



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/piyyuushh/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

52 Pages

Submission Date

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

9,015 Words

Download Date

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

46,366 Characters

File Name

23048660 Piyush Karn.docx

File Size

54.4 KB

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

1. Introduction.....	1
1.1. Overview and Purpose	1
1.2. Audience	1
1.3. Aims, Objectives and Features	1
1.3.1. Aim.....	1
1.3.2. Objectives	1
1.3.3. Features	2
2. Database Design and Development	3
2.1. Normalization.....	3
2.1.1. Unnormalized form (UNF)	3
2.1.2. First Normal Form (1NF).....	3
2.1.3. Second Normal Form (2NF)	4
2.1.4. Third Normal Form (3NF)	4
2.1.5. Final Table.....	4
2.2. Entity Relationship Diagram	6
2.3. Database Development	7
2.3.1. Table 1: Users.....	8
2.3.2. Table 2: Products.....	8
3. Wireframe and Design.....	9
3.1. Navigation Bar	9
3.1.1. Research Reference.....	10
3.1.2. Wireframe Design	11
3.1.3. Design Development.....	11
3.2. Home Page	12
3.2.1. Research Reference.....	12
3.2.2. Wireframe Design	14
3.2.3. Design Development.....	14
3.3. Products Page.....	17
3.3.1. Wireframe Design	17

3.4. Login/Register	19
3.4.1. Research Reference.....	19
3.4.2. Wireframe Design	19
3.4.3. Design Development.....	20
3.5. Admin Page.....	22
3.5.1. Research Reference.....	22
3.5.2. Wireframe Design	22
3.5.3. Design Development.....	23
3.6. Contact Page	24
3.6.1. Research Reference.....	24
3.6.2. Wireframe Design	24
3.6.3. Design development	25
3.7. About us	26
3.7.1. Research reference.....	26
3.7.2. Wireframe Design	26
3.7.3. Design Development.....	27
3.8. Profile Page	29
3.8.1. Research Reference.....	29
3.8.2. Wireframe Design	29
3.8.3. Design Development.....	30
3.9. Footer.....	31
3.9.1. Research reference.....	31
3.9.2. Wireframe Design	31
3.9.3. Design Development.....	32
4. Java Classes and Methods	33
4.1. Individual Class Diagram and Methods Description	33
4.1.1. DbConfig	33
4.1.2. About.java controller	35
4.1.3. Admin.java Controller	37
4.1.4. Contact.java controller	39
4.1.5. Home.java controller	41
4.1.6. Login.java controller	42
4.1.7. Product.java controller	43
4.1.8. Profile.java controller.....	46
4.1.9. Register.java controller	48

4.1.10.	AuthenticationFilter.java Filter	51
4.1.11.	ProductModel.java Model	53
4.1.12.	UserModel.java Model	55
4.1.13.	LoginService.java Service	56
4.1.14.	ProductService.java Service.....	58
4.1.15.	RegisterService.java Service	62
4.1.16.	UserService.java Service.....	63
4.1.17.	CookieUtil.java Util	66
4.1.18.	ImageUtil.java Util	67
4.1.19.	PasswordUtil.java Util	70
4.1.20.	SessionUtil.java Util.....	72
4.1.21.	UpdateProfile.java Util	74
4.1.22.	ValidationUtil.java Util	75
5.	Test Cases	77
5.1.	Planning Test Cases	77
5.2.	Test 1: User register.....	78
5.3.	Test 2: User Login	82
5.4.	Test 3: Profile Update	86
5.5.	Test 4: Add Product	88
5.6.	Test 5: Updating and Deleting Product	93
5.7.	Test:6 Search Products	99
5.8.	Test 7: Cookie Testing.....	101
5.9.	Test 8: Filter Testing.....	103
5.10.	Test 9: Profile Data Fetch.....	107
5.11.	Test 10: Profile Update	109
6.	Coursework Development and Analysis	112
6.1.	UI/UX Design and Development	112
6.2.	Register Feature	112
6.2.1.	Completion of the feature	112
6.2.2.	Internet Support.....	112
6.2.3.	Critical Analysis	113

6.3. Login Feature	116
6.3.1. Completion of the feature	116
6.3.2. Internet Support.....	116
6.3.3. Critical Analysis	117
6.4. Profile Feature	121
6.4.1. Completion of the feature	121
6.4.2. Internet Support.....	121
6.4.3. Critical Analysis	122
6.5. Admin Page Graph.....	130
6.5.1. Completion of feature	130
6.5.2. Critical Analysis	130
7. Conclusion	133
8. Bibliography.....	134

