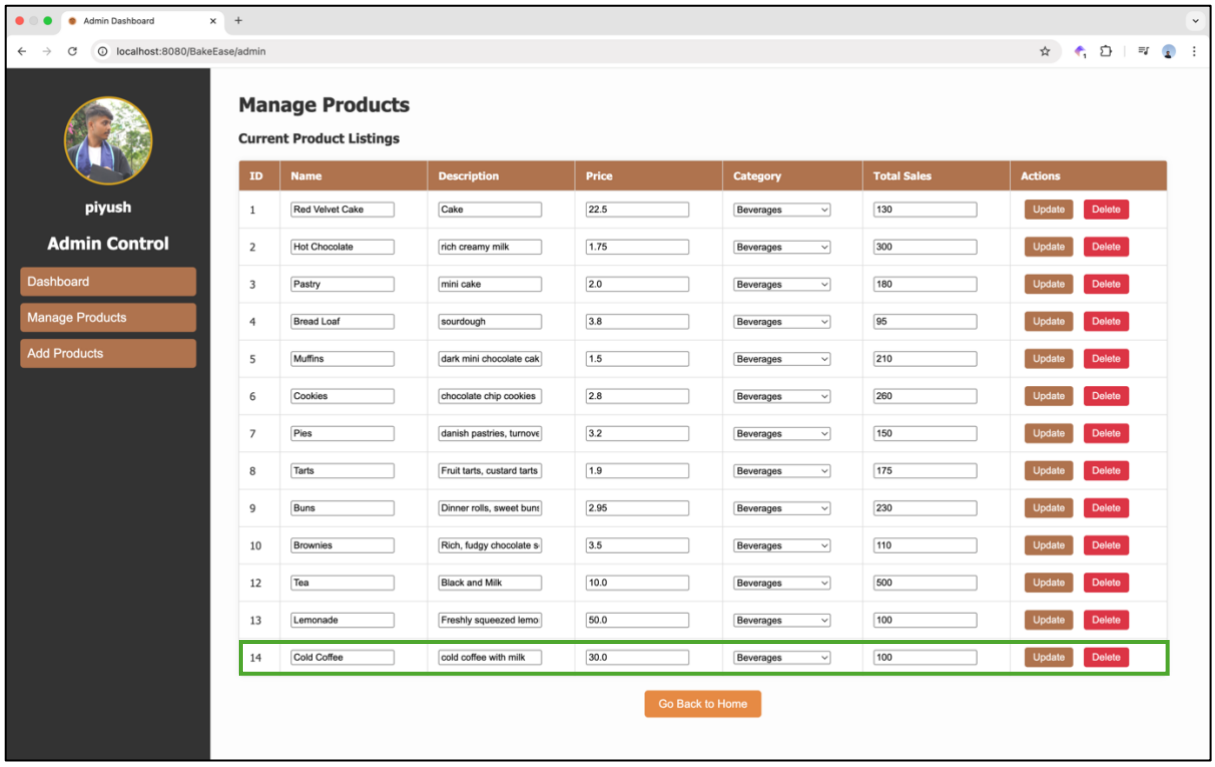


Figure 131: target product to update and delete

- **Updating Product**

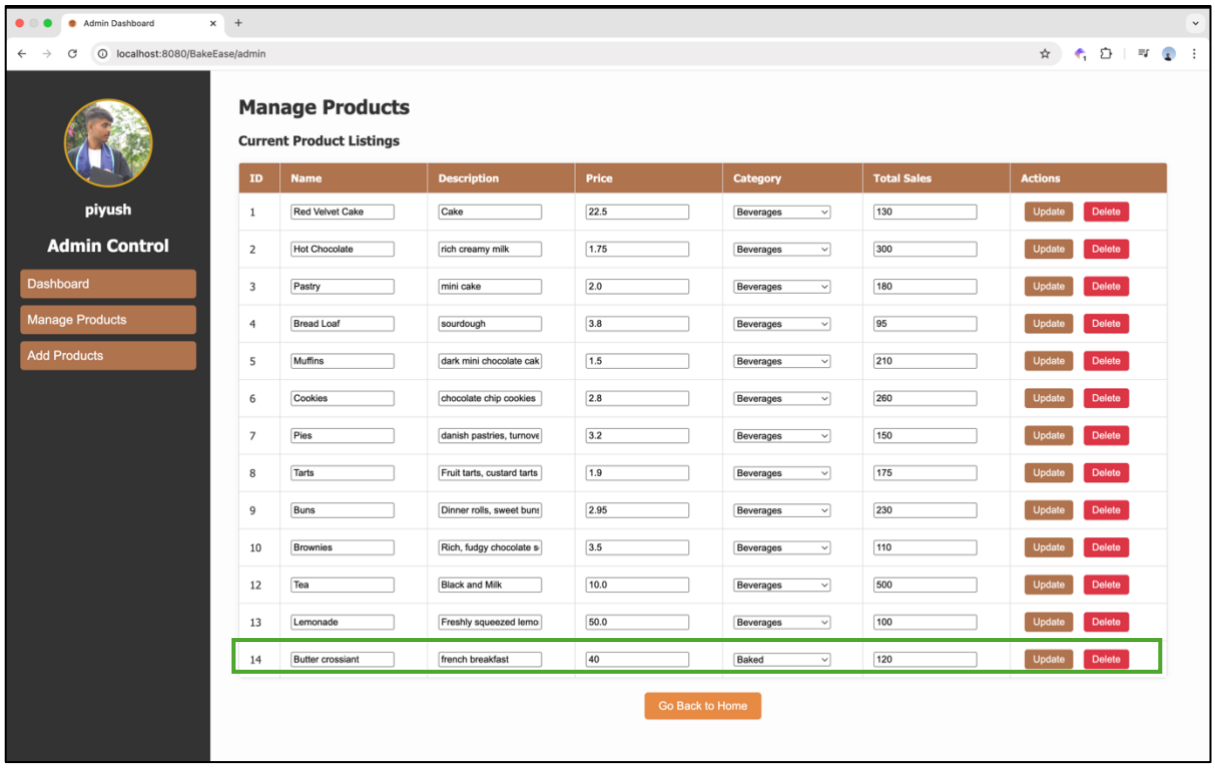


Figure 132: updating target product

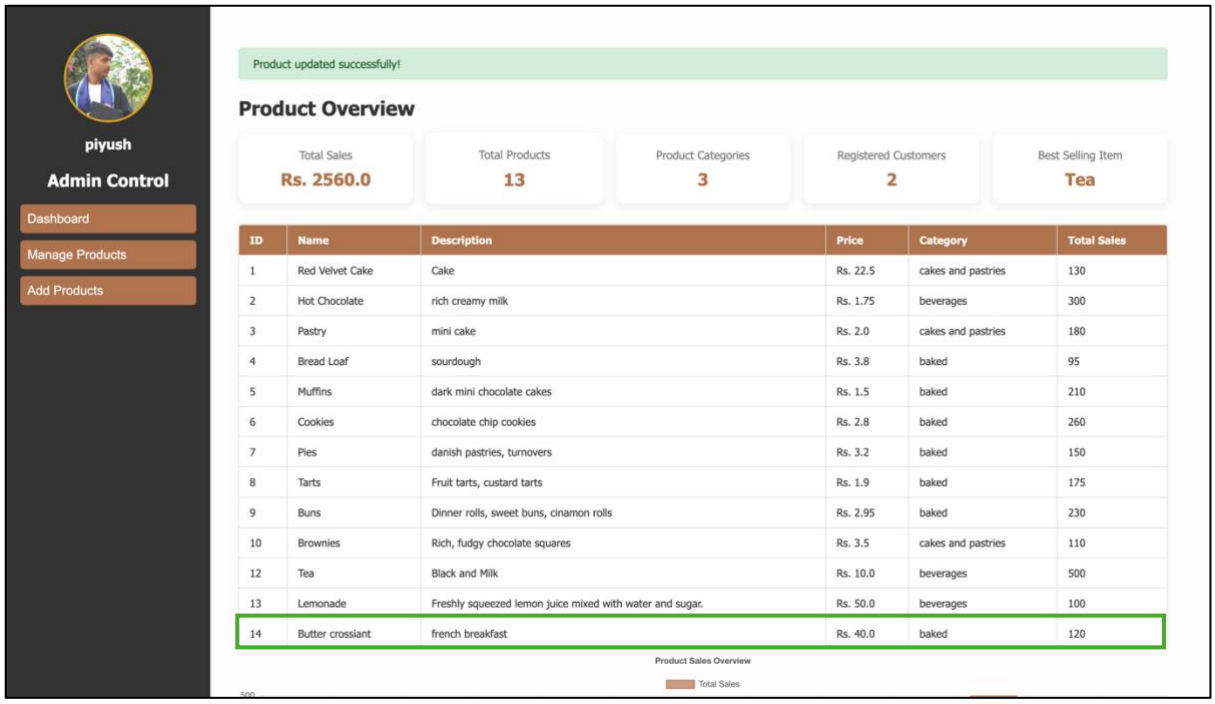


Figure 133: after updating target product

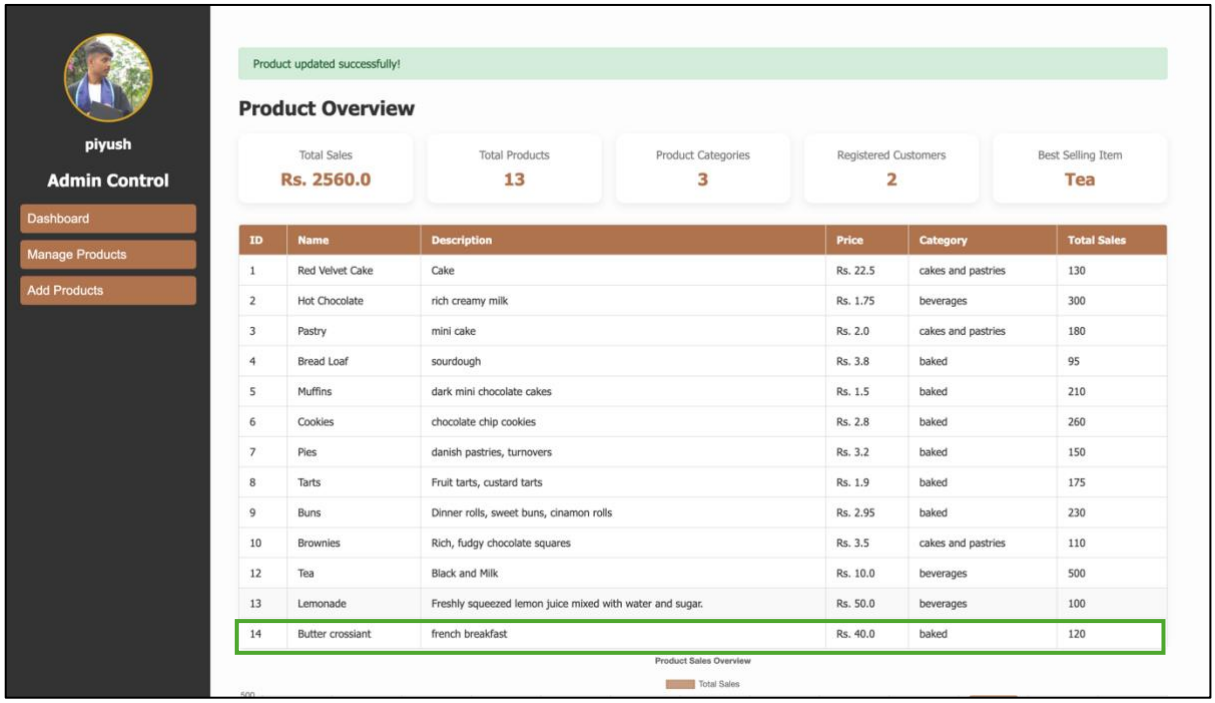


Figure 134: updated result in dashboard table

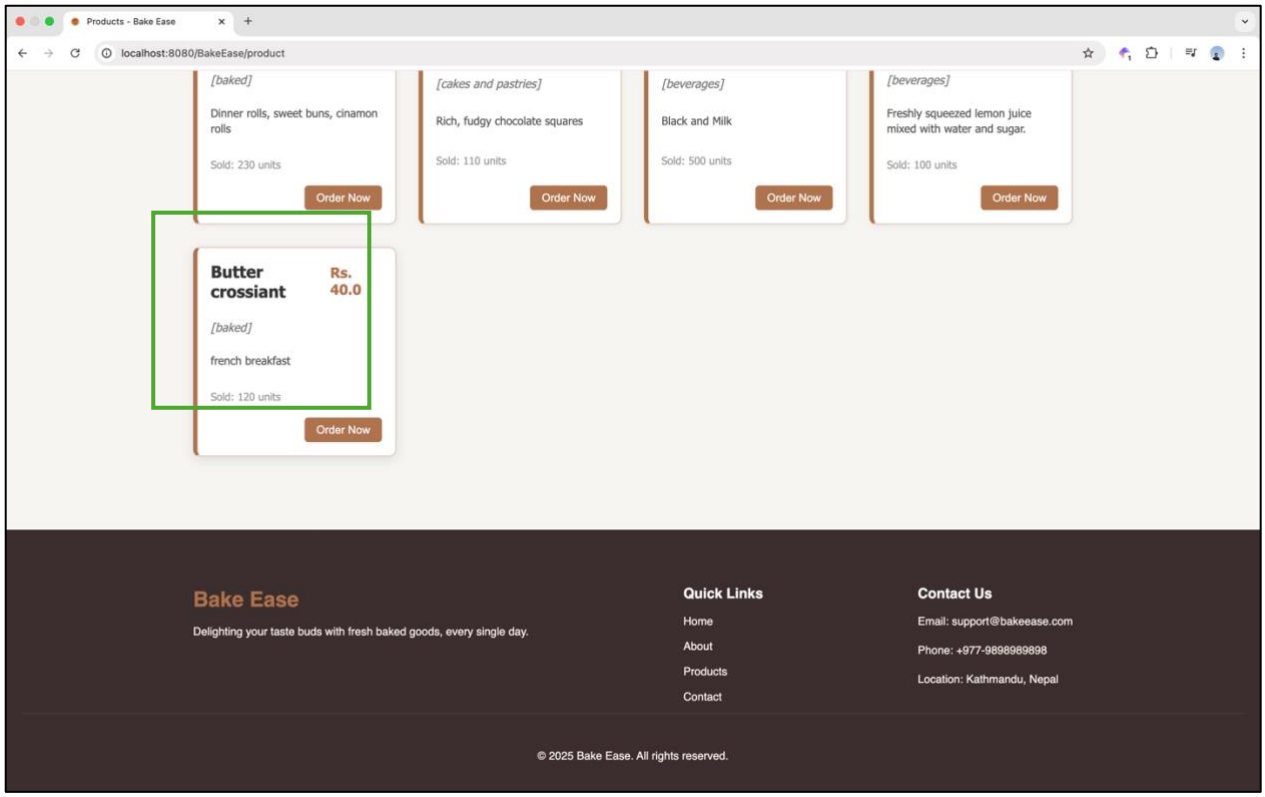


Figure 135: update result in products page

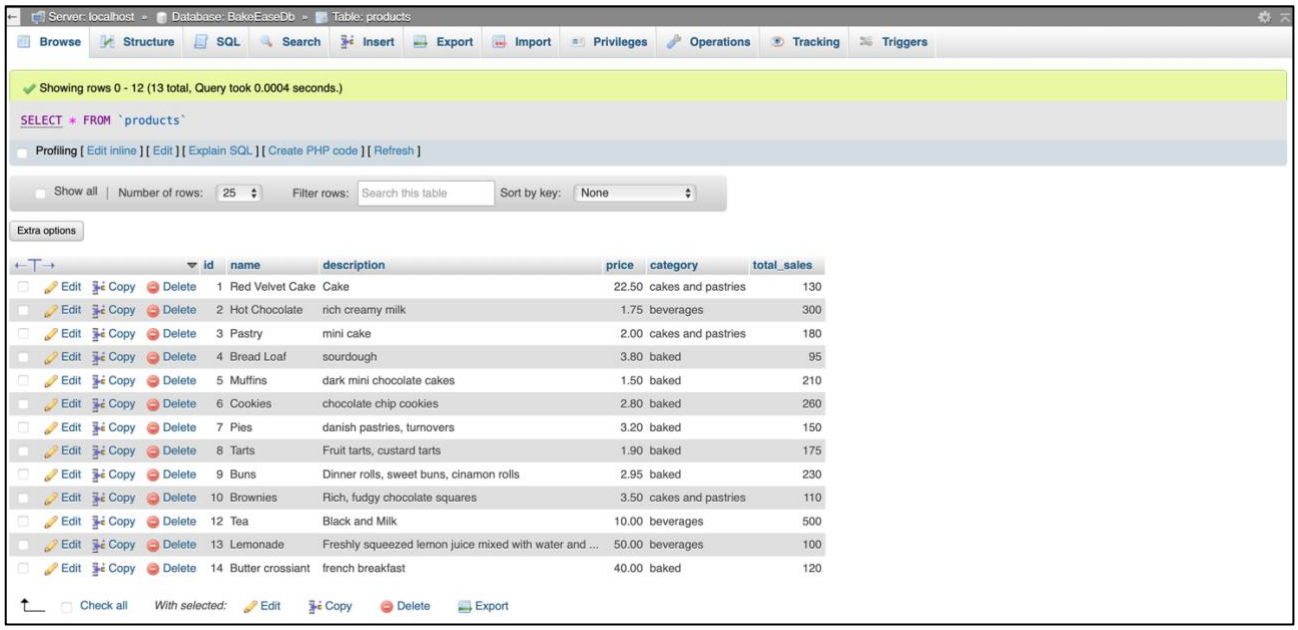


Figure 136: update result in database

- **Deleting Target Product**

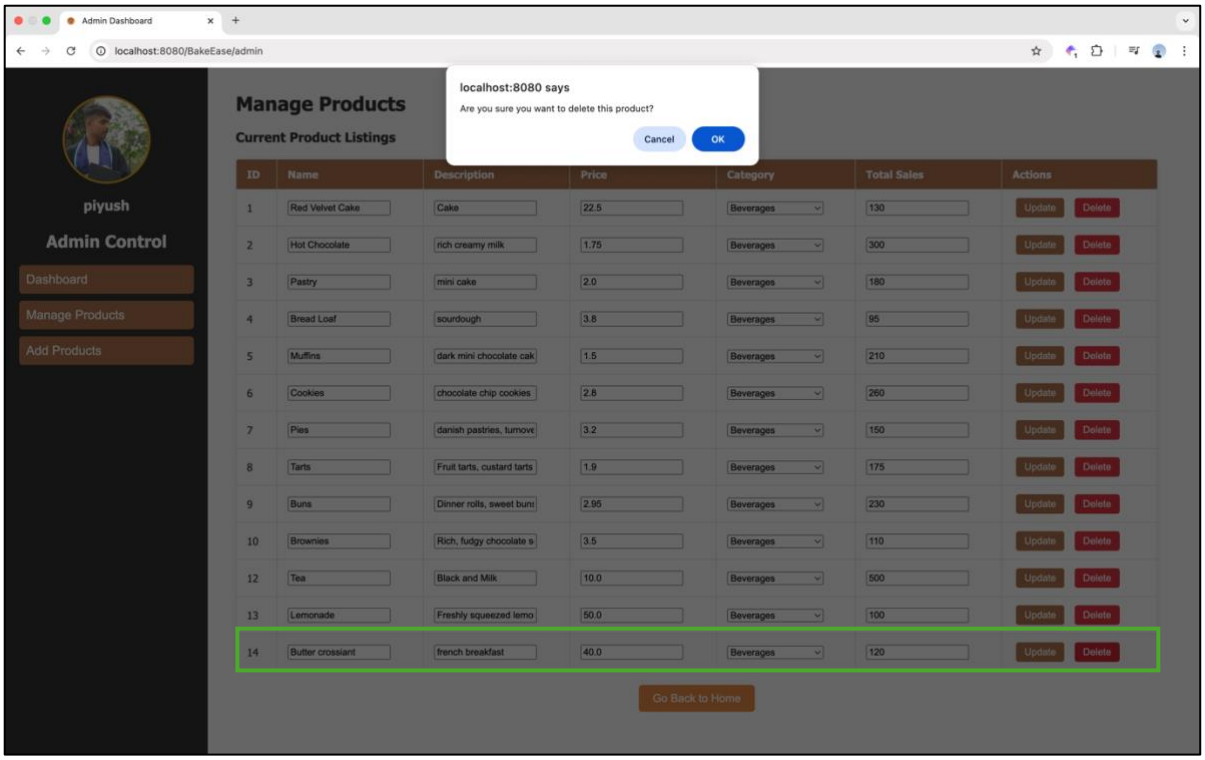


Figure 137: before deleting the target product

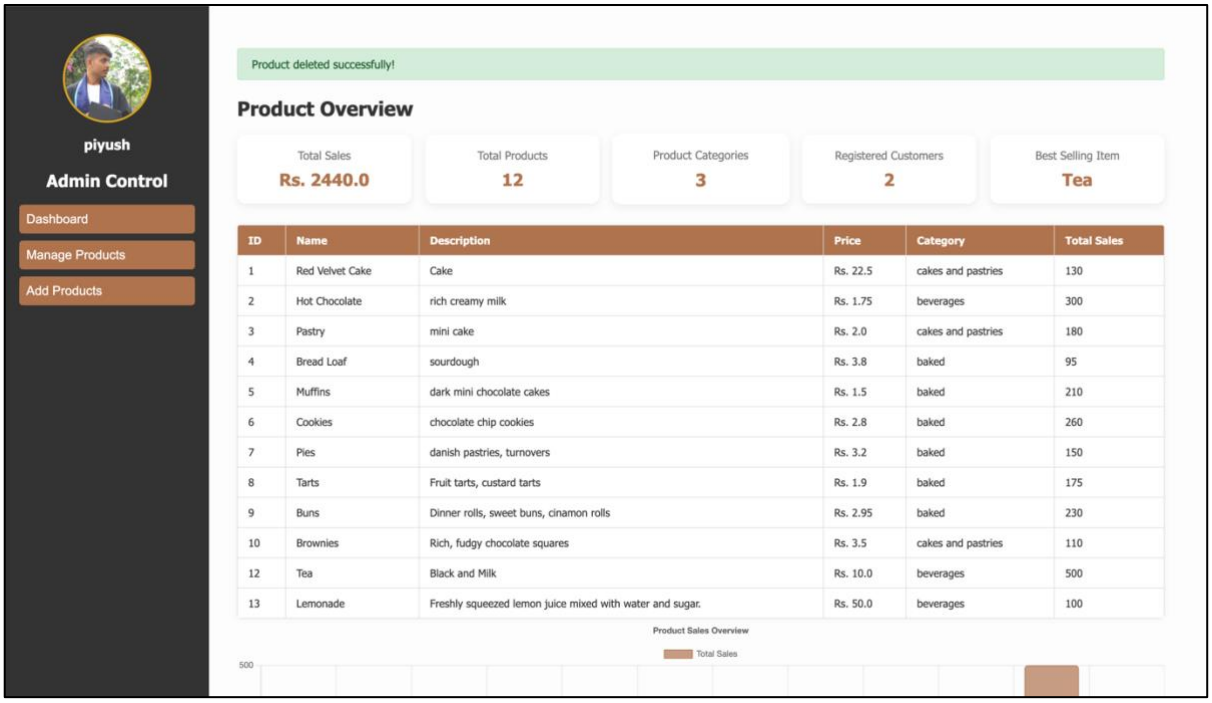
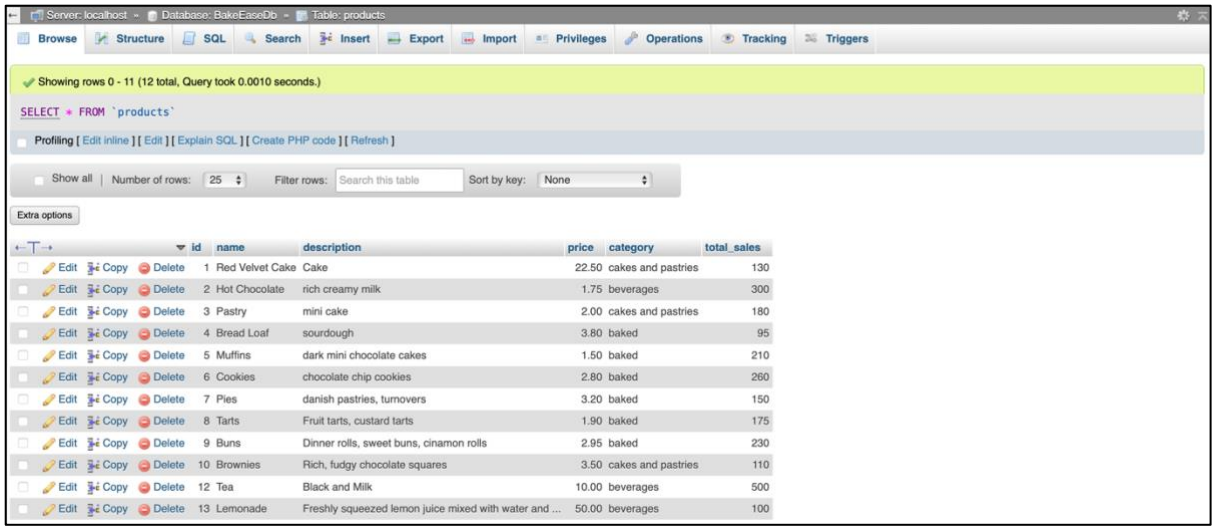


Figure 138: After deleting target product



	id	name	description	price	category	total_sales
Edit Copy Delete	1	Red Velvet Cake	Cake	22.50	cakes and pastries	130
Edit Copy Delete	2	Hot Chocolate	rich creamy milk	1.75	beverages	300
Edit Copy Delete	3	Pastry	mini cake	2.00	cakes and pastries	180
Edit Copy Delete	4	Bread Loaf	sourdough	3.80	baked	95
Edit Copy Delete	5	Muffins	dark mini chocolate cakes	1.50	baked	210
Edit Copy Delete	6	Cookies	chocolate chip cookies	2.80	baked	260
Edit Copy Delete	7	Pies	danish pastries, turnovers	3.20	baked	150
Edit Copy Delete	8	Tarts	Fruit tarts, custard tarts	1.90	baked	175
Edit Copy Delete	9	Buns	Dinner rolls, sweet buns, cinamon rolls	2.95	baked	230
Edit Copy Delete	10	Brownies	Rich, fudgy chocolate squares	3.50	cakes and pastries	110
Edit Copy Delete	12	Tea	Black and Milk	10.00	beverages	500
Edit Copy Delete	13	Lemonade	Freshly squeezed lemon juice mixed with water and ...	50.00	beverages	100

Figure 139: results in database after deleting product

5.7. Test:6 Search Products

Table 7: Test 6: Search Products

Test Name	Search Product
Test Description	This feature should allow the user to search any products and if the product is found then display its product card else show appropriate message to let user know that the requested product doesn't exist.
Actions Performed	The invalid product name is entered first and the correct product information is entered.