Table of Figures

Figure 1: Entity Relationship diagram	6
Figure 2: User table structure.....	8
Figure 3: data in users table.....	8
Figure 4: table structure of products table	9
Figure 5: values inside products table	9
Figure 6: Reference of Nav bar.....	10
Figure 7: Designing the navbar wireframe in Canva	11
Figure 8: Design Development of navigation bar	11
Figure 9: Research reference for Home page (i).....	12
Figure 10: research reference for home page (ii).....	13
Figure 11: wireframe of Home page (i)	14
Figure 12: wireframe of home page (ii).....	14
Figure 13: Home page design (i)	15
Figure 14: home page design (ii)	16
Figure 15: reference for product page	17
Figure 16: Actual product page design	18
Figure 17: research reference for login/register	19
Figure 18: wireframe for login/register	20
Figure 19: login page.....	20
Figure 20: register page	21
Figure 21: research reference of admin page	22
Figure 22: wireframe of admin page	23
Figure 23: admin dashboard	23
Figure 24: research reference of contact page	24
Figure 25: Wireframe of contact page	25
Figure 26: design of contact page.....	25
Figure 27: research reference of about page	26
Figure 28: wireframe design of about us page	27
Figure 29: design of about page (i).....	27
Figure 30: design of about page (ii)	28
Figure 31: research reference for profile page	29
Figure 32: wireframe of profile page	30

Figure 33: actual design of profile page.....	30
Figure 34: Research reference for footer.....	31
Figure 35: Wireframe design of footer	31
Figure 36: design of footer	32
Figure 37: overall class diagram	33
Figure 38: class diagram of DbConfig	34
Figure 39: code for DbConfig file	34
Figure 40: method description for about controller.....	35
Figure 41: code for doGet and doPost method	36
Figure 42: class diagram of about controller	37
Figure 43: code for do get method.....	38
Figure 44: code for doPost method.....	39
Figure 45: class diagram of contact controller.....	39
Figure 46: code for doGet and doPost method	40
Figure 47: class diagram of home controller	41
Figure 48: code for doGet and doPost.....	41
Figure 49: class diagram of login controller.....	42
Figure 50: code for doGet and doPost.....	43
Figure 51: class diagram of product controller	43
Figure 52: code for doGet method	44
Figure 53: code for doPost method.....	45
Figure 54: class diagram of profile controller.....	46
Figure 55: code for doGet method	46
Figure 56: code for doPost method.....	47
Figure 57: class diagram of register controller	48
Figure 58: code for doGet method	48
Figure 59: code for doPost method.....	50
Figure 60: class diagram of authentication filter.....	51
Figure 61: doFilter code for authentication filter	52
Figure 62: class diagram of product model	53
Figure 63: code of product model	54
Figure 64: class diagram for user model	55
Figure 65: code for user model	56
Figure 66: class diagram of login service	56

Figure 67: code for login service	57
Figure 68: class diagram for product service.....	58
Figure 69: code for getAllProduct method	59
Figure 70: code for search products method.....	59
Figure 71: code for add products	60
Figure 72: code for update product.....	60
Figure 73: code to delete product	61
Figure 74: method to identify the best selling product.....	61
Figure 75: class diagram of register service.....	62
Figure 76: code for add user	62
Figure 77: code for checking duplicate entries.....	63
Figure 78: class diagram of user service	63
Figure 79: code for get user by username.....	64
Figure 80: code for update user.....	65
Figure 81: code to calculate the number of customers	65
Figure 82: class diagram for cookie util	66
Figure 83: code for add cookie.....	66
Figure 84: code for get cookie.....	66
Figure 85: code for delete cookie.....	67
Figure 86: class diagram for image util.....	67
Figure 87: code for get image name	68
Figure 88: code for upload image	69
Figure 89: code for get save path	69
Figure 90: class diagram for password util	70
Figure 91: code for get random nonce and get AES key from password	71
Figure 92: code for encrypt password	71
Figure 93: code for decrypt password	72
Figure 94: class diagram for session util	72
Figure 95: code for set attribute method.....	73
Figure 96: code for get attribute	73
Figure 97: code for remove attribute.....	73
Figure 98: code for invalidate session	73
Figure 99: class diagram of update profile	74
Figure 100: code for update profile	74

Figure 101: class diagram of validation util	75
Figure 102: code for null or empty	75
Figure 103: code for isAlphabetic	75
Figure 104: code for isAlphanumericsString.....	76
Figure 105: code for isValidEmail	76
Figure 106: code for isValidPhoneNumber.....	76
Figure 107: code for valid image.....	76
Figure 108: code for password matching.....	76
Figure 109: empty fields validation in register page	79
Figure 110: incorrect details validation in register page	79
Figure 111: entering correct details in the register page	80
Figure 112: registered user in the database	80
Figure 113: before entering wrong details in the login page	82
Figure 114: after entering wrong details in the login page	83
Figure 115: before entering correct details in the web page	83
Figure 116: after entering the correct details in the web page	84
Figure 117: image of the logged in user in the system.....	85
Figure 118: user profile of the currently logged in user	85
Figure 119: before updating user profile page.....	86
Figure 120: database before updating user.....	87
Figure 121: After updating user profile.....	87
Figure 122: Database after updating table	87
Figure 123: updating the user role from the database	88
Figure 124: changed role information in profile page.....	89
Figure 125: Before adding product	89
Figure 126: adding product	90
Figure 127: product added	90
Figure 128: displaying the added product in the dashboard table	91
Figure 129: displaying the added product in the dashboard graph	91
Figure 130: displaying the added product in the product page	92
Figure 131: target product to update and delete	94
Figure 132: updating target product.....	94
Figure 133: after updating target product	95
Figure 134: updated result in dashboard table.....	95

Figure 135: update result in products page	96
Figure 136: update result in database	96
Figure 137: before deleting the target product	97
Figure 138: After deleting target product	97
Figure 139: results in database after deleting product	98
Figure 140: product not found	99
Figure 141: product found	100
Figure 142: cookie testing before logging in.....	102
Figure 143: testing cookie after logging in.....	102
Figure 144: logging in as a customer.....	104
Figure 145: trying to access admin dashboard as a customer	104
Figure 146: login redirection.....	105
Figure 147: logging in as admin	105
Figure 148: accessing admin dashboard as admin.....	106
Figure 149: admin page redirection	106
Figure 150: target user for the profile page	107
Figure 151: logging in with the target user	108
Figure 152: target user for profile update	109
Figure 153: Current user credentials of the target user	110
Figure 154: updating credentials of the target user.....	110
Figure 155: updated user credentials of the target user	111
Figure 156: after updating user credentials of the target user	111
Figure 157: register feature support.....	113
Figure 158: problem faced (before)	114
Figure 159: problem faced (after)	114
Figure 160: solution to register problem	115
Figure 161: database condition after solving the issue	116
Figure 162: login feature solution.....	117
Figure 163: Evidence of the login correct data type validation error (1)	118
Figure 164: evidence of correct data in login error (2)	118
Figure 165: cause of the error in login page.....	119
Figure 166: fixing the login error from GitHub	120
Figure 167: solution of the login error	120
Figure 168: fetching user data to profile page	121

Figure 169: internet support for update user profile	122
Figure 170: update method to update user credentials in the profile page	123
Figure 171: before updating user profile in profile page	123
Figure 172: before updating user credentials in database.....	124
Figure 173: after updating the user credentials from the profile page	124
Figure 174: updated user information in the database.....	124
Figure 175: fetching data query	125
Figure 176: update query	126
Figure 177: code to search code	127
Figure 178: code to add product.....	127
Figure 179: code to update product.....	127
Figure 180: code to delete product.....	128
Figure 181: column structure of products table in the database.....	128
Figure 182: screenshot of the problem	129
Figure 183: adding unique identifier to name column in products table	130
Figure 184: using chart.js in admin dashboard	131
Figure 185: initializing the graph	132
Figure 186: bar graph in admin dashboard	132

Table of Tables

Table 1: Test cases planning	77
Table 2: Test 1: User register	78
Table 3: Test 2: user login.....	82
Table 4: Test 3: Profile update.....	86
Table 5: Test 4: adding product.....	88
Table 6: Test 5: Update and Delete product	93
Table 7: Test 6: Search Products	99
Table 8: Cookie testing	101
Table 9: Filter testing	103
Table 10: Profile data fetch test.....	107
Table 11: Profile update test.....	109