Expected Result	When the product is found, display its product card else show error message.
Actual Result	When correct product is found, it displayed its product card and showed error message when invalid product information was entered.
Remarks	The test was successful.

- **Product Not Found**

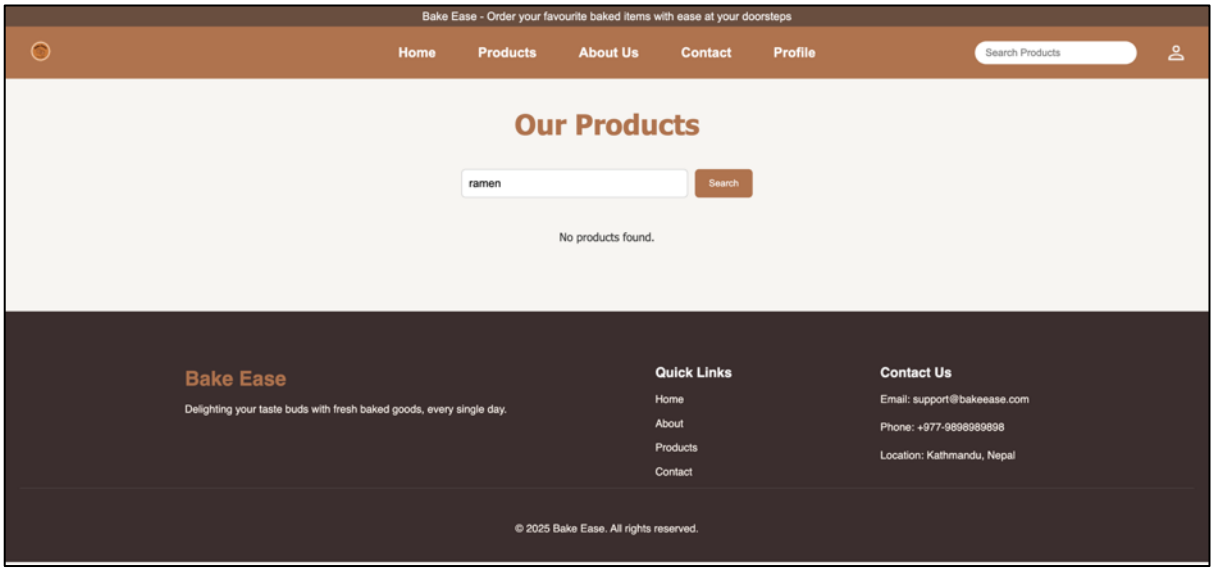


Figure 140: product not found

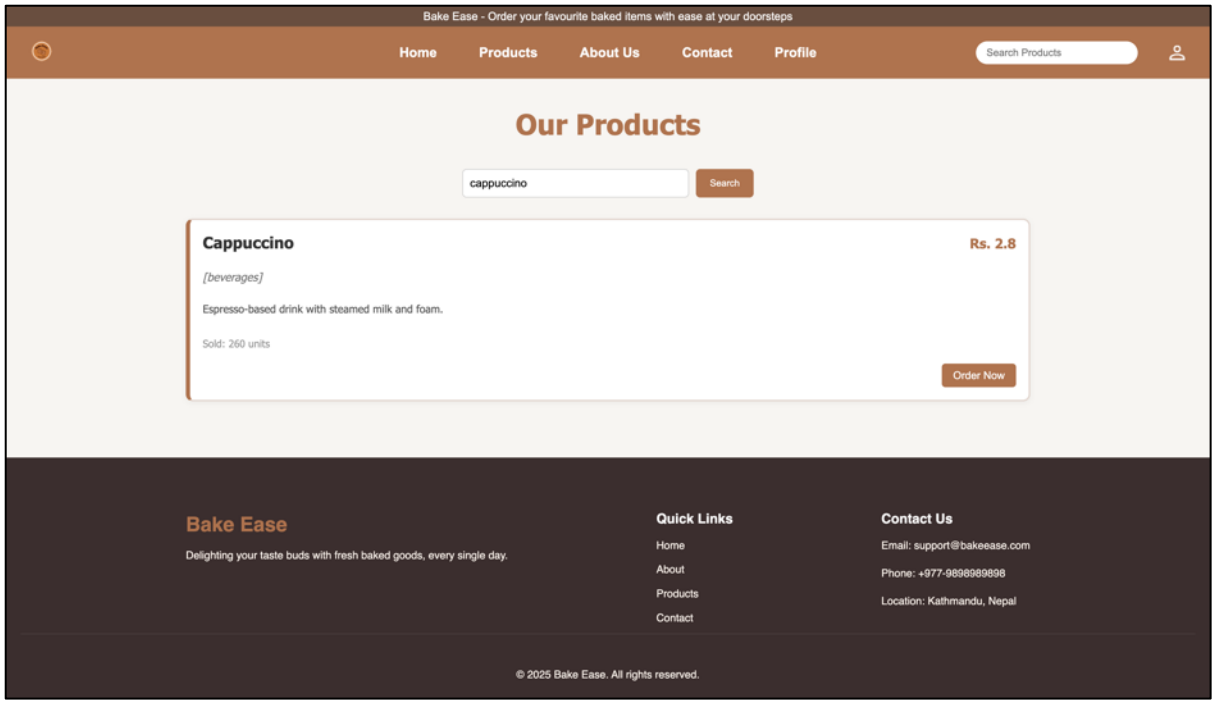


Figure 141: product found

5.8. Test 7: Cookie Testing

Table 8: Cookie testing

Test Name	Cookie Testing
Test Description	The cookie allocates the role to the user who is trying to access the system which creates two different sides of the system i.e. admin and customer then the system allows the certain features to be used as a customer or admin.
Actions Performed	The cookie should allocate the role to the user and should display the username, session Id and role of the user in the console window.
Expected Result	The cookie should display the result as the username, session id and role of the user in the console window.
Actual Result	The cookie displayed the result as the username, session id and role of the user in the console window.
Remarks	The test was successful.

Before logging in

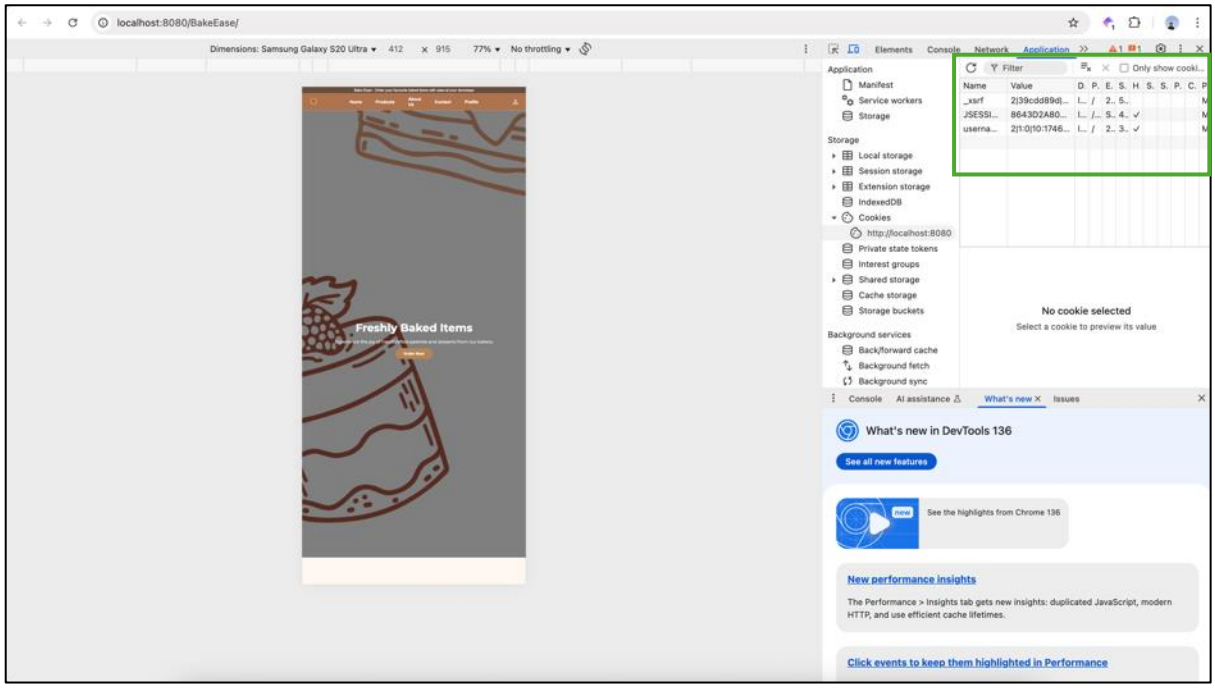


Figure 142: cookie testing before logging in

After Logging in

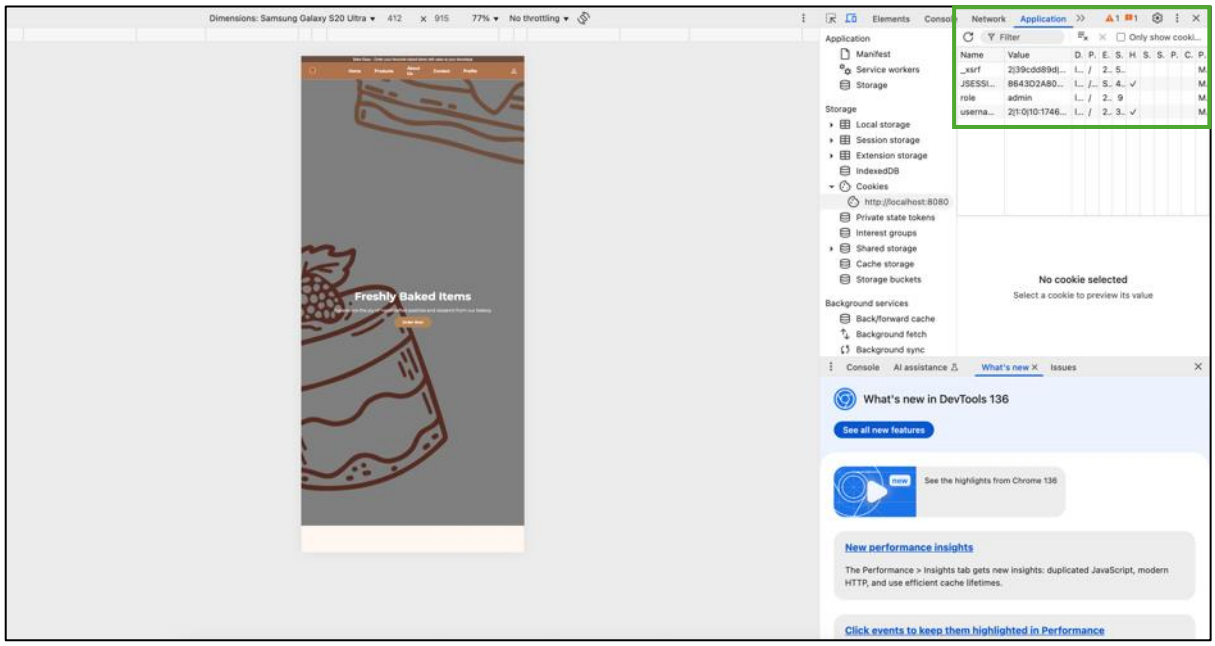


Figure 143: testing cookie after logging in

5.9. Test 8: Filter Testing

Table 9: Filter testing

Test Name	Filter Testing
Test Description	The filter takes the role from the cookie and then creates a barrier between the customer and admin which means there are certain features that only the admin can operate such as admin dashboard.
Actions Performed	The user should login or register and if his/her role is “admin” only then they should be able to access admin dashboard and to access profile page the user should login compulsorily.
Expected Result	The filter should allow only the user with the role as “admin” to access the admin dashboard.
Actual Result	The filter allowed only the user with the role as “admin” to access the admin dashboard.
Remarks	The test was successful.