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

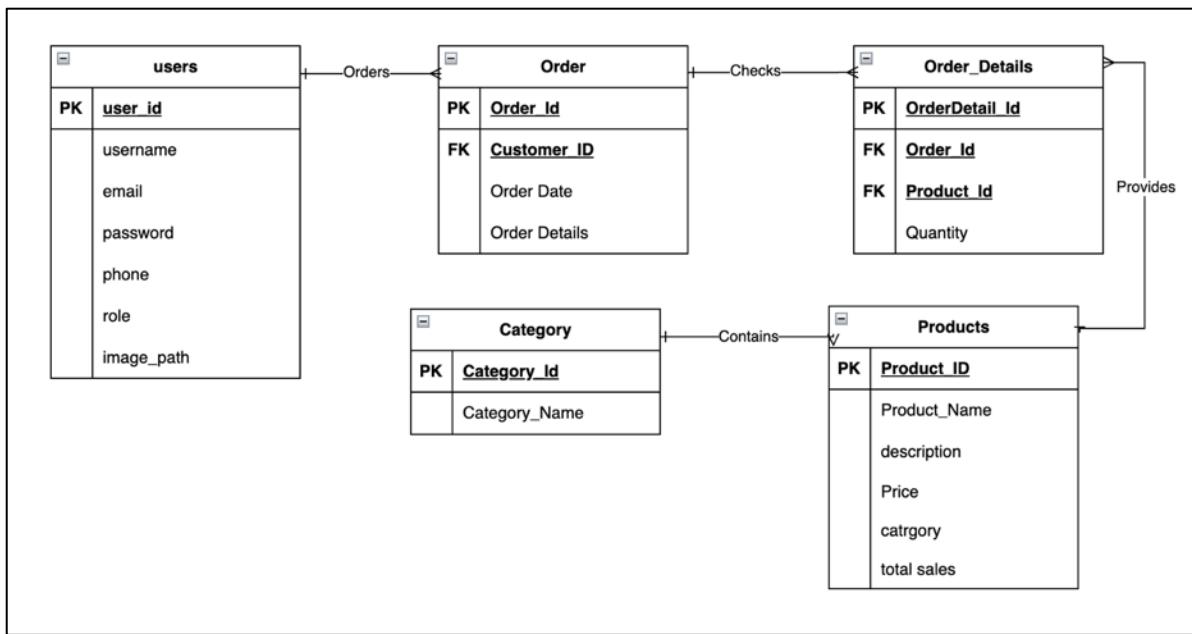


Figure 1: Entity Relationship diagram

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

The screenshot shows the 'Structure' tab for the 'users' table in the 'BakeEaseDb' database. The table has 7 columns:

#	Name	Type	Collation	Attributes	Default	Comments	Extra	Action
1	<code>id</code>	<code>int(11)</code>	<code>utf8mb4_general_ci</code>	No	None	AUTO_INCREMENT		Change Drop More
2	<code>username</code>	<code>varchar(50)</code>	<code>utf8mb4_general_ci</code>	No	None			Change Drop More
3	<code>email</code>	<code>varchar(100)</code>	<code>utf8mb4_general_ci</code>	No	None			Change Drop More
4	<code>password</code>	<code>varchar(255)</code>	<code>utf8mb4_general_ci</code>	No	None			Change Drop More
5	<code>phone</code>	<code>varchar(15)</code>	<code>utf8mb4_general_ci</code>	Yes	<code>NULL</code>			Change Drop More
6	<code>role</code>	<code>enum('customer', 'admin')</code>	<code>utf8mb4_general_ci</code>	Yes	<code>'customer'</code>			Change Drop More
7	<code>image_path</code>	<code>varchar(255)</code>	<code>utf8mb4_general_ci</code>	Yes	<code>NULL</code>			Change Drop More

Below the table structure, there are sections for 'Indexes' and 'Partitions'. The 'Indexes' section shows two indexes: 'PRIMARY' (BTREE, id) and 'email' (BTREE, email). The 'Partitions' section indicates 'No partitioning defined'.

Figure 2: User table structure

The screenshot shows the 'Browse' tab for the 'users' table. It displays two rows of data:

	<code>id</code>	<code>username</code>	<code>email</code>	<code>password</code>	<code>phone</code>	<code>role</code>	<code>image_path</code>
1	1	test1	test@gmail.com	V50Vke3eKvp1ydZvkNlQvgLtlUoIB05MdIEo+Bw3hQARqRe...	9801001000	admin	me.jpg
2	2	admin	admin@gmail.com	NRT2i2CYjzJr5+hekfICXkm2u+S8TIKOS85SmtpJYttjpuB...	9833333333	customer	me.jpg

Below the table, there are buttons for 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'.

Figure 3: data in users table

2.3.2.Table 2: Products

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

Figure 4: table structure of products table

		<code>id</code>	<code>name</code>	<code>description</code>	<code>price</code>	<code>category</code>	<code>total_sales</code>
	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, cinnamon 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
	Edit Copy Delete	15	Cake	cake	10.00	cakes and pastries	100
	Edit Copy Delete	16	Pastry	mini cake	10.00	baked	100

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.)

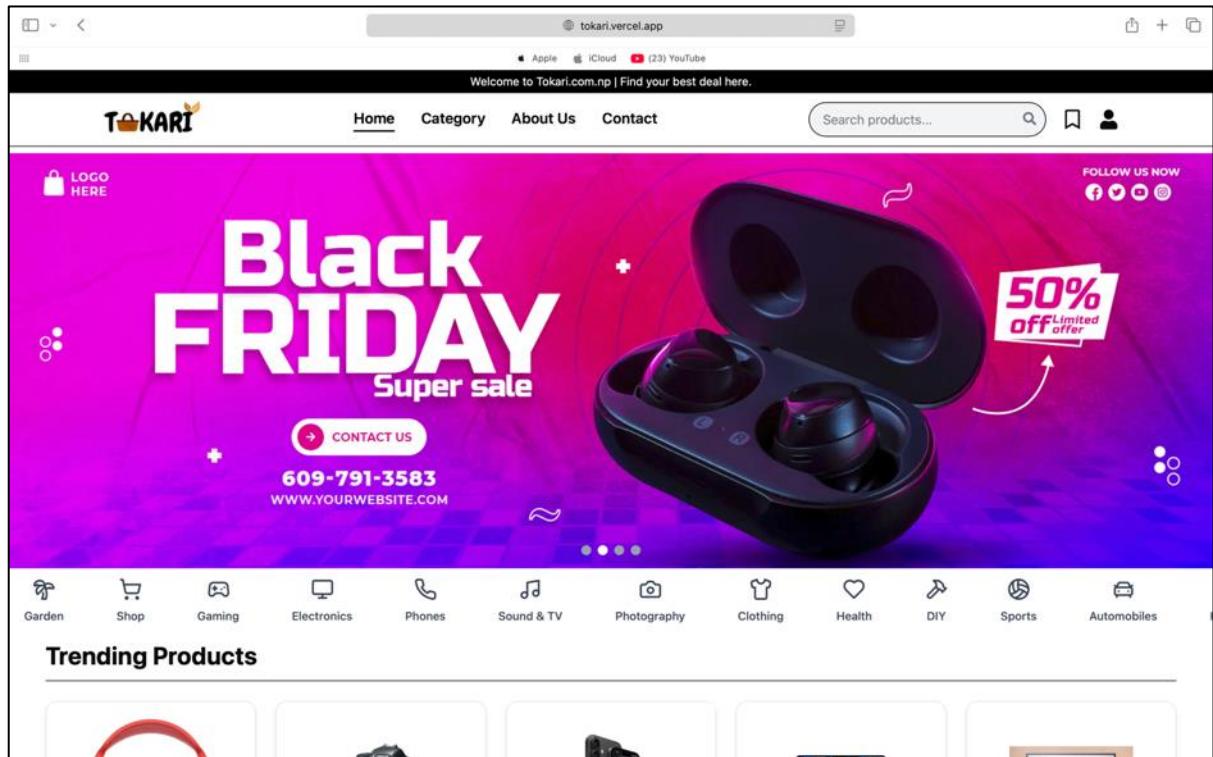


Figure 6: Reference of Nav bar

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

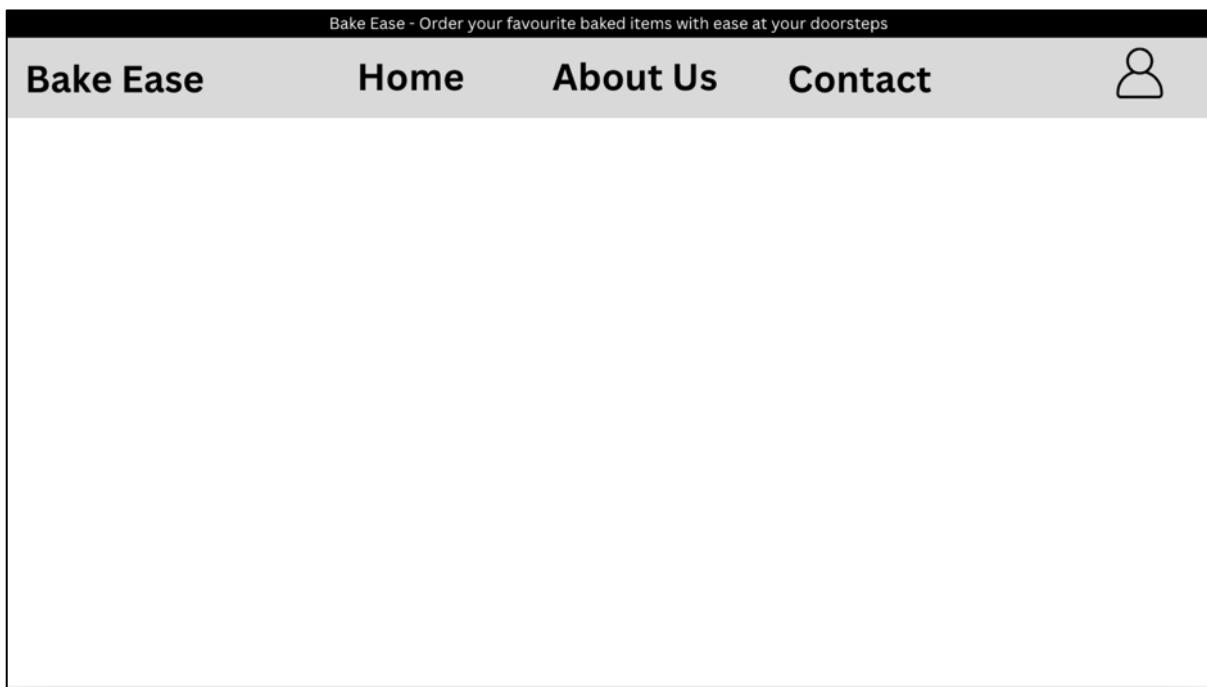