Logging in as Customer

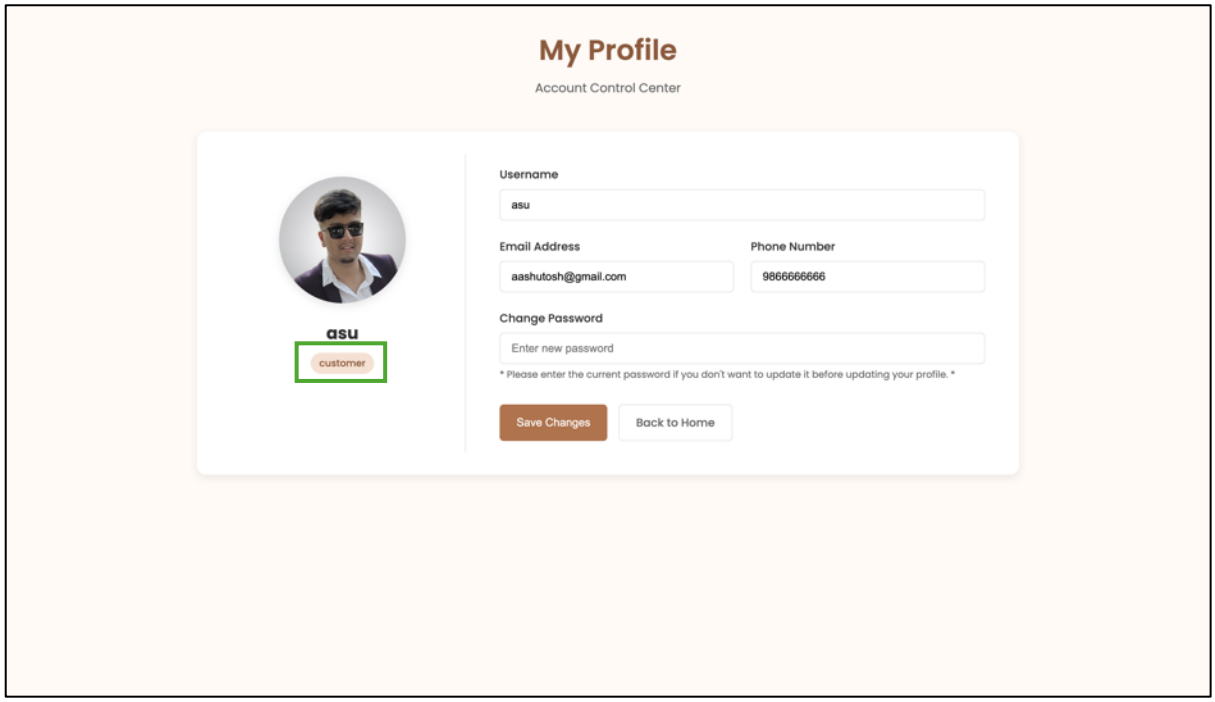


Figure 144: logging in as a customer

Trying to access admin dashboard as a customer

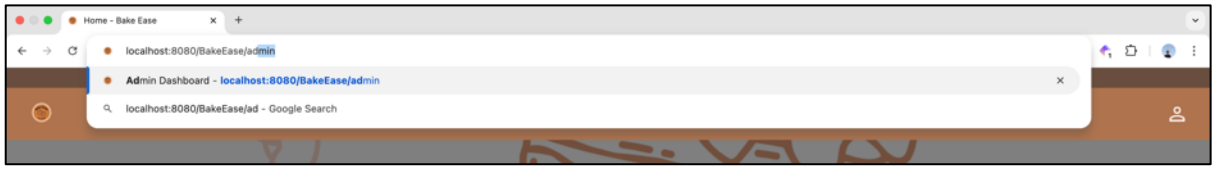


Figure 145: trying to access admin dashboard as a customer

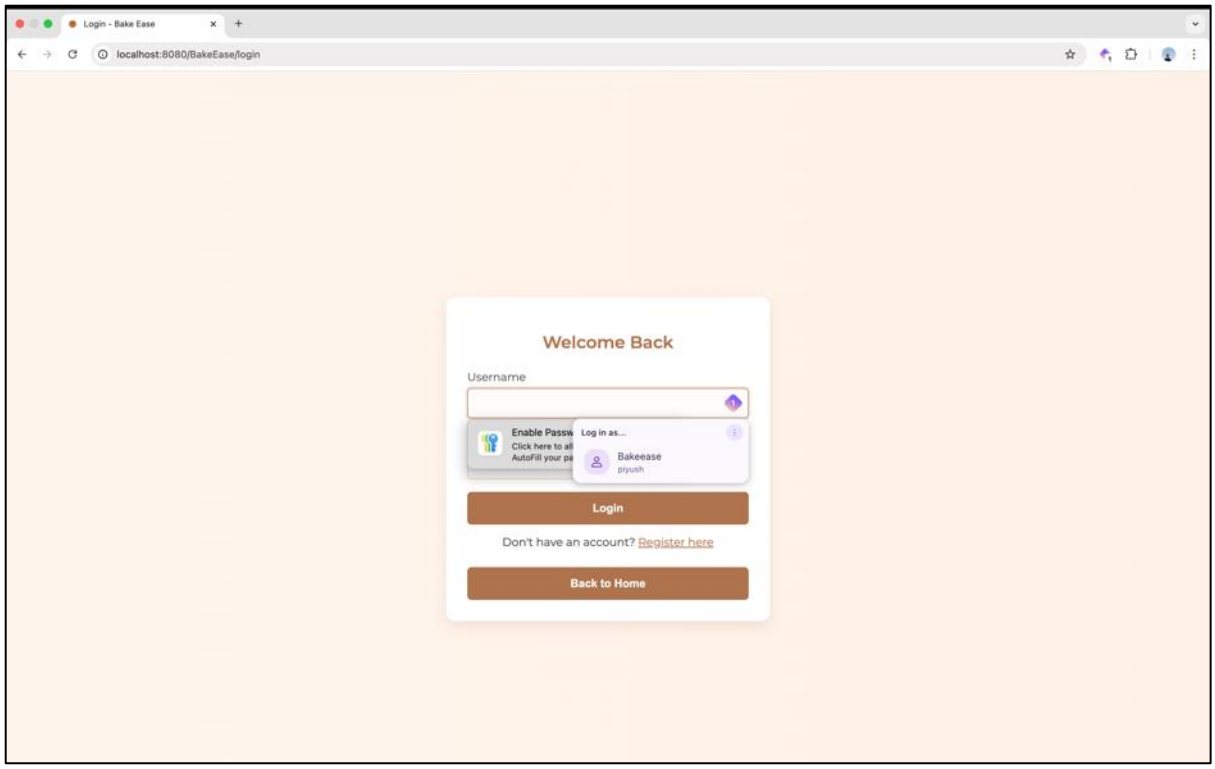


Figure 146: login redirection

Here, we saw that when a customer tried to access the admin dashboard it redirects to login page to login with the admin credentials.

Logging in as admin

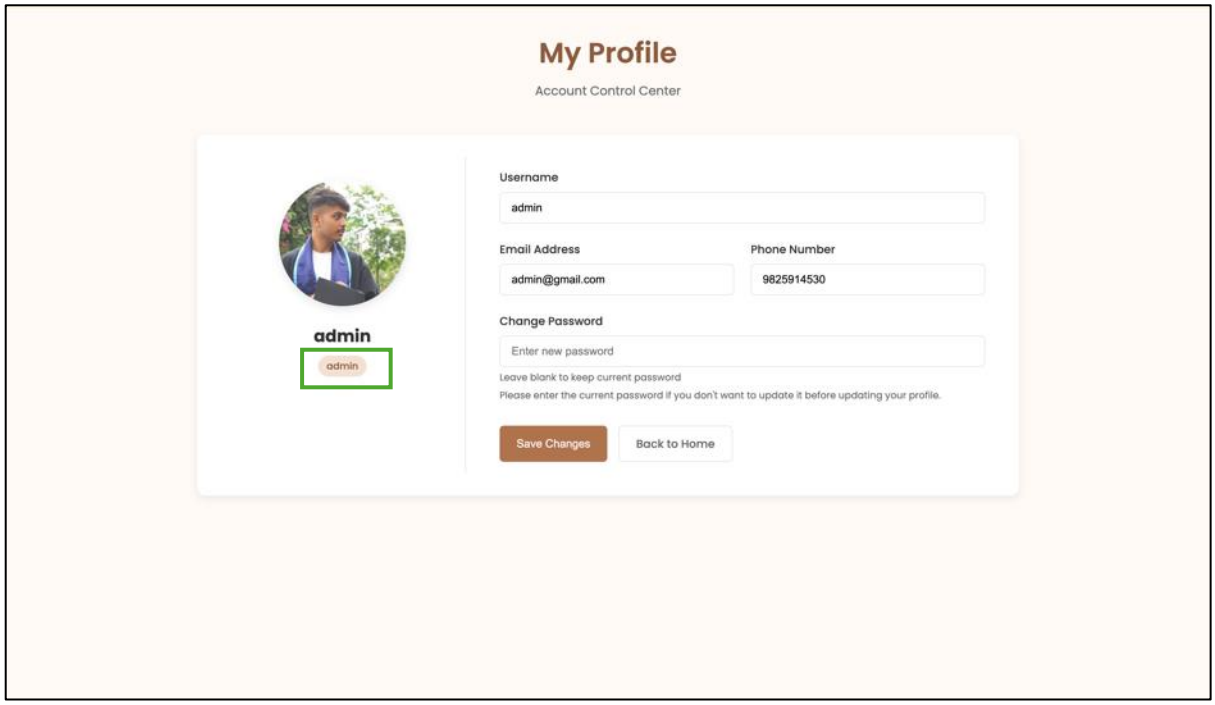


Figure 147: logging in as admin

Trying to access admin dashboard as admin

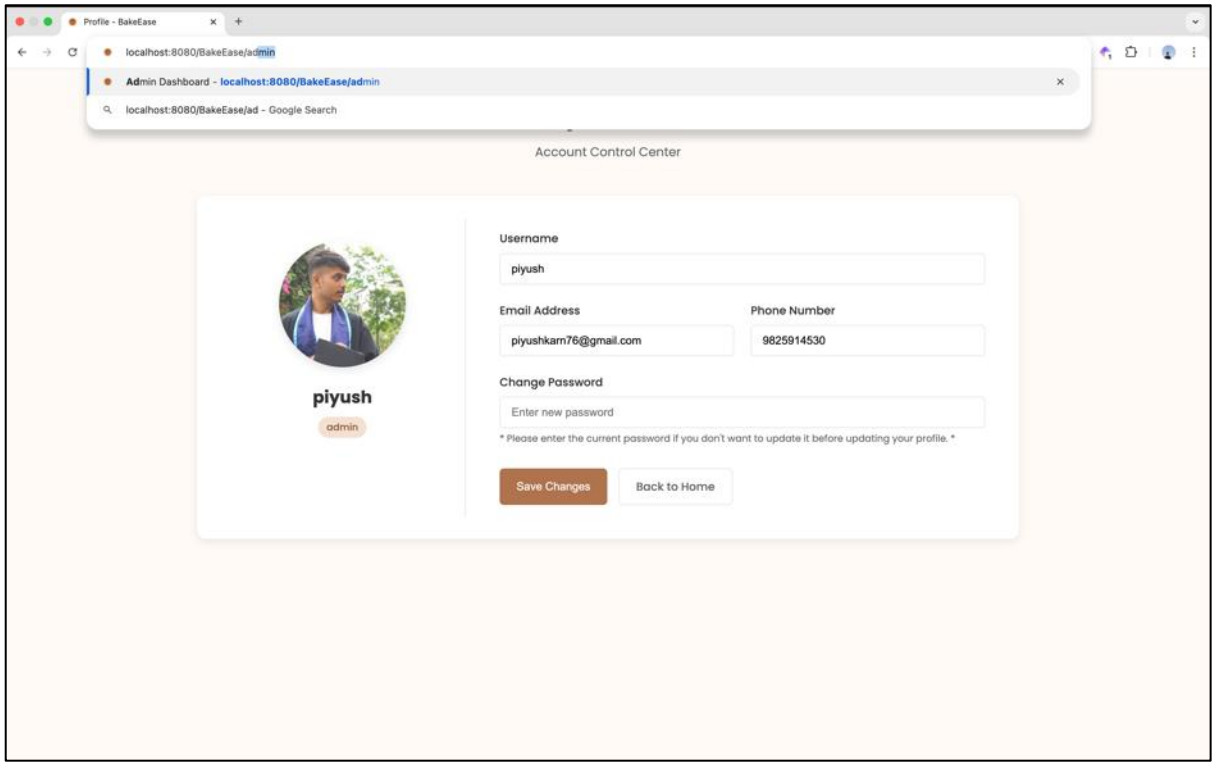


Figure 148: accessing admin dashboard as admin

Successfully redirected to Admin Dashboard

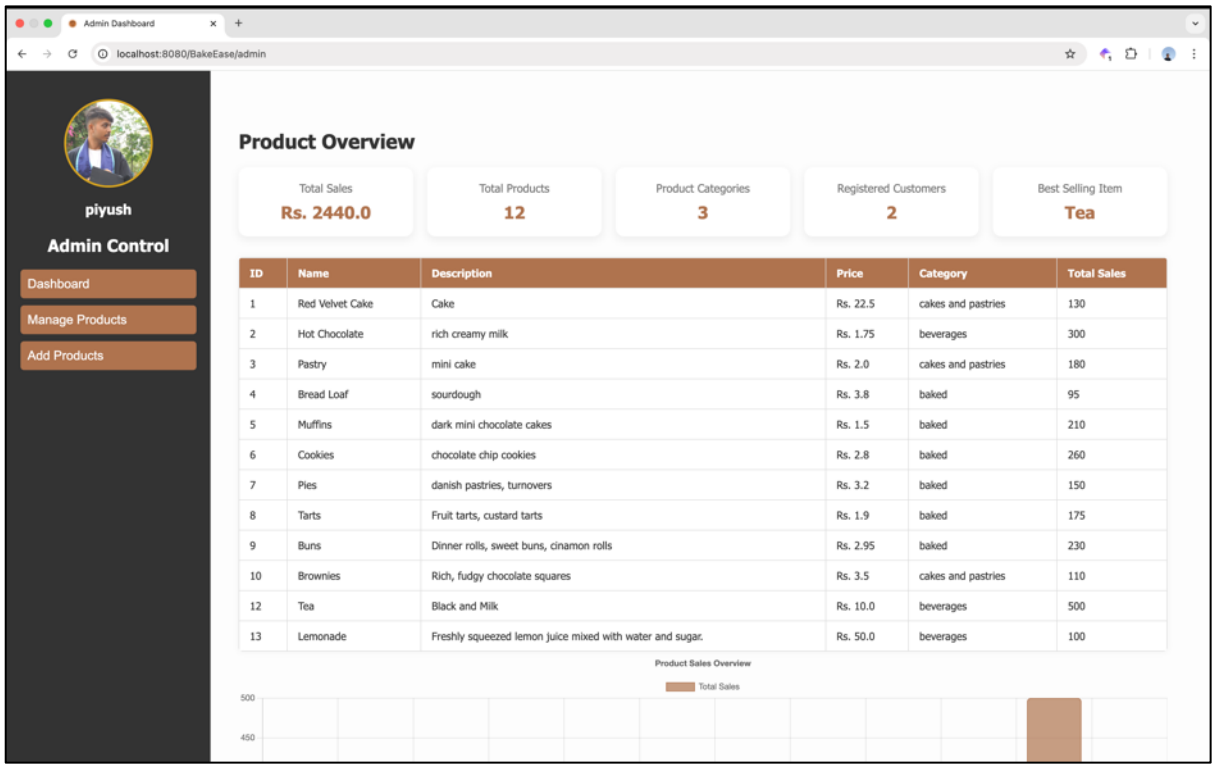


Figure 149: admin page redirection

Here, we saw that when we tried logging in with the admin credentials the page successfully navigated to admin dashboard.

5.10.Test 9: Profile Data Fetch

Table 10: Profile data fetch test

Test Name	Profile Data Fetch
Test Description	This feature is supposed to fetch the data of the user credentials and display it in the profile page.
Actions Performed	After logging in, the profile page displays the user credentials in the profile page which includes username, email, phone number, profile image and role.
Expected Result	The profile page should display the user credentials fetched from the database.
Actual Result	The profile page displays the user credentials fetched from the database.
Remarks	The test was successful.

Target User

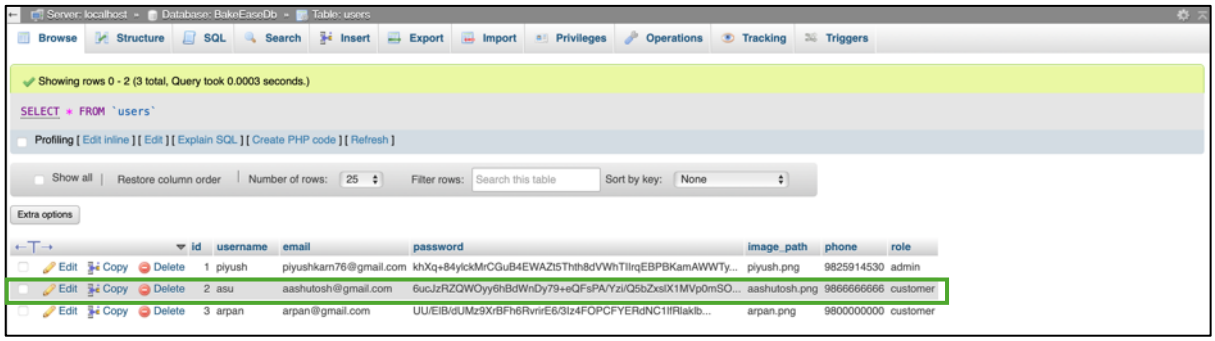


Figure 150: target user for the profile page

Logging in with the target user

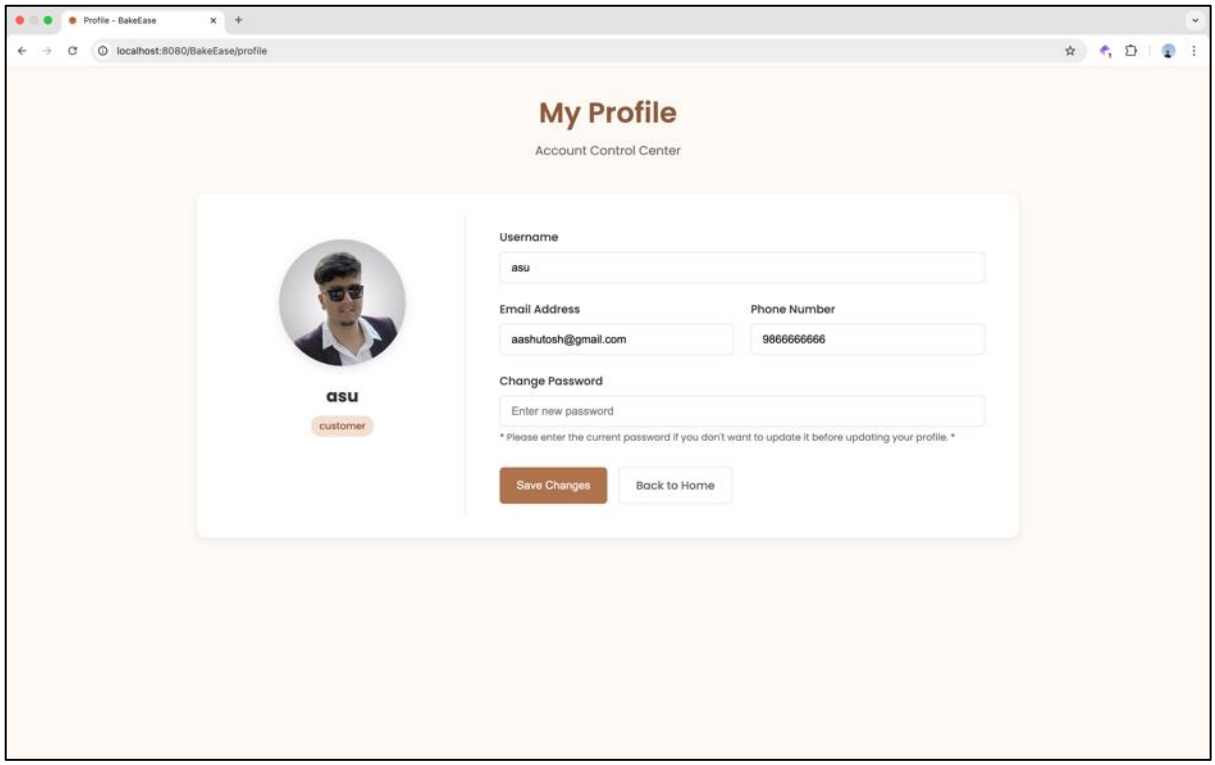


Figure 151: logging in with the target user

Here from the database, we selected a target user and logged in with its user credentials and as a result we saw the target user profile information fetched from the database.