Figure 7: Designing the navbar wireframe in Canva

3.1.3. Design Development

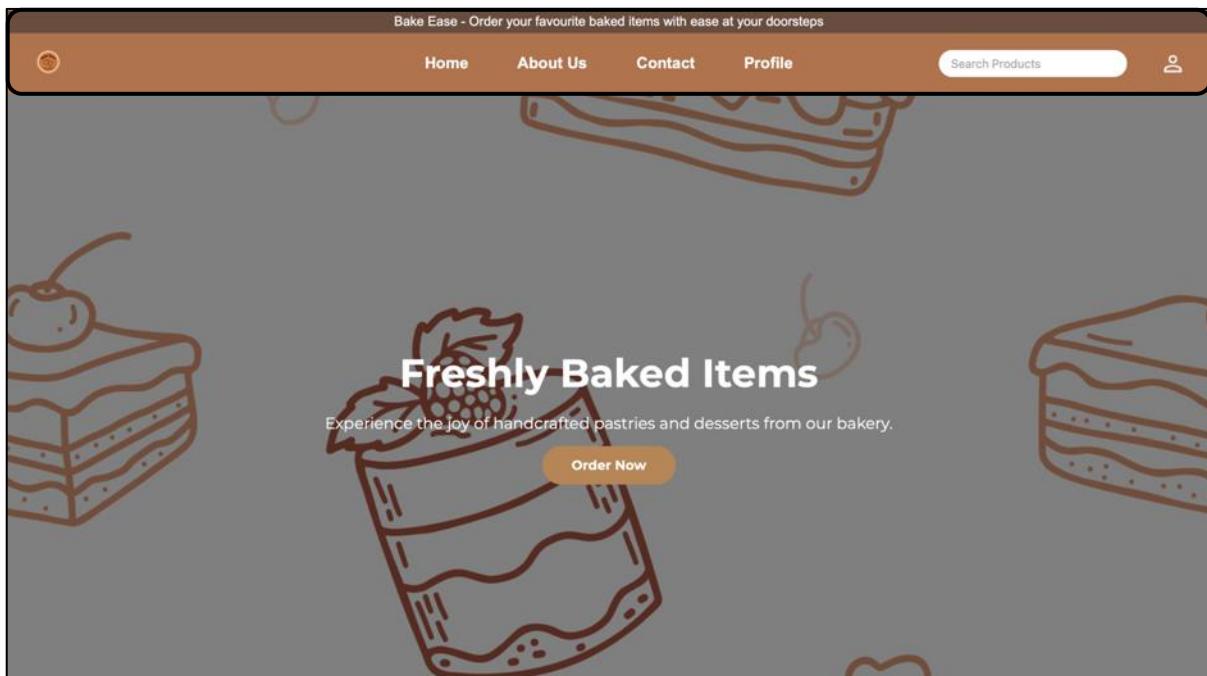


Figure 8: Design Development of navigation bar