5.11.Test 10: Profile Update

Table 11: Profile update test

Test Name	Profile Update
Test Description	This feature is supposed to update the user credentials and reflect those changes in the database.
Actions Performed	The user credentials of the target user is updated and then saved, hence those changes should reflect in the database.
Expected Result	The updated user information should be displayed in the database of the target user.
Actual Result	The updated user information is displayed in the database of the target user.
Remarks	The test was successful.

Before updating credentials of the target user

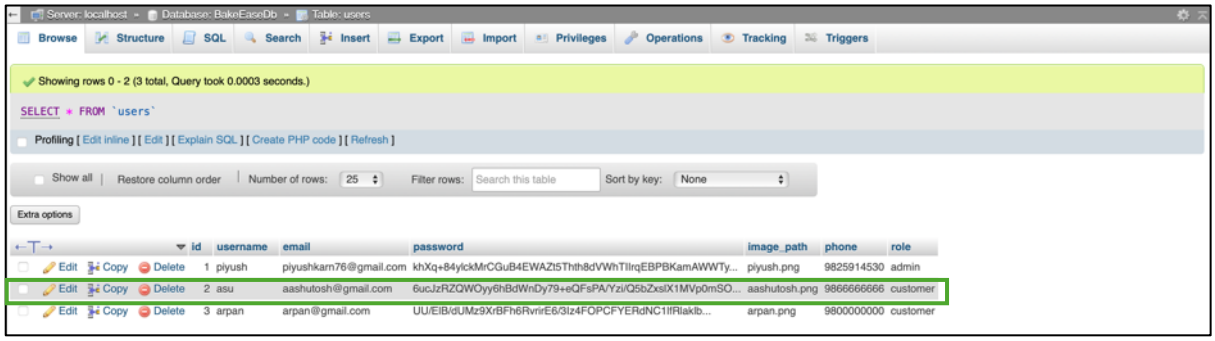


Figure 152: target user for profile update

Current Profile credentials of the target user

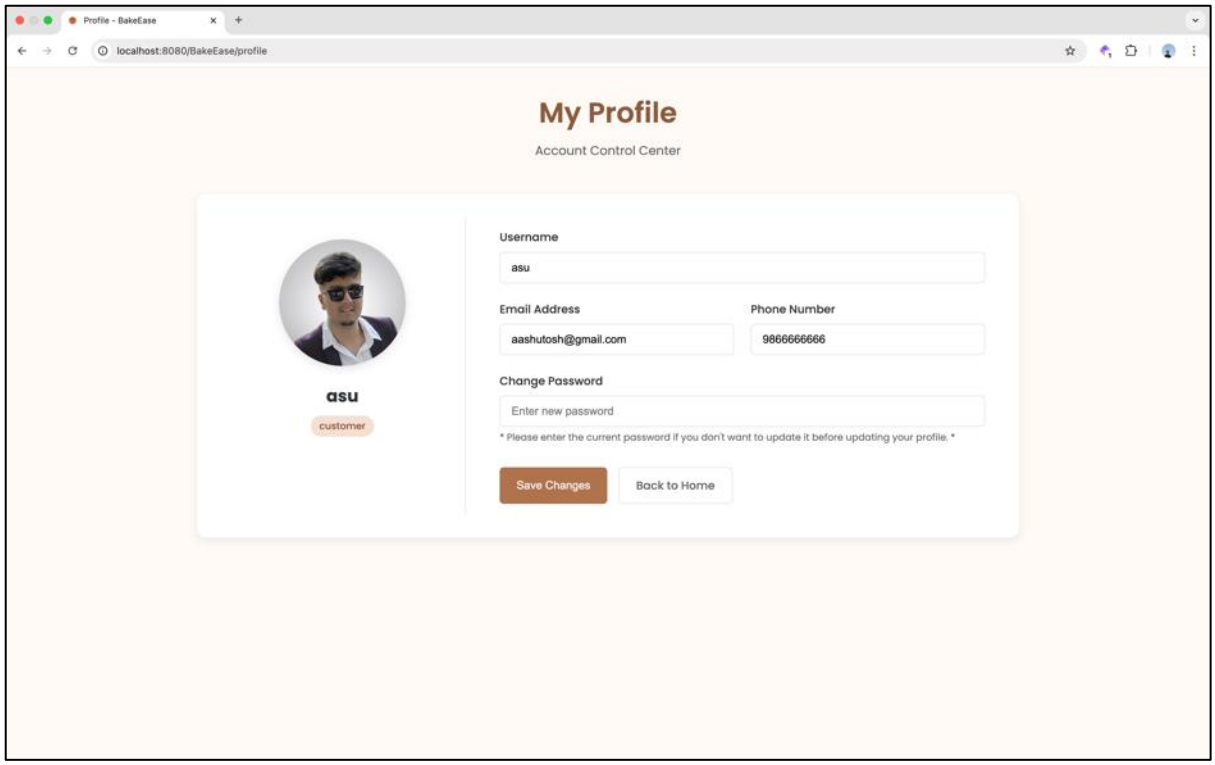


Figure 153: Current user credentials of the target user

Updating credentials of the target user

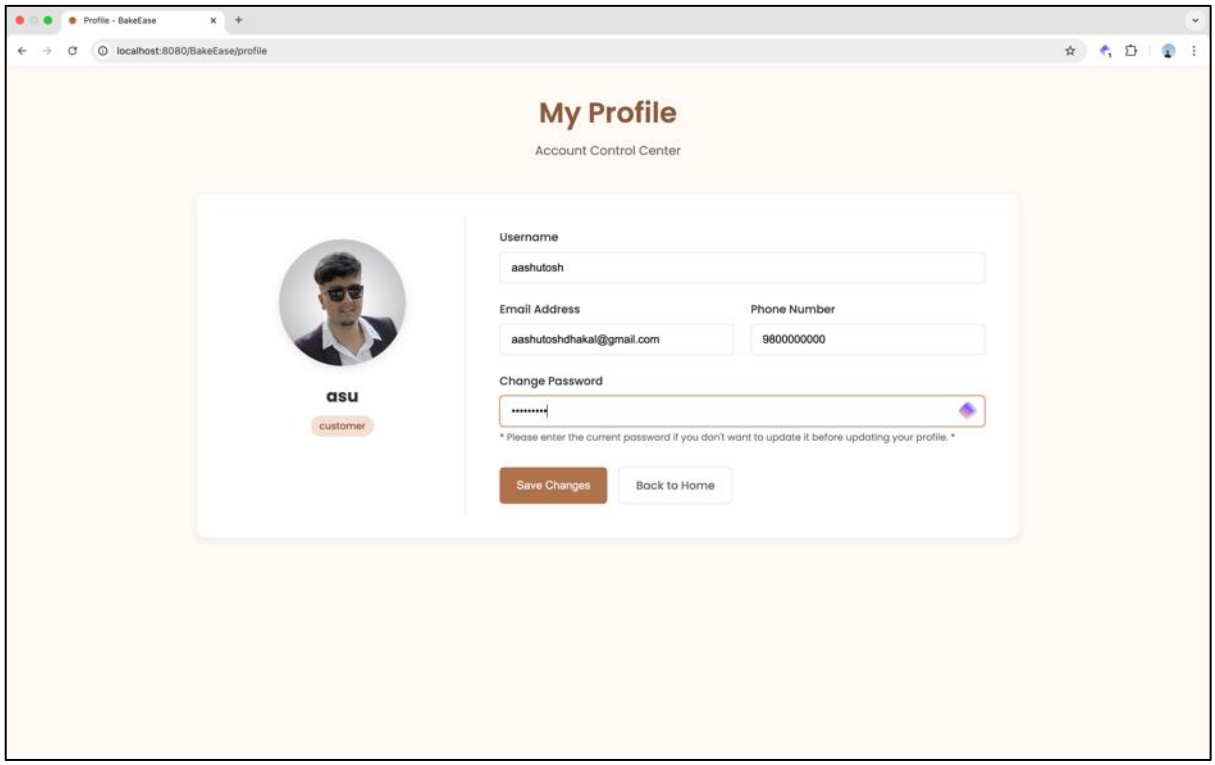


Figure 154: updating credentials of the target user

Updated user credentials of the target user

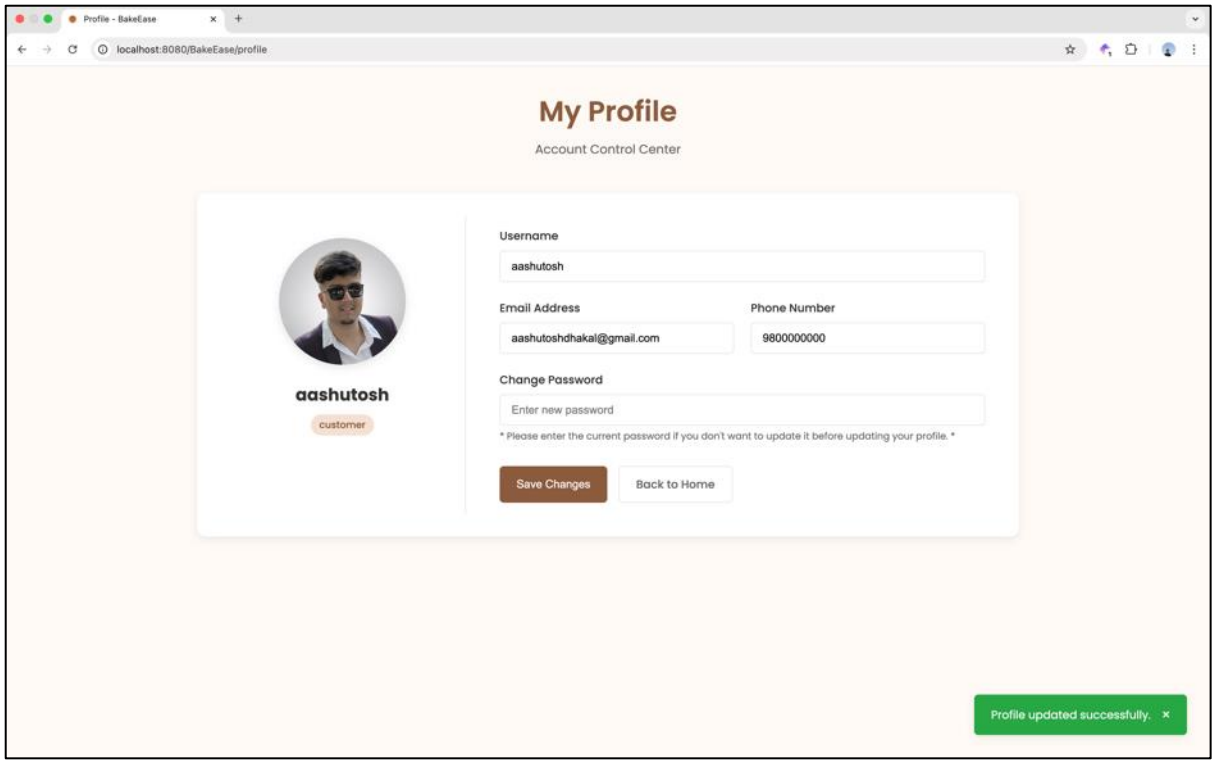


Figure 155: updated user credentials of the target user

After Updating User credentials of the target user

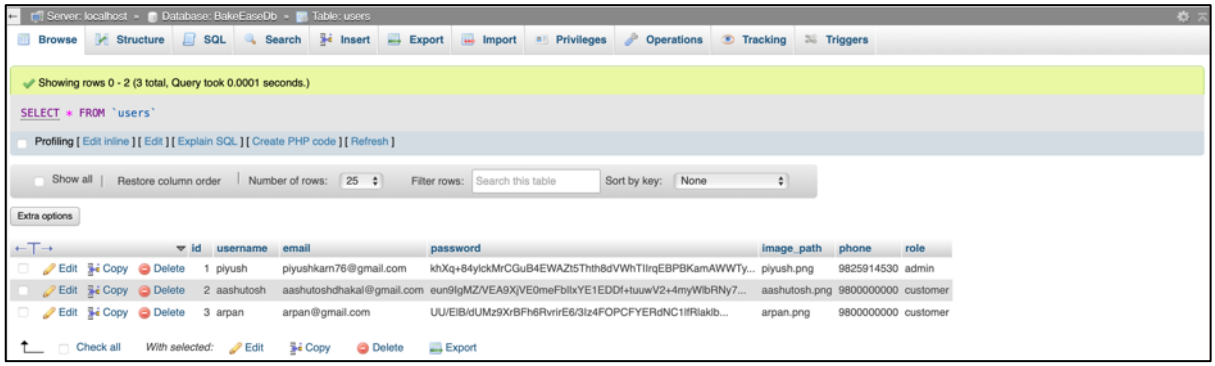


Figure 156: after updating user credentials of the target user

6. Coursework Development and Analysis

The software is about bakery management system which includes features like customer and admin filtration, database management, cookies management, session management, image filtration and much more. The software aims to provide admin and customer filtration which allows user to view the software and admin can make changes in the database also in the web page but requires authentication.

6.1. UI/UX Design and Development

While designing the UI/UX, we used an online designing tool named Canva. (Canva, n.d.) It is an online designing tool which allows users to create their own creative design elements providing large number of assets including elements, shapes, fonts, color options, custom size support etc. In the context of the coursework, we have used Canva to develop wireframes of all the pages which are being shown in the web page. To check the wireframes of all the pages you can visit point number 3 named as Wireframes and Design. Click on the highlighted part to reach the destined part of the document.

For designing the frontend of the project we have used Eclipse IDE, it is an opensource IDE made for developers who are using different technologies at the same time and want multiple features for their project in a single code editor. (IDE, n.d.)

6.2. Register Feature

6.2.1. Completion of the feature

The register feature requires text fields for username, email, password, confirm password, phone number and a profile image. All text fields are validated with their empty fields' detection and correct data type validation. It registers the user as a customer by default, to change the role of the user we manually have to change it from the database.

6.2.2. Internet Support

For developing the wireframe of the page, we used Canva and for developing the frontend we used Eclipse Ide. Here, while designing the wireframes we did not face many errors as designing the wireframes but however while designing the frontend

and making the login page dynamic we took help from the GitHub repository that our module leader had provided us to make our coding part easier for our project. (Maharjan, 2025)

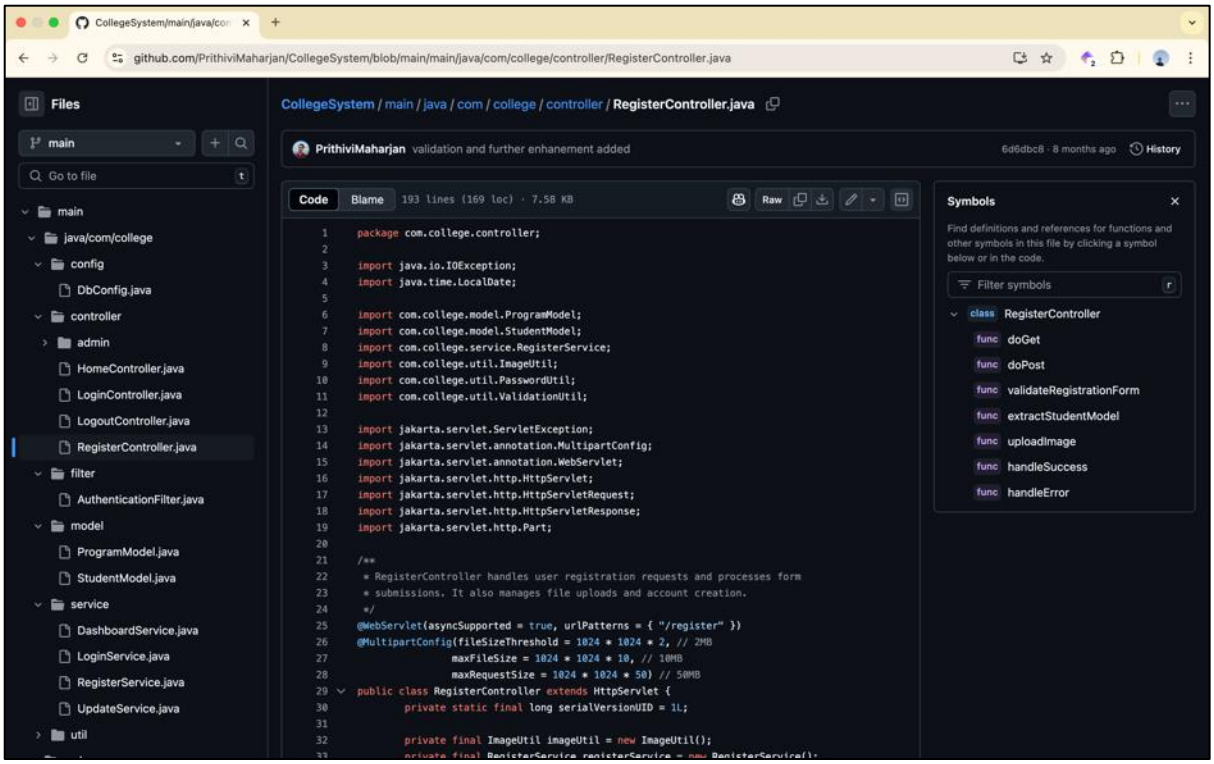


Figure 157: register feature support

6.2.3. Critical Analysis

6.2.3.1. Challenges

The challenges I faced while developing this feature is the page reload problem where after entering correct details the page reloads instead of performing correct data type validation.

6.2.3.2. Problem faced

6.2.3.2.1. Page Reload Problem

In this problem, we saw that empty fields validation works completely fine and wrong data type validation also works completely fine but the after the correct data is entered the page reloads and all the data in the text fields is cleared and an error message is seen which says invalid credentials.

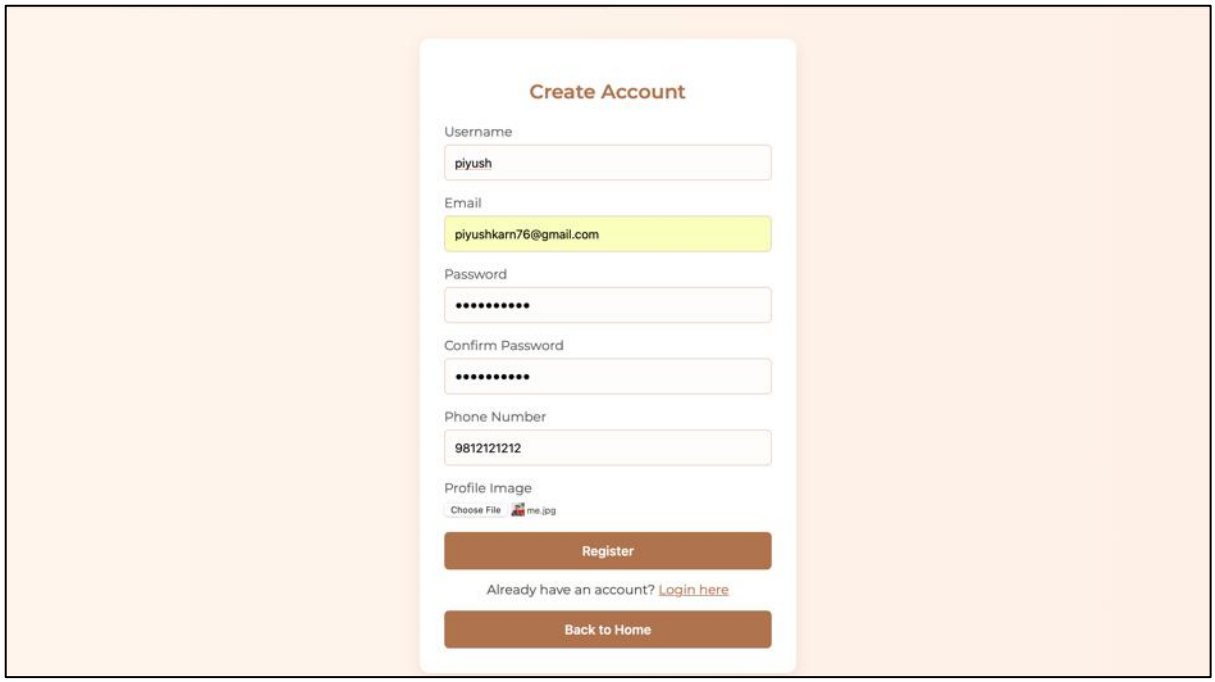


Figure 158: problem faced (before)

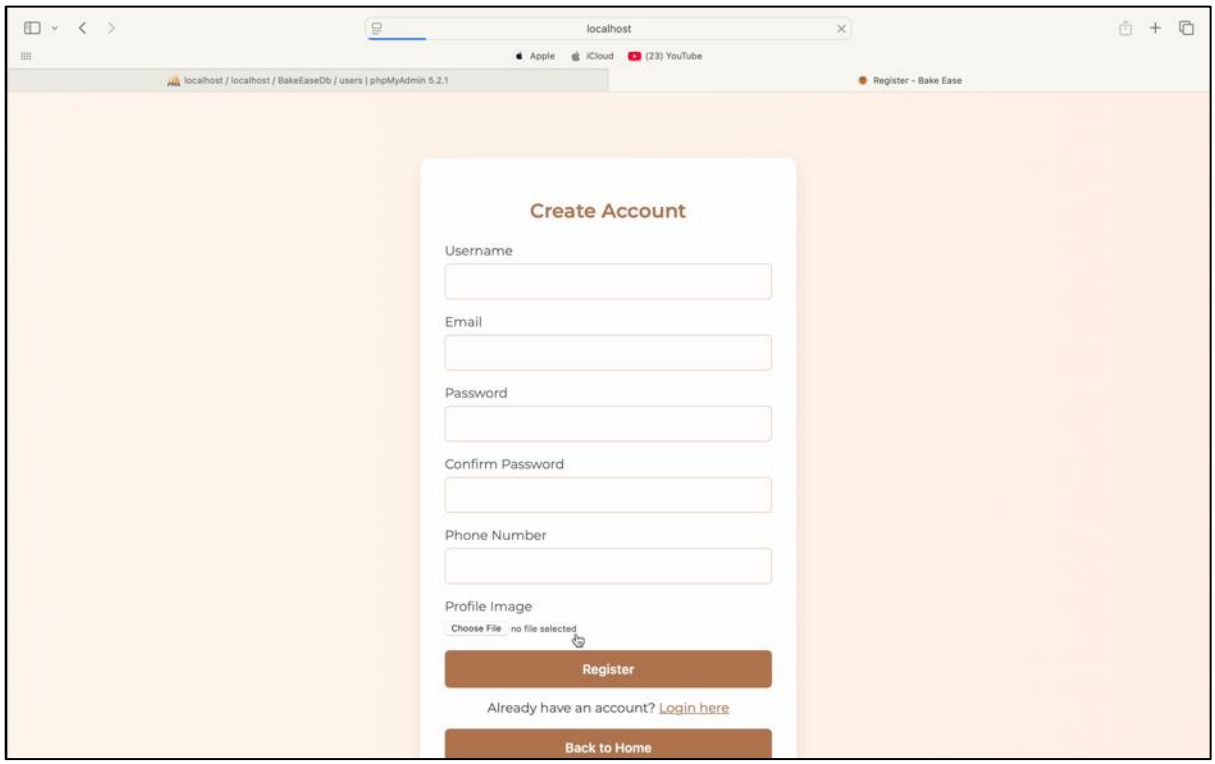


Figure 159: problem faced (after)

Here, we can see that when the user clicks the register button the page reloads clearing all the text fields and nothing is done. Neither validation nor anything.

The problem was at the forwarding message section of the code.

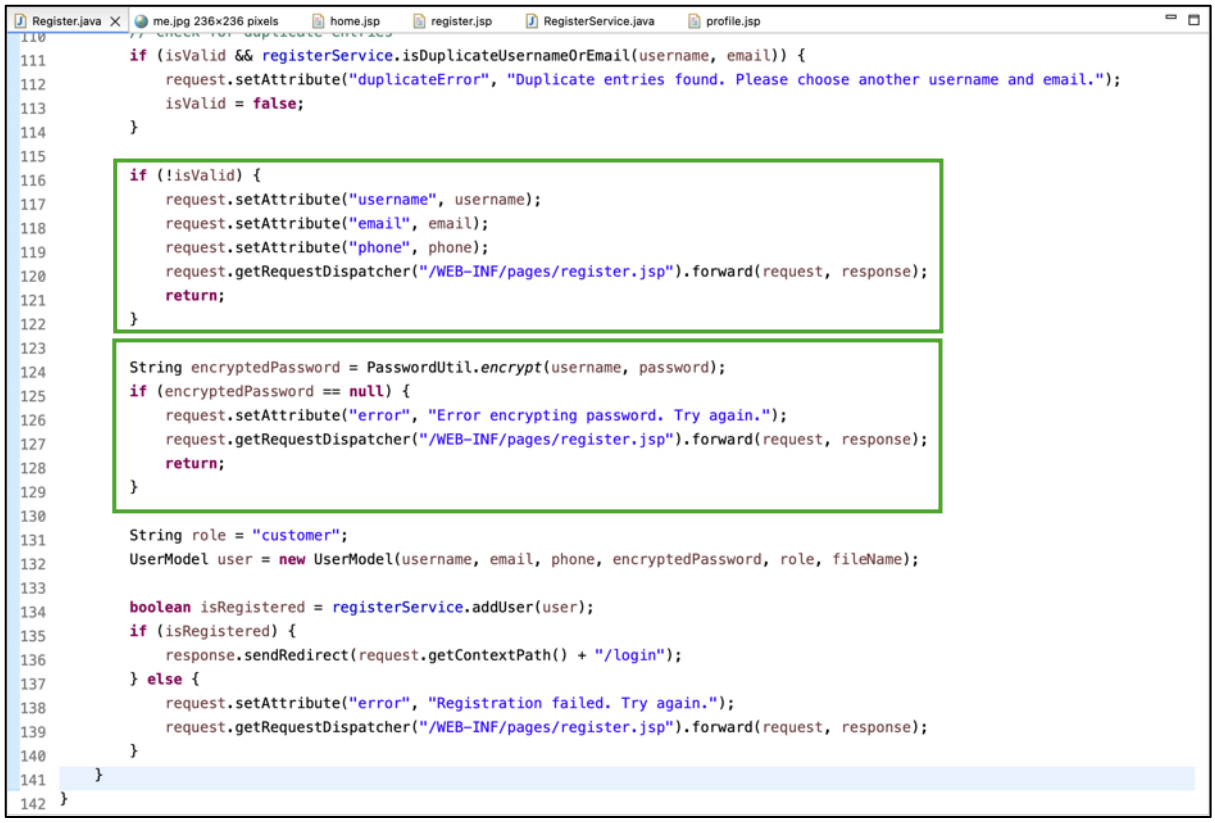


Figure 160: solution to register problem

This is the key part to solve the problem. The function checks whether the form inputs failed any validation inputs. Previously it was set to false condition which led to this problem. Now after solving the problem the website navigates to login page and data is also seen in the database. Since this problem occurred after performing password util, the green highlighted part of the screenshot are the reasons to fix the error.

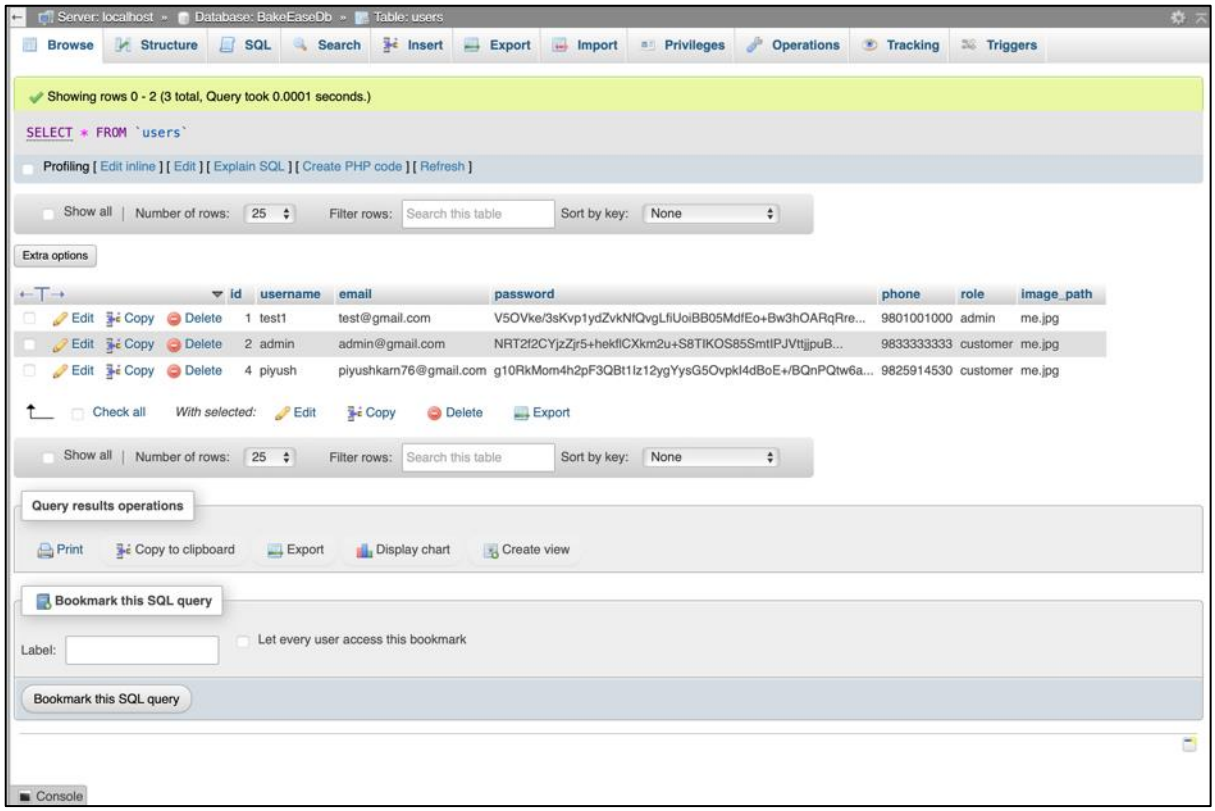


Figure 161: database condition after solving the issue

6.3. Login Feature

6.3.1. Completion of the feature

The login feature requires two text fields for username and password and two buttons where one button is used to submit the data entered in the text field and another button is there to navigate to home page to make it easier for the user. Also, there is a hyperlink to navigate to register page in case the user is not registered.

6.3.2. Internet Support

The GitHub repository that was provided to us by our tutor is the only internet source that helped me to solve the errors in this page and develop the overall page.

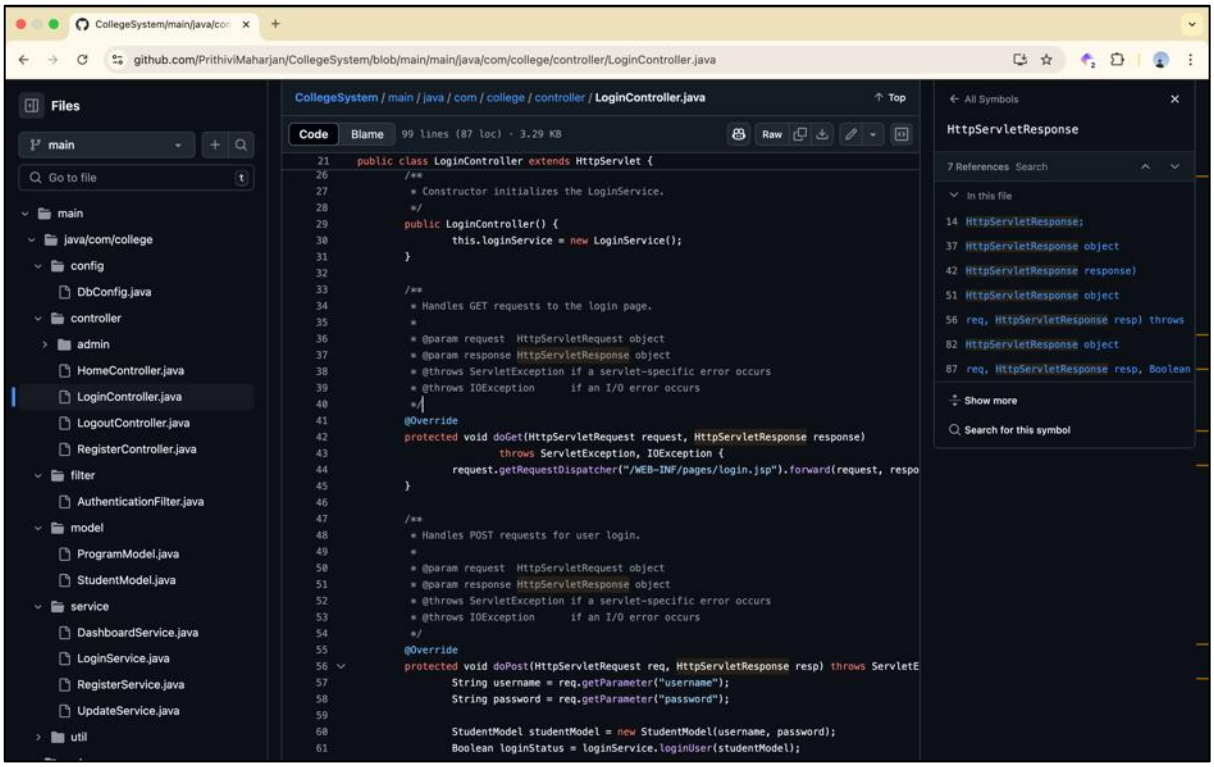


Figure 162: login feature solution

6.3.3. Critical Analysis

6.3.3.1. Challenge

Here, the challenge was to retrieve the data from the database, decrypt the password using the password util file and then verify the customer or admin using authentication filter file.

6.3.3.2. Problems Faced

6.3.3.2.1. Correct Data type Validation

Here, even after entering the correct data in the username and password field it showed error saying invalid username or password.

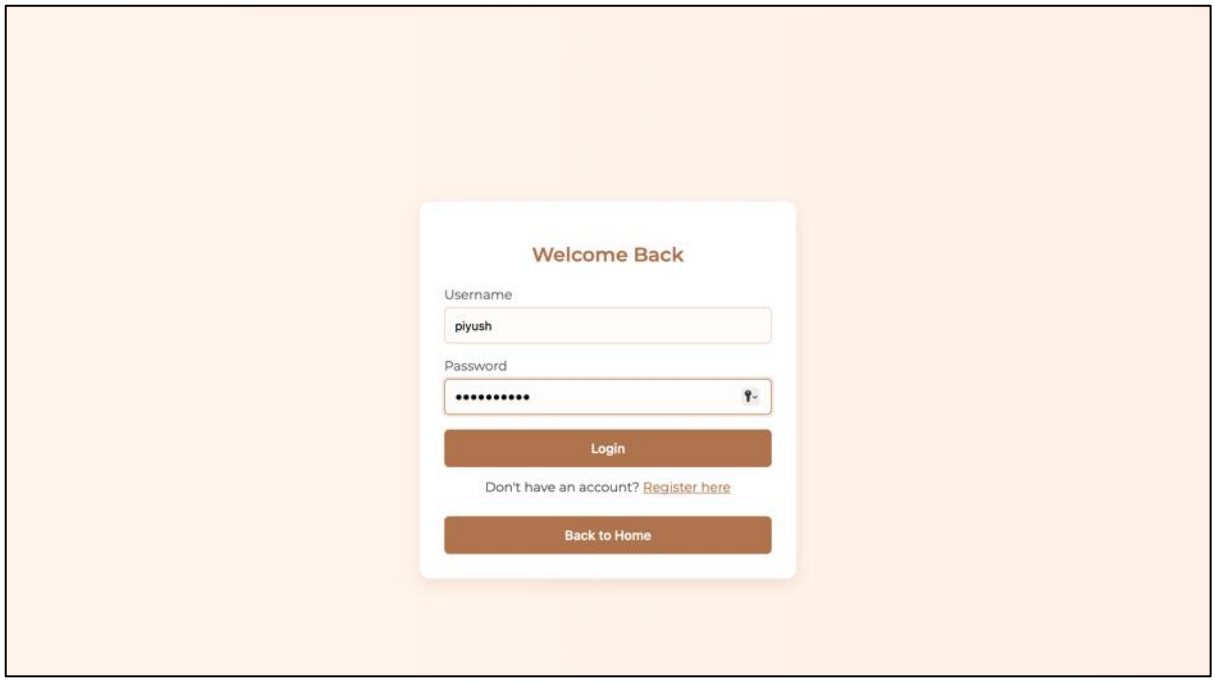


Figure 163: Evidence of the login correct data type validation error (1)

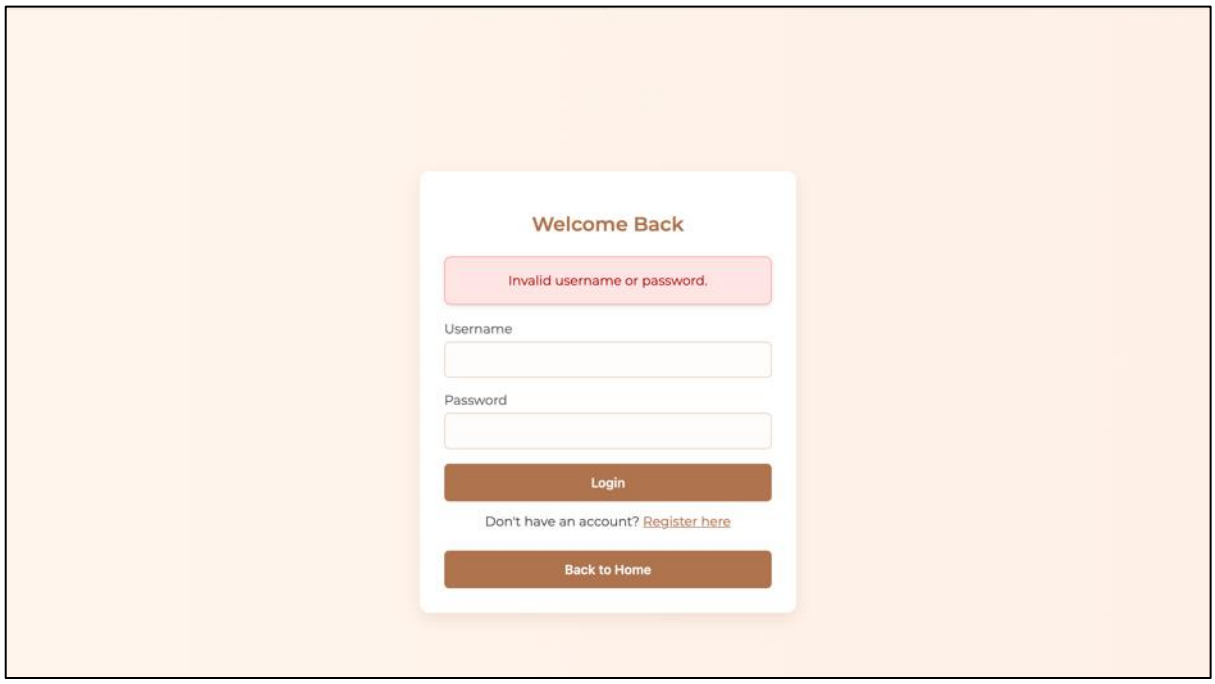


Figure 164: evidence of correct data in login error (2)

Here, the username “piyush” is already registered in the database whose password is “Piyush@123”. But after entering correct data, the page reloads to show invalid username or password error.