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

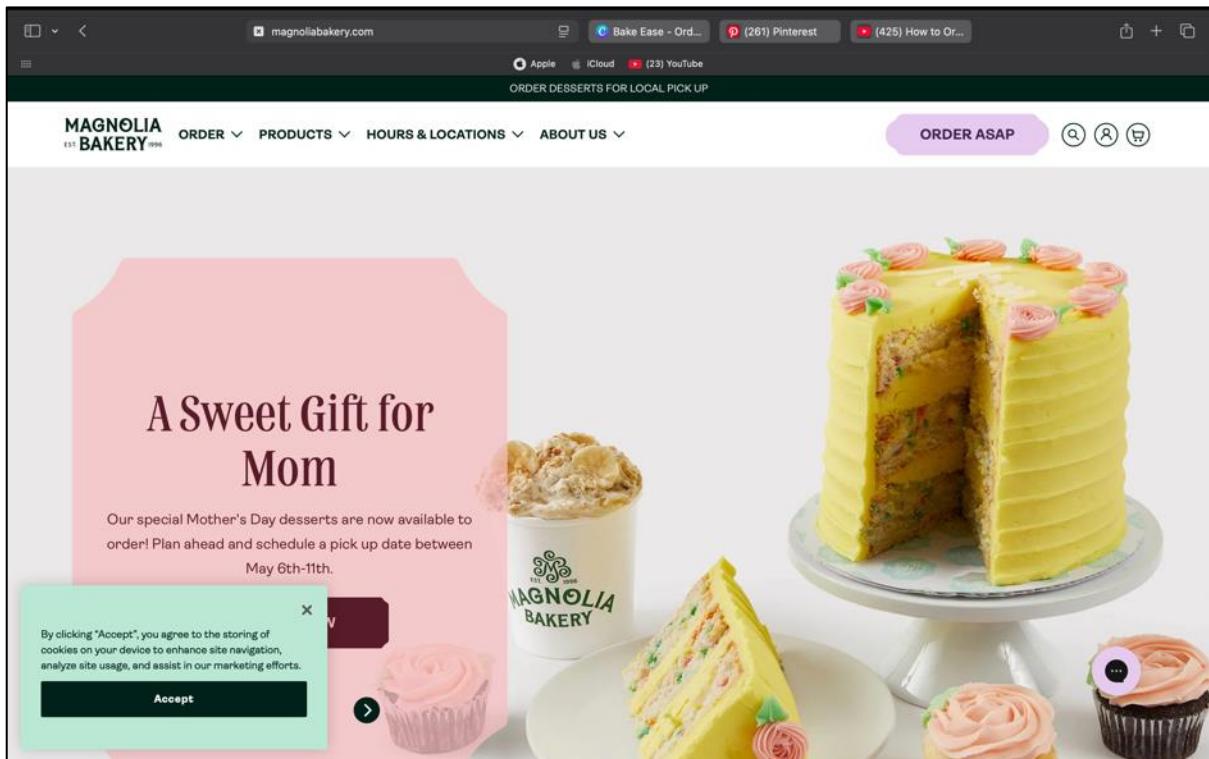


Figure 9: Research reference for Home page (i)