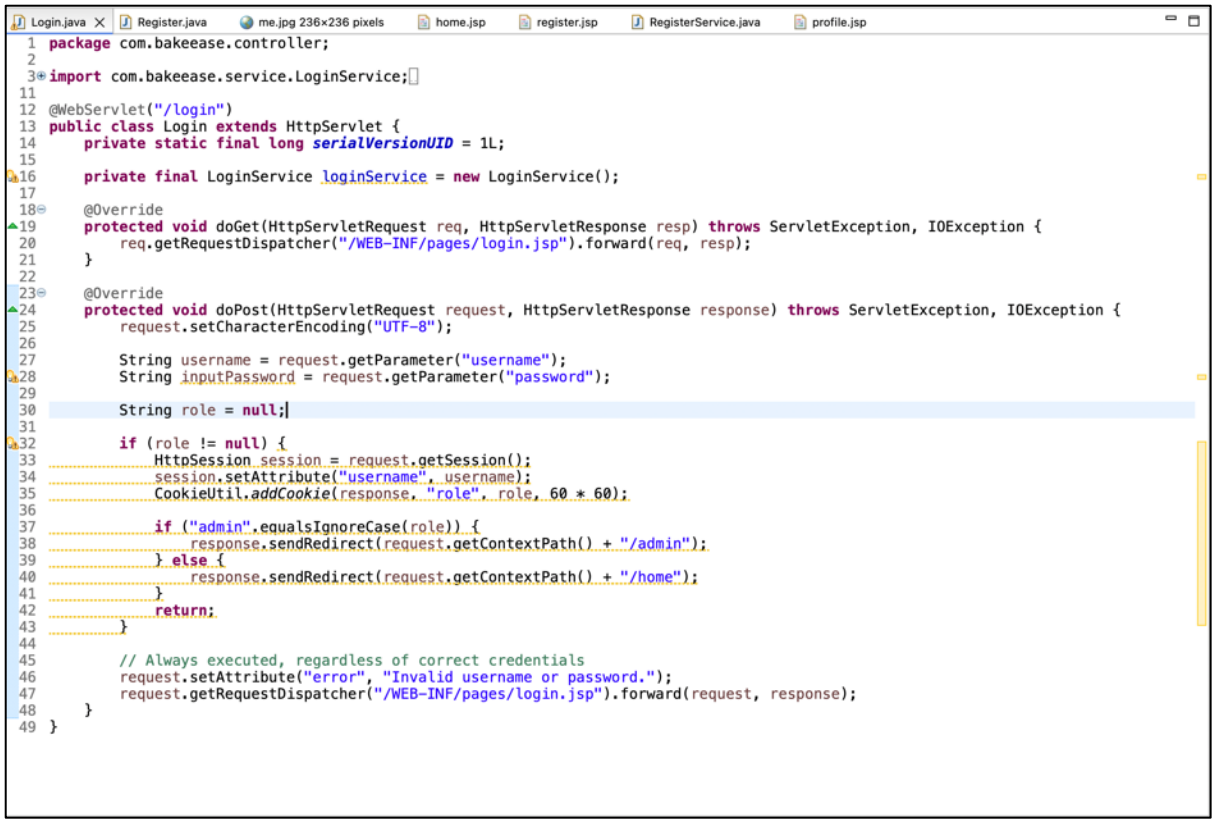


Figure 165: cause of the error in login page

The cause of the problem was the role was set to null which led to fail credentials and show invalid username and password error message in the web page. After taking reference from the college app GitHub repository, It was the specific line of code that fixed the error.

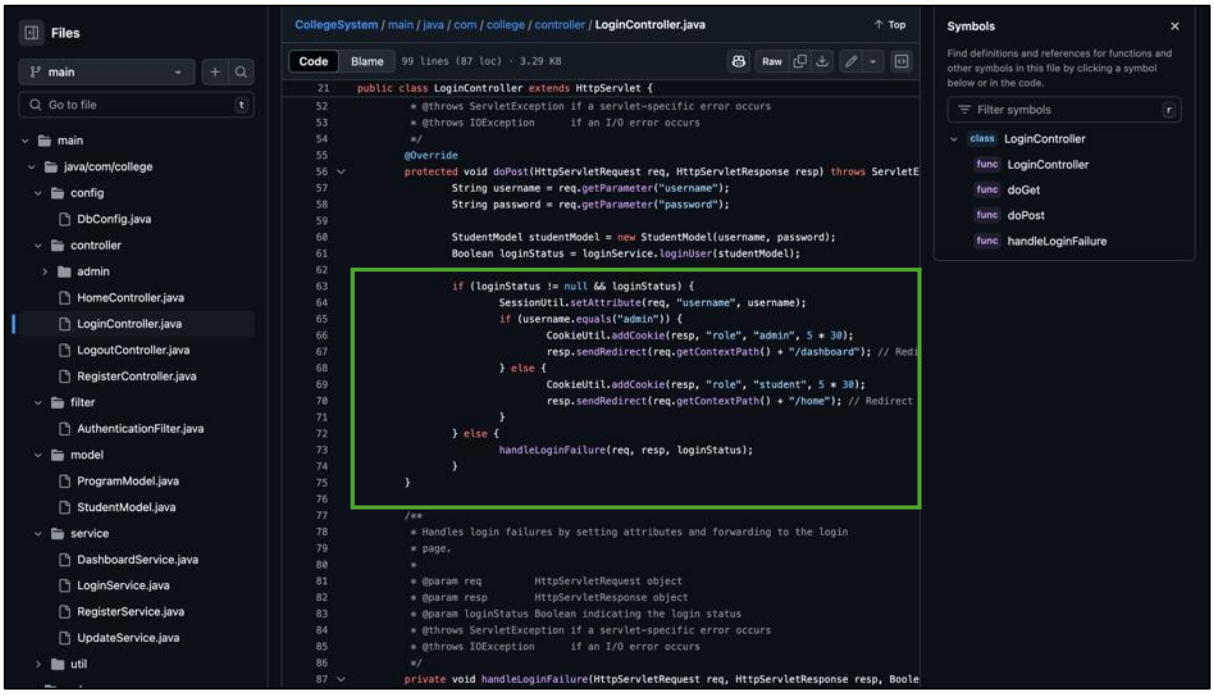


Figure 166: fixing the login error from GitHub

Here the cookie util code that checks the role and login status condition helps to fix the error. The same code is done in our project source code.

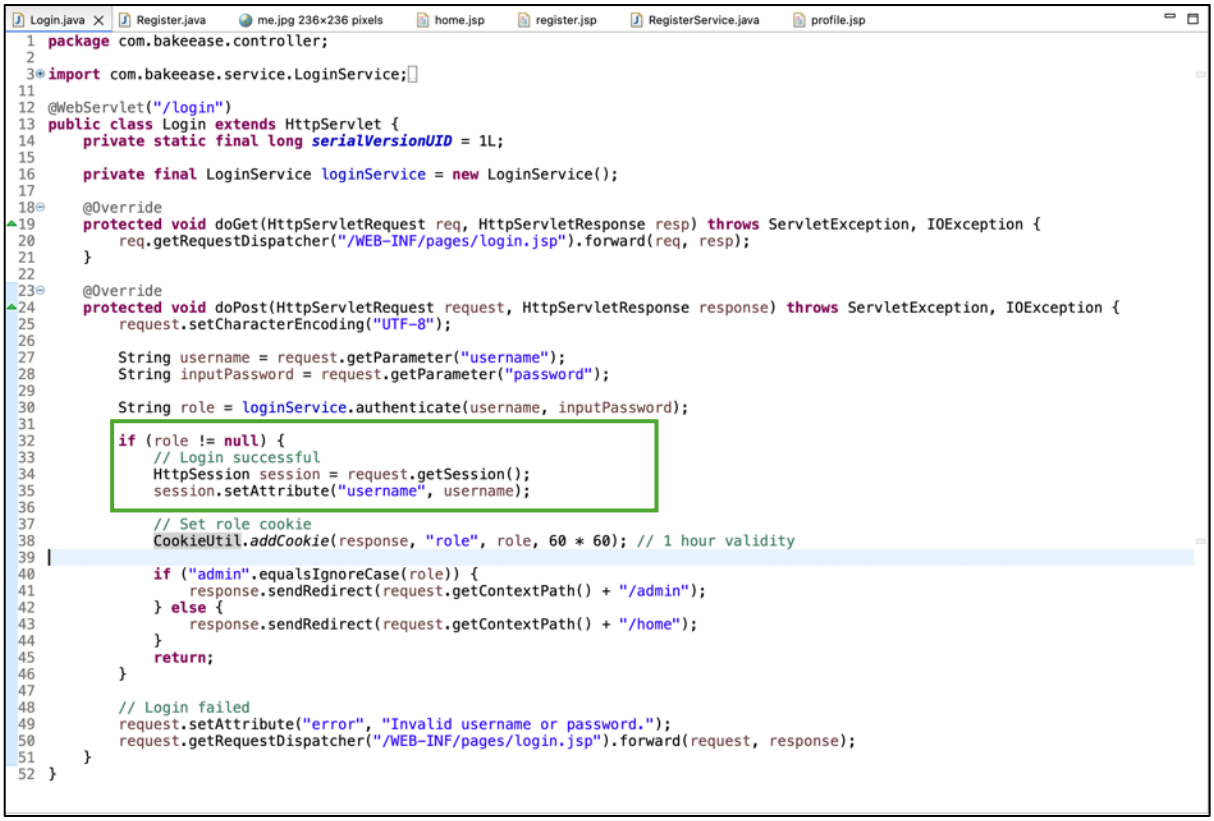


Figure 167: solution of the login error

After the role is identified, the login fields are successfully validated and hence allows user to access the pages according to their role. Hence customer cannot access admin dashboard page whereas admin can access that page etc.

6.4. Profile Feature

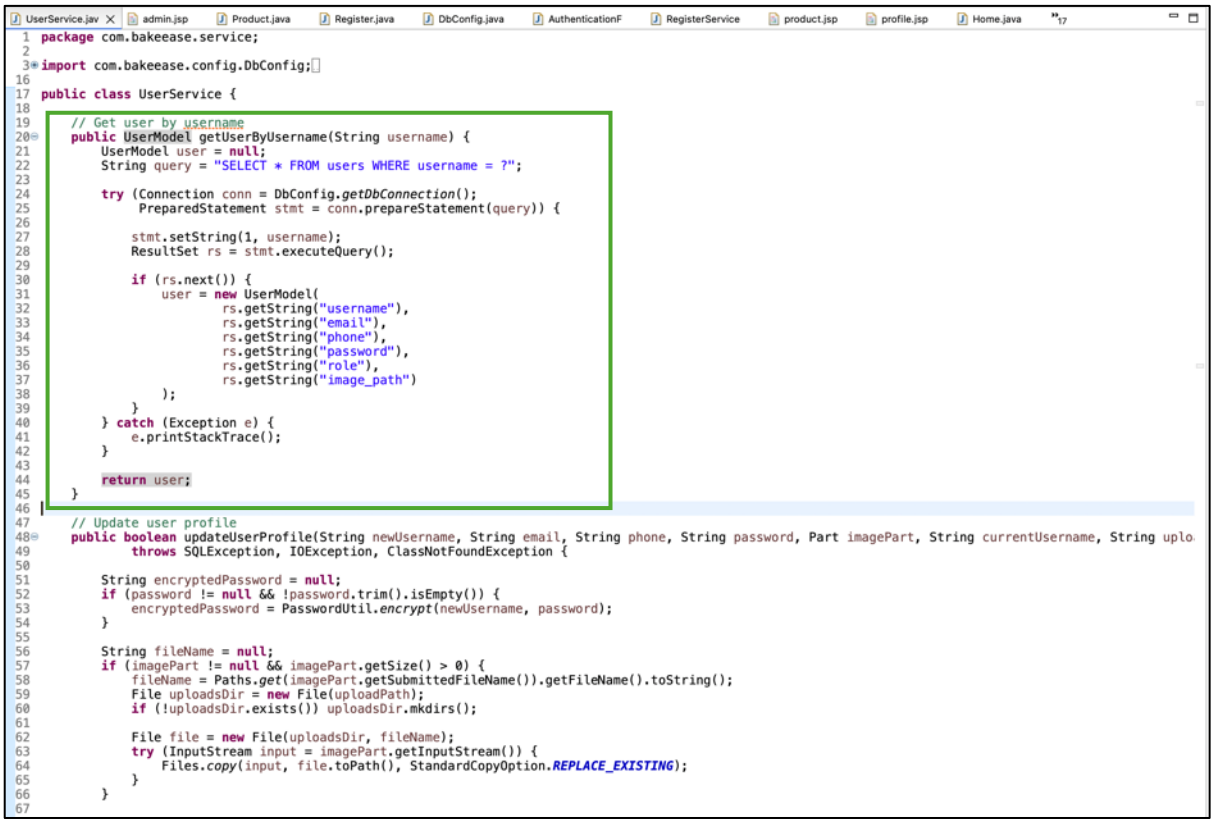
The profile feature is used to show the user their credentials and also allow user to update their user credentials from the account control center that is profile page.

6.4.1. Completion of the feature

The profile page contains the credentials of the user and provides real time text fields to update the user credentials such as username, password, email, phone number and profile image.

6.4.2. Internet Support

To fetch the user details, it didn't require much effort as the user details was retrieved from the service package's method which retrieves the data from the database using select query in the database and store it in model file.



```
UserService.jav | admin.jsp | Product.java | Register.java | DbConfig.java | AuthenticationF | RegisterService | product.jsp | profile.jsp | Home.java
1  package com.bakeease.service;
2
3  import com.bakeease.config.DbConfig;
16
17 public class UserService {
18
19     // Get user by username
20     public UserModel getUserByUsername(String username) {
21         UserModel user = null;
22         String query = "SELECT * FROM users WHERE username = ?";
23
24         try (Connection conn = DbConfig.getDbConnection();
25              PreparedStatement stmt = conn.prepareStatement(query)) {
26
27             stmt.setString(1, username);
28             ResultSet rs = stmt.executeQuery();
29
30             if (rs.next()) {
31                 user = new UserModel(
32                         rs.getString("username"),
33                         rs.getString("email"),
34                         rs.getString("phone"),
35                         rs.getString("password"),
36                         rs.getString("role"),
37                         rs.getString("image_path")
38                 );
39             }
40         } catch (Exception e) {
41             e.printStackTrace();
42         }
43
44         return user;
45     }
46
47     // Update user profile
48     public boolean updateUserProfile(String newUsername, String email, String phone, String password, Part imagePart, String currentUsername, String uplo
49             throws SQLException, IOException, ClassNotFoundException {
50
51         String encryptedPassword = null;
52         if (password != null && !password.trim().isEmpty()) {
53             encryptedPassword = PasswordUtil.encrypt(newUsername, password);
54         }
55
56         String fileName = null;
57         if (imagePart != null && imagePart.getSize() > 0) {
58             fileName = Paths.get(imagePart.getSubmittedFileName()).getFileName().toString();
59             File uploadsDir = new File(uploadPath);
60             if (!uploadsDir.exists()) uploadsDir.mkdirs();
61
62             File file = new File(uploadsDir, fileName);
63             try (InputStream input = imagePart.getInputStream()) {
64                 Files.copy(input, file.toPath(), StandardCopyOption.REPLACE_EXISTING);
65             }
66         }
67
```

Figure 168: fetching user data to profile page

To update the user credentials to the database, we have used update method named update user profile which runs update query in the database to update the user credentials. I took reference from the GitHub repository of the college app.

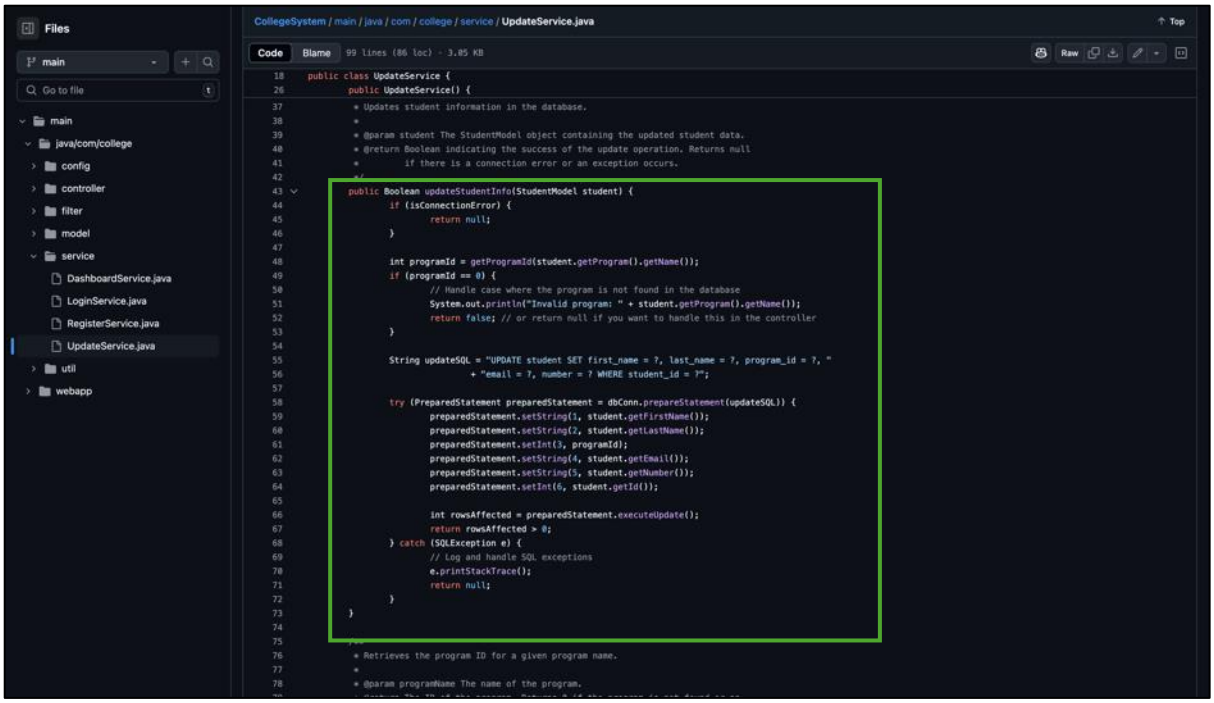


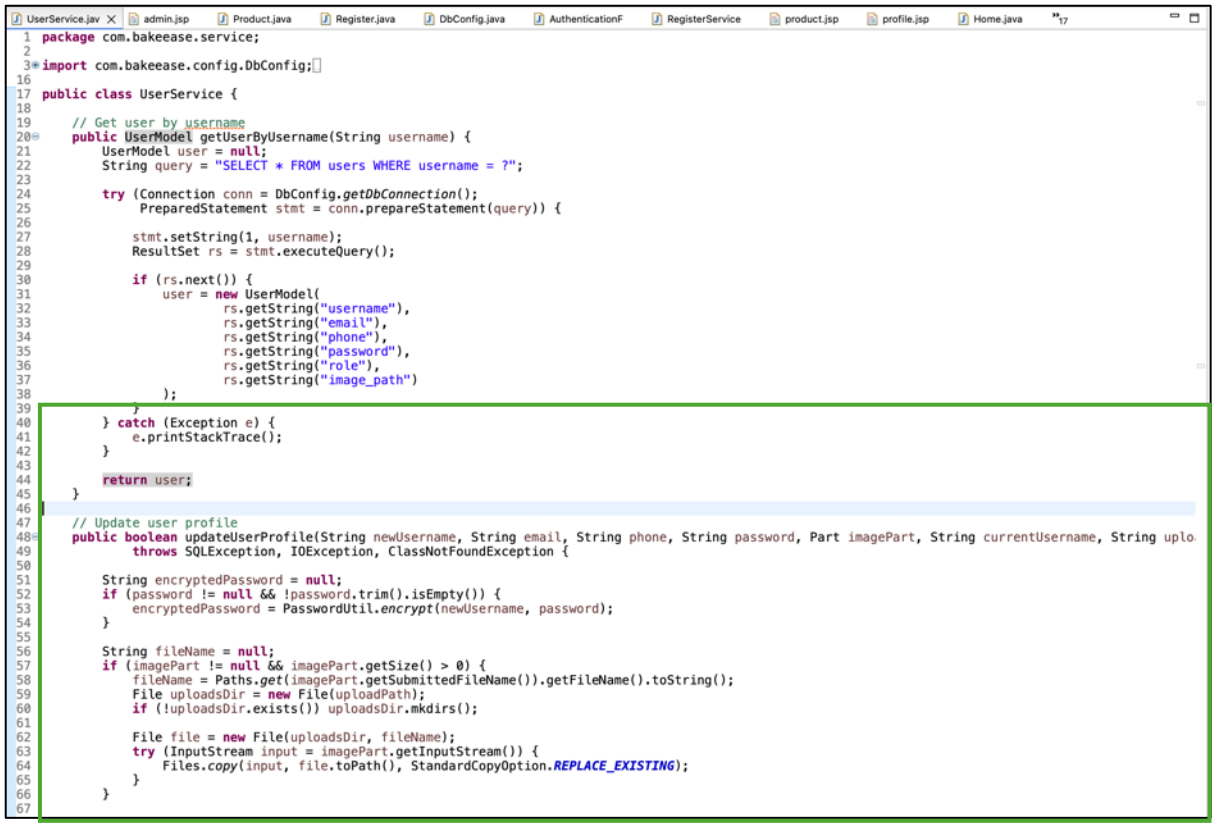
Figure 169: internet support for update user profile

6.4.3. Critical Analysis

6.4.3.1. Challenge

The challenge I faced was to update the profile information to the database. The update method is supposed to be written in such a way that it retrieves the data from the database first then runs the update query and then the updated information should also be updated in the database.

To overcome the challenge, we created a method inside service package named update user profile. It first does password encryption, validates image and runs the update query in the user table in the database returns true value is update was successful.



```java
package com.bakeease.service;

import com.bakeease.config.DbConfig;

public class UserService {

    // Get user by username
    public UserModel getUserByUsername(String username) {
        UserModel user = null;
        String query = "SELECT * FROM users WHERE username = ?";

        try (Connection conn = DbConfig.getDbConnection();
             PreparedStatement stmt = conn.prepareStatement(query)) {

            stmt.setString(1, username);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                user = new UserModel(
                        rs.getString("username"),
                        rs.getString("email"),
                        rs.getString("phone"),
                        rs.getString("password"),
                        rs.getString("role"),
                        rs.getString("image_path")
                );
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return user;
    }

    // Update user profile
    public boolean updateUserProfile(String newUsername, String email, String phone, String password, Part imagePart, String currentUsername, String uplo
            throws SQLException, IOException, ClassNotFoundException {

        String encryptedPassword = null;
        if (password != null && !password.trim().isEmpty()) {
            encryptedPassword = PasswordUtil.encrypt(newUsername, password);
        }

        String fileName = null;
        if (imagePart != null && imagePart.getSize() > 0) {
            fileName = Paths.get(imagePart.getSubmittedFileName()).getFileName().toString();
            File uploadsDir = new File(uploadPath);
            if (!uploadsDir.exists()) uploadsDir.mkdirs();

            File file = new File(uploadsDir, fileName);
            try (InputStream input = imagePart.getInputStream()) {
                Files.copy(input, file.toPath(), StandardCopyOption.REPLACE_EXISTING);
            }
        }
```

Figure 170: update method to update user credentials in the profile page

Output

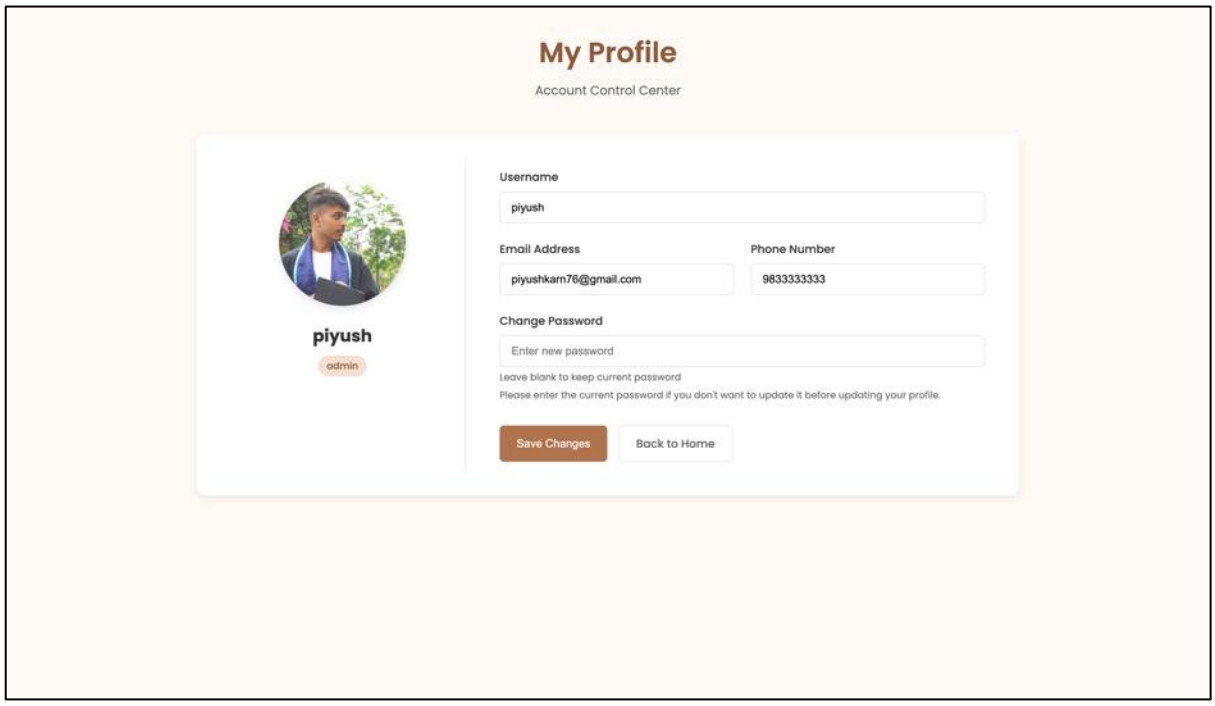


Figure 171: before updating user profile in profile page

Figure 172: before updating user credentials in database

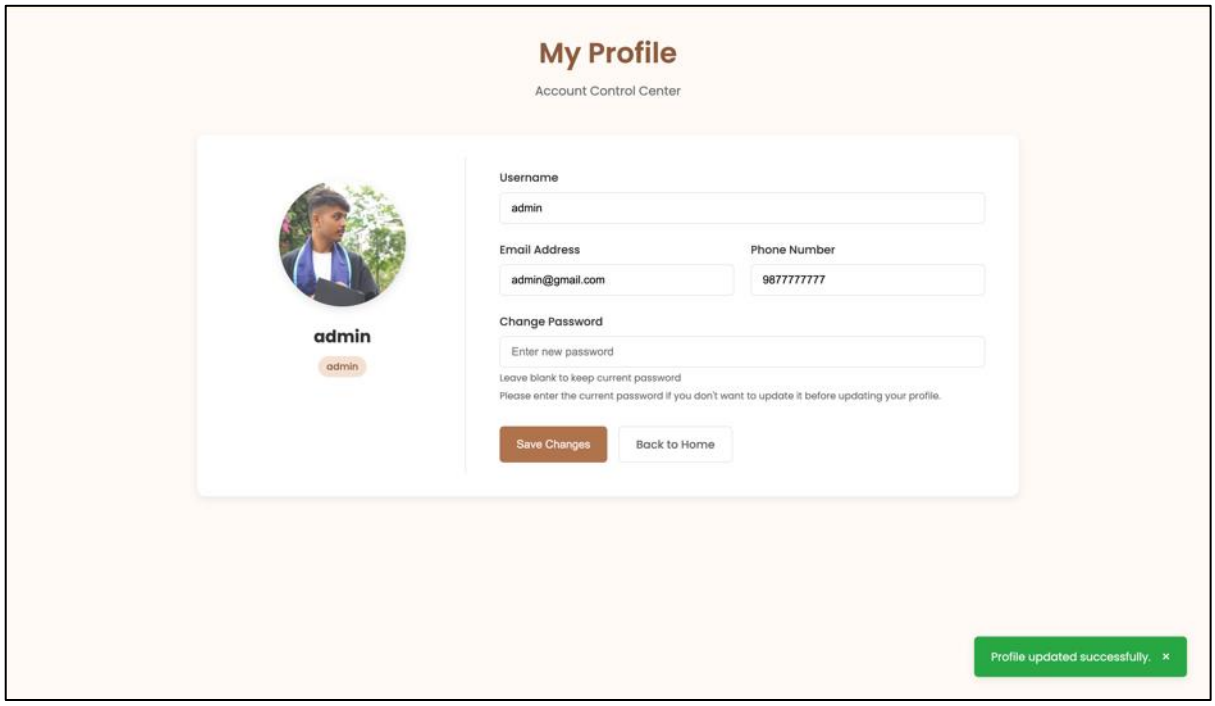


Figure 173: after updating the user credentials from the profile page

Figure 174: updated user information in the database

6.5. Admin CRUD Operations

This feature should allow the admin to perform main CRUD operations in the system and the changes done by the admin should be seen by both the admin and customers.

6.5.1. Completion of features

The feature required didn't require much effort as most features only required to run the sql queries for example to fetch the data from the database, to update the data to the database and delete data from the database. Just like other features, here also we used the GitHub repository provided to us by our tutor which helped us to create our own services and controller files to complete those features.

6.5.2. Internet Support

The following screenshots show the evidences of using internet support used to complete the CRUD operations.

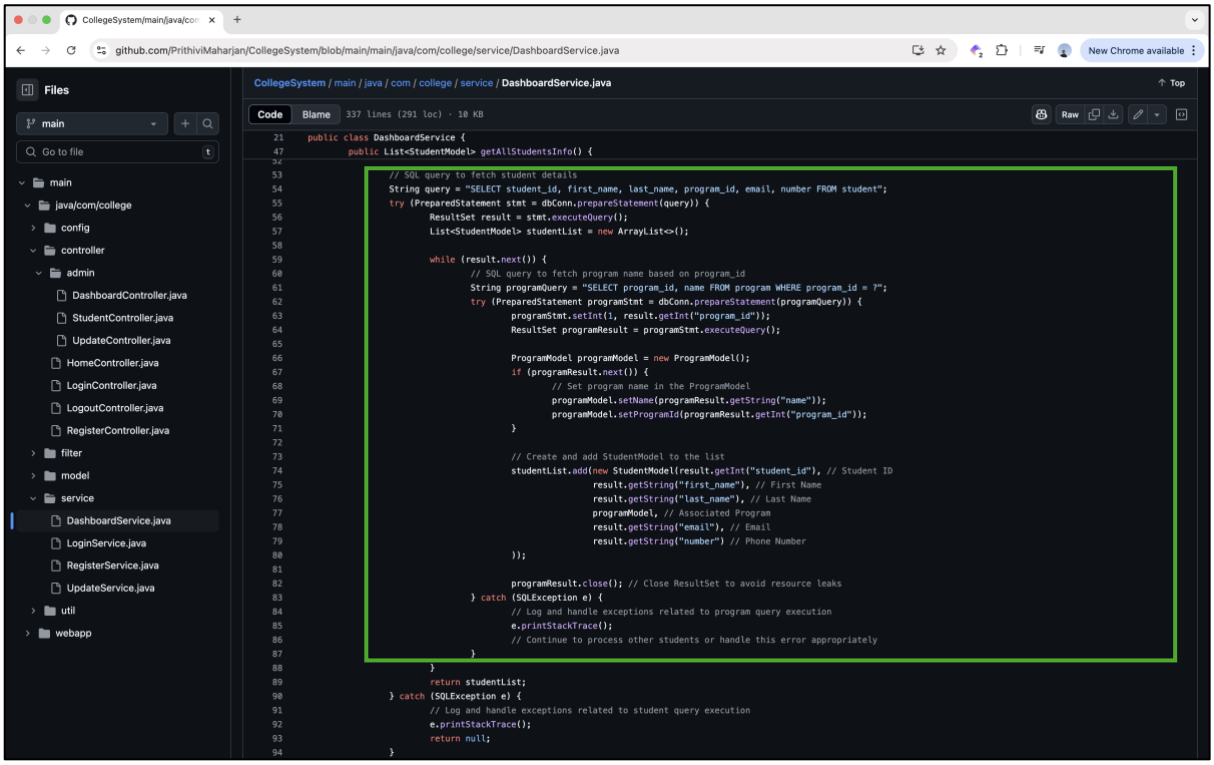


Figure 175: fetching data query

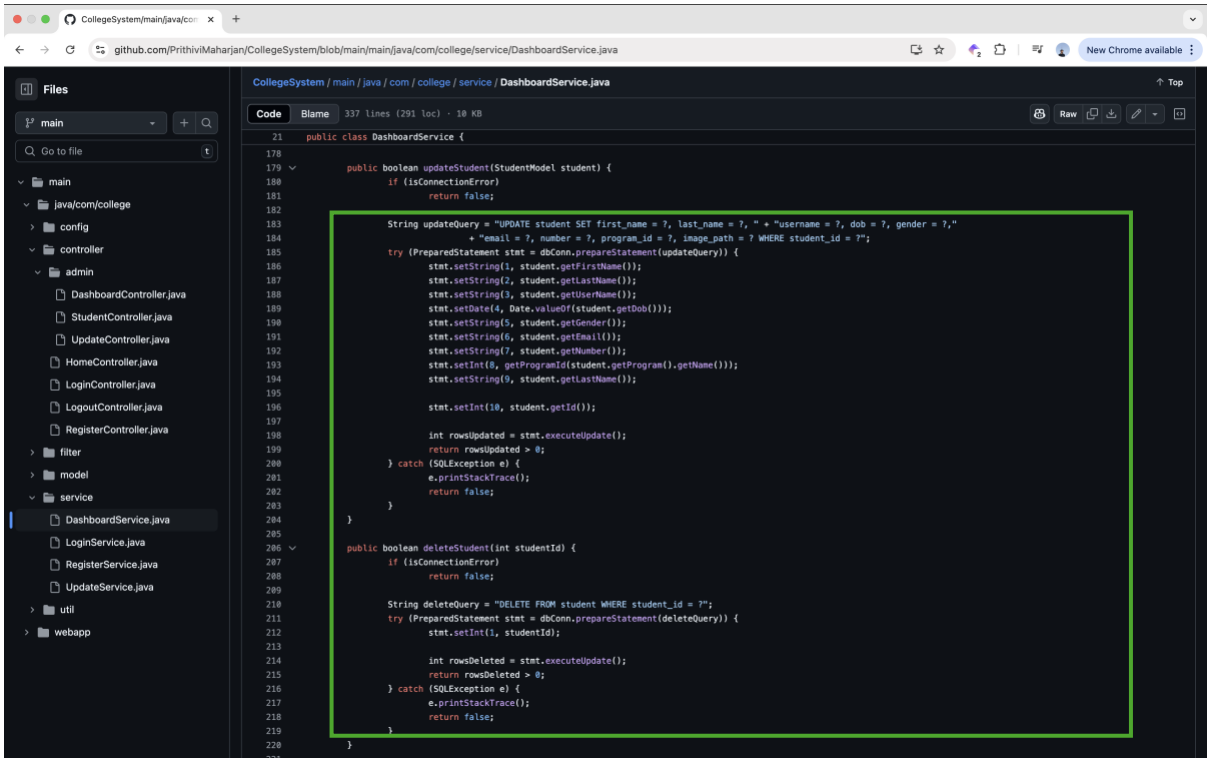


Figure 176: update query

6.5.3. Critical Analysis

6.5.3.1. Challenges Faced

The challenges faced during implementing the operations in the system were such as to reflect the changes done by the admin to the dashboard main table, admin graph and customer accessible pages. This challenge required the above-mentioned internet support and some internal logics to be solved. Following screenshots show the evidence of solving the problem.

ProductService.java

```
41
42      /**
43       * Searches for products based on the given search term.
44       * The search term is matched against product name, description, and category.
45       *
46       * @param searchTerm the search term to use for matching product attributes
47       * @return a list of products that match the search term
48       */
49      public List<ProductModel> searchProducts(String searchTerm) {
50          List<ProductModel> products = new ArrayList<>();
51          String query = "SELECT * FROM products WHERE LOWER(name) LIKE ? OR LOWER(description) LIKE ? OR LOWER(category) LIKE ?";
52
53          try (Connection conn = DbConfig.getDbConnection();
54               PreparedStatement pstmt = conn.prepareStatement(query)) {
55
56              String keyword = "%" + searchTerm.toLowerCase() + "%";
57              pstmt.setString(1, keyword);
58              pstmt.setString(2, keyword);
59              pstmt.setString(3, keyword);
60
61              try (ResultSet rs = pstmt.executeQuery()) {
62                  while (rs.next()) {
63                      ProductModel product = extractProductFromResultSet(rs);
64                      products.add(product);
65                  }
66              }
67
68          } catch (SQLException | ClassNotFoundException e) {
69              e.printStackTrace();
70          }
71
72          return products;
73      }
```

Figure 177: code to search code

ProductService.java

```
74
75      /**
76       * Adds a new product to the database.
77       *
78       * @param product the product to add to the database
79       */
80      public void addProduct(ProductModel product) {
81          String query = "INSERT INTO products (name, description, price, category, total_sales) VALUES (?, ?, ?, ?, ?)";
82
83          try (Connection conn = DbConfig.getDbConnection();
84               PreparedStatement pstmt = conn.prepareStatement(query)) {
85
86              pstmt.setString(1, product.getName());
87              pstmt.setString(2, product.getDescription());
88              pstmt.setDouble(3, product.getPrice());
89              pstmt.setString(4, product.getCategory());
90              pstmt.setInt(5, product.getTotalSales());
91              pstmt.executeUpdate();
92
93          } catch (SQLException | ClassNotFoundException e) {
94              e.printStackTrace();
95          }
96      }
97
```

Figure 178: code to add product

ProductService.java

```
97
98      /**
99       * Updates an existing product in the database.
100      *
101      * @param product the product to update
102      */
103     public void updateProduct(ProductModel product) {
104         String query = "UPDATE products SET name = ?, description = ?, price = ?, category = ?, total_sales = ? WHERE id = ?";
105
106         try (Connection conn = DbConfig.getDbConnection();
107              PreparedStatement pstmt = conn.prepareStatement(query)) {
108
109             pstmt.setString(1, product.getName());
110             pstmt.setString(2, product.getDescription());
111             pstmt.setDouble(3, product.getPrice());
112             pstmt.setString(4, product.getCategory());
113             pstmt.setInt(5, product.getTotalSales());
114             pstmt.setInt(6, product.getId());
115             pstmt.executeUpdate();
116
117         } catch (SQLException | ClassNotFoundException e) {
118             e.printStackTrace();
119         }
120     }
121
```

Figure 179: code to update product

ProductService.java

```
121
122     /**
123      * Deletes a product from the database by its ID.
124      *
125      * @param productId the ID of the product to delete
126      */
127     public void deleteProduct(int productId) {
128         String query = "DELETE FROM products WHERE id = ?";
129
130         try (Connection conn = DbConfig.getDbConnection();
131              PreparedStatement pstmt = conn.prepareStatement(query)) {
132
133             pstmt.setInt(1, productId);
134             pstmt.executeUpdate();
135
136         } catch (SQLException | ClassNotFoundException e) {
137             e.printStackTrace();
138         }
139     }
140
```

Figure 180: code to delete product

The above screenshots are the solution to the challenge we faced during the implementation of the CRUD operations in the admin side.

6.5.4. Problems faced

Here, while developing this feature we came across two errors. One of them was not checking the duplicate products and the other problem was selecting the best-selling product. In the best-selling product, the problem was if any two products have the same number of total sales it would return either one randomly there was no such criteria of measurement. Since our system doesn't have enough values to have measurement of such criteria.

6.5.4.1. Solution to the problem

To solve this problem, we first took a glace at the database. To solve the problem it required the name column of the products table to be UNIQUE identifier. First we changed the column type of the products column.

Server: localhost » Database: BakeEaseDb » Table: products

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracking | Triggers

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More
3	description	text	utf8mb4_general_ci		Yes	NULL			Change Drop More
4	price	decimal(10,2)			No	None			Change Drop More
5	category	enum('beverages', 'baked', 'cakes and pastries')	utf8mb4_general_ci		No	baked			Change Drop More
6	total_sales	int(11)			Yes	NULL			Change Drop More

Figure 181: column structure of products table in the database

Here, we can see that the name column isn't of UNIQUE identifier. So we will be changing that.

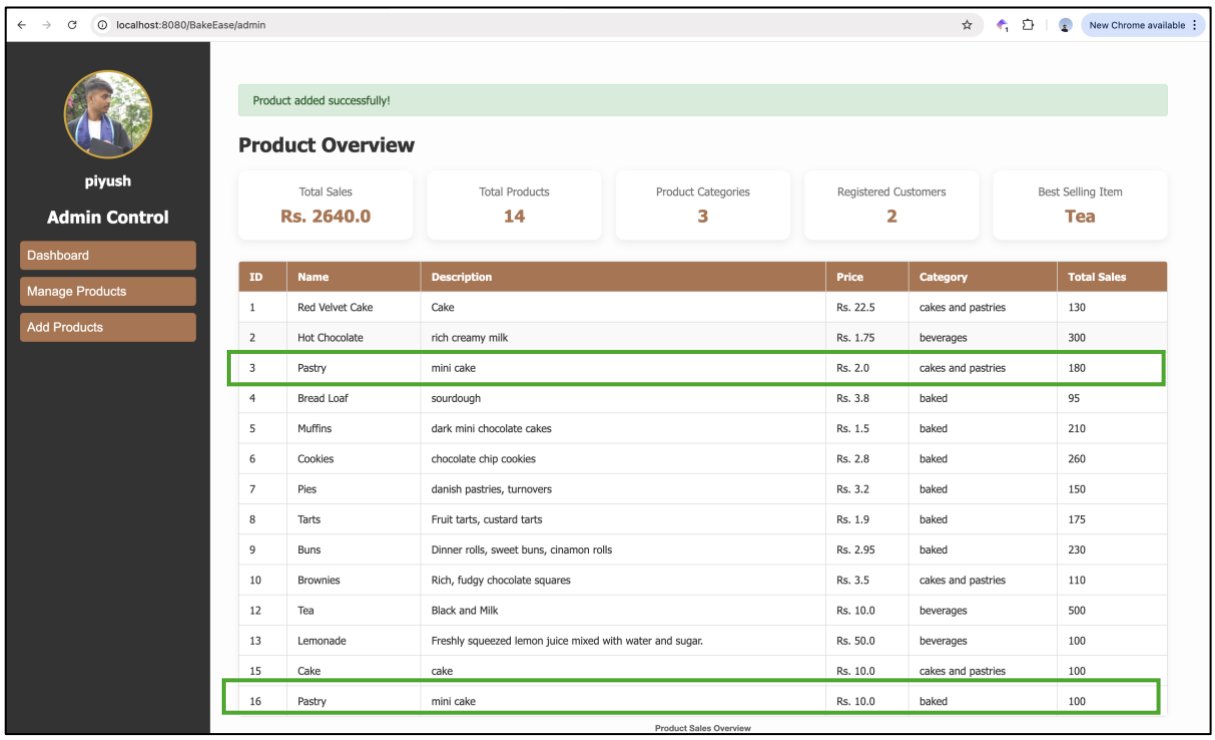


Figure 182: screenshot of the problem

Here we can see the problem evidence. Here, pastry is added two times in our products table.

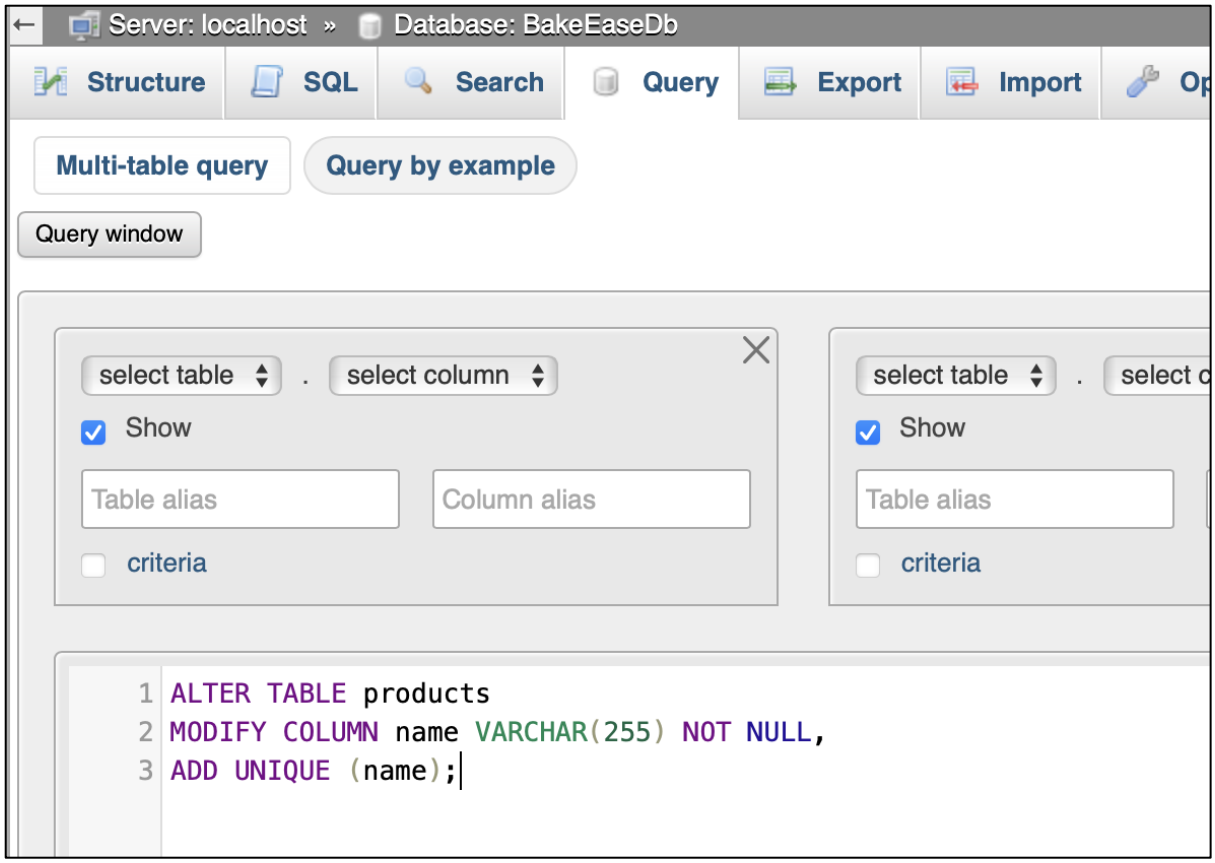


Figure 183: adding unique identifier to name column in products table

And hence after changing the column identifier to UNIQUE the problem was solved.

6.6. Admin Page Graph

The bar graph in the admin page is used to show the real time products chart in the admin page for detailed products analysis. A java script tool is used here to show the bar graph named Chart.js which is a tool that provides code snippets to show graphs in the website elements. (Chart.js, n.d.)

6.6.1. Completion of feature

The feature required two fields to be compared using bar graph. So, we chose the name of the item and its total sales to be represented in the graph.

6.6.2. Critical Analysis

6.6.2.1. Challenge

The challenge was to provide the admin with the detailed products analysis about the total sales of the products so that the stock management becomes easier for the inventory.

The challenge was overcome by using the content delivery network to import the graph from the web using script tag which is used to display graph in the admin dashboard.

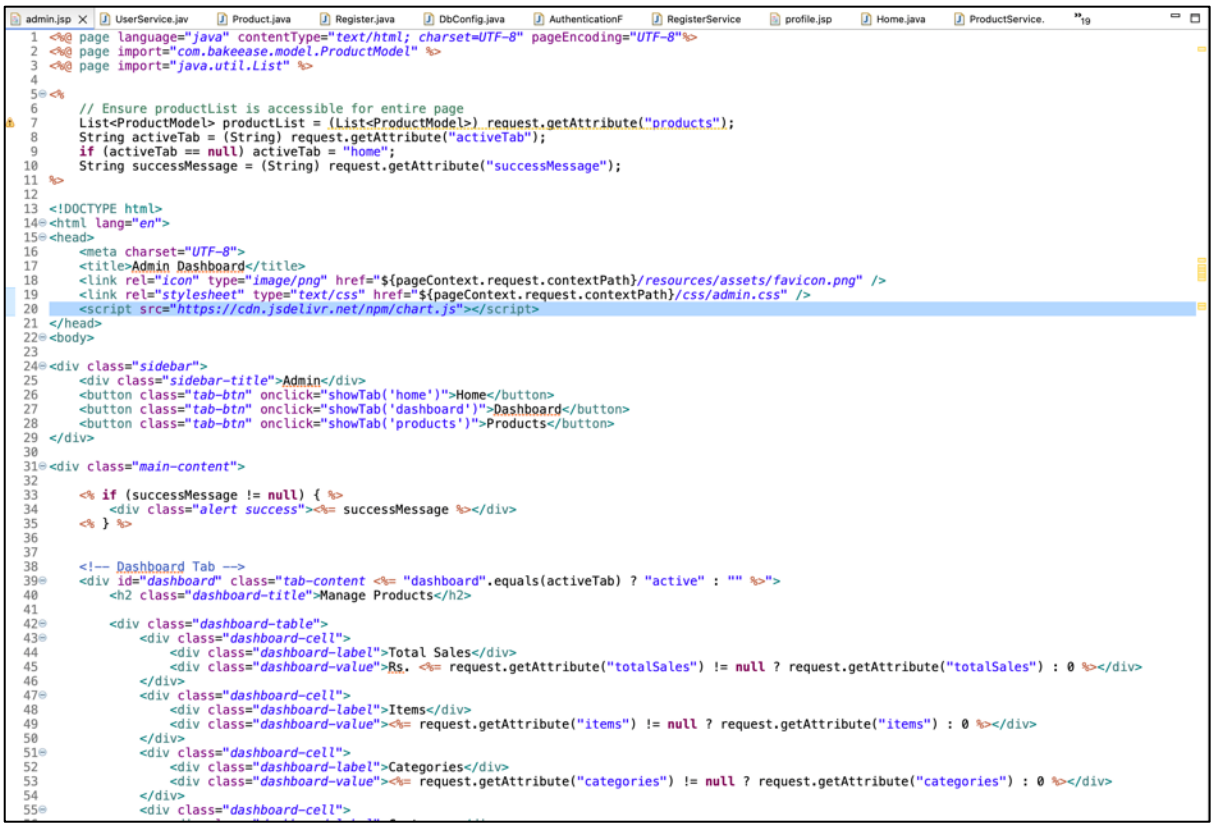


Figure 184: using chart.js in admin dashboard

A javascript function is used to initialize the graph canvas and provide necessary inputs to the graph.

```
admin.jsp | UserService.jav | Product.java | Register.java | DbConfig.java | AuthenticationF | RegisterService | profile.jsp | Home.java | ProductService.
193         }
194         document.getElementById(tabName).classList.add('active');
195     }
196
197     window.onload = function () {
198         const activeTab = "<%= activeTab %>";
199         showTab(activeTab);
200
201         const names = [<% for (ProductModel p : productList) { %>"<%= p.getName() %>",<% } %>];
202         const sales = [<% for (ProductModel p : productList) { %><%= p.getTotalSales() %>,<% } %>];
203
204         if (activeTab === 'home') {
205             renderChart('productSalesChart', names, sales);
206         } else if (activeTab === 'home') {
207             renderChart('homeProductSalesChart', names, sales);
208         }
209     };
210
211     function renderChart(canvasId, labels, data) {
212         const ctx = document.getElementById(canvasId).getContext('2d');
213         new Chart(ctx, {
214             type: 'bar',
215             data: {
216                 labels: labels,
217                 datasets: [{
218                     label: 'Total Sales',
219                     data: data,
220                     backgroundColor: 'rgba(175, 115, 78, 0.7)',
221                     borderColor: 'rgba(175, 115, 78, 1)',
222                     borderWidth: 1,
223                     borderRadius: 5
224                 }]
225             },
226             options: {
227                 responsive: true,
228                 plugins: {
229                     title: {
230                         display: true,
231                         text: 'Product Sales Overview'
232                     }
233                 },
234                 scales: {
235                     y: {
236                         beginAtZero: true
237                     }
238                 }
239             }
240         });
241     }
242 </script>
243
244 </body>
245 </html>
246
247
```

Figure 185: initializing the graph

Now the canvas used in the home tab of the admin dashboard and the output looks like this.

Figure 186: bar graph in admin dashboard

The graph allows admin to view the product sales and make decisions for the products management.

7. Conclusion

Bake Ease is a bakery management system that connects the client to the server in all bakery management processes by using web-based technology. The system makes possible the successful implementation of some core features such as user authentication, product management, and processing of orders, as well as real-time data visualization, through the utilization of Java Server Pages, MySQL, and the Eclipse IDE.

Heavy testing and validation have revealed that the system functions perfectly, allowing one to register, log in or update, or manage a profile with regard to users, products, and role-based access control. The use of JDBC, servlets, and utility classes managed this coupled with effective problem-solving for password encryption, session management, and dynamic data rendering.

This feature interacts with Chart.js to further enhance the administration dashboard by providing a real-time view of sales analytics. Bake Ease seems to be a perfect solution that exhibits scalability to any bakery making its jump onto the digital platforms. Mobile compatibility, payment gateways, and advanced reporting can be combined to increase its functionality and user experience in future versions. Being an open-source project, continuous improvements should make it apt for bakers of small and medium category seeking to enhance their baking business online.

8. Bibliography

Ahmed, H., n.d. *Simplelearn.com.* [Online]
Available at: https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql#:~:text=CourseExplore%20Program-,What%20is%20Normalization%20in%20DBMS?,%2C%20Update%2C%20and%20Deletion%20anomalies.

Tonyloi, I., n.d. *Dev Community.* [Online]
Available at: https://dev.to/isaactony/database-normalization-explained-simply-with-an-example-4lkn?utm_source=chatgpt.com

Wikipedia, n.d. *Wikimedia.* [Online]
Available at: https://en.wikipedia.org/wiki/First_normal_form

Fayard, M., n.d. *Second Normal Form.* [Online]
Available at: https://www.datacamp.com/tutorial/second-normal-form

Chris, K., n.d. *Free Code Camp.* [Online]
Available at: https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/

Bakery, M., n.d. *Magnolia Bakery.* [Online]
Available at: https://www.magnoliabakery.com/?srsltid=AfmBOoonIXAwamP3EMvf4w2AvnDPfrO8k1cqMgxSyHnUK6mTuotEOpJr

Tokari, n.d. *Tokari Login.* [Online]
Available at: https://tokari.vercel.app/login

Projector, S., n.d. [Online]
Available at: https://www.sideprojectors.com/project/7781/bakery-website-with-admin-dashboard

Jynx, n.d. *Jynx.* [Online]
Available at: https://jynxnepal.vercel.app

Tokari, n.d. *Navigation Bar.* [Online]
Available at: https://tokari.vercel.app/

Sandbox.io, n.d. *Profile Page.* [Online]
Available at: https://codesandbox.io/p/sandbox/profile-page-ooh16

Canva, n.d. *Canva Designing Tool.* [Online]
Available at: https://www.canva.com/

IDE, E., n.d. *Eclipse IDE.* [Online]
Available at: https://eclipseide.org/

Maharjan, P., 2025. *College App GitHub repository.* [Online]
Available at: https://github.com/PrithiviMaharjan/CollegeSystem
[Accessed 2025].

Anon., n.d. *Daraz.* [Online]
Available at: https://www.daraz.com.np/#hp-flash-sale

Chart.js, n.d. *Chart.js.* [Online]
Available at: https://www.chartjs.org/