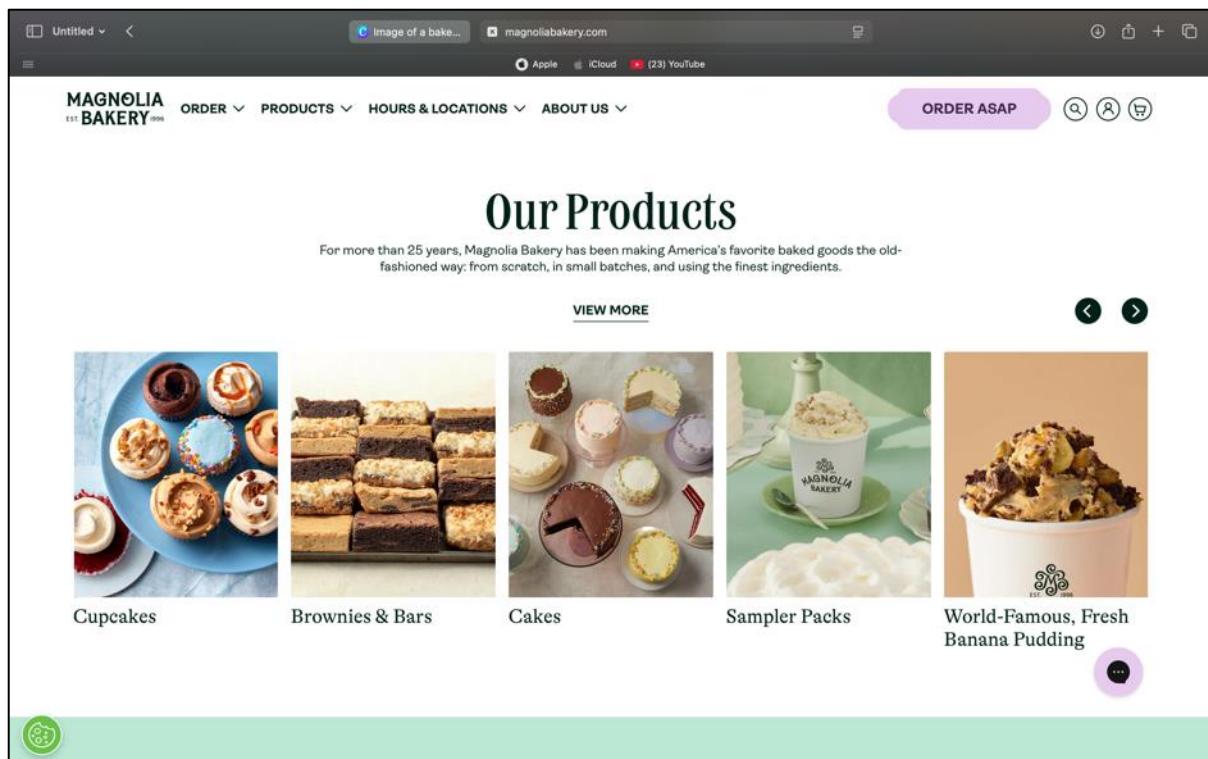


Figure 10: research reference for home page (ii)

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)

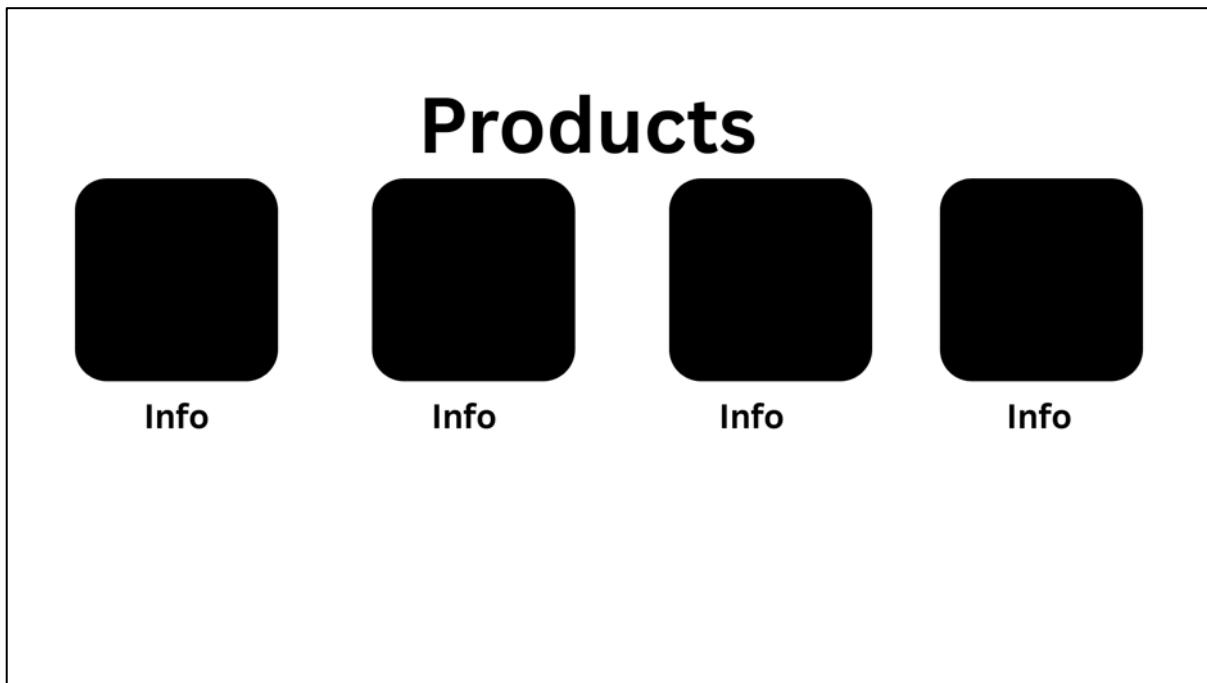


Figure 12: wireframe of home page (ii)

3.2.3. Design Development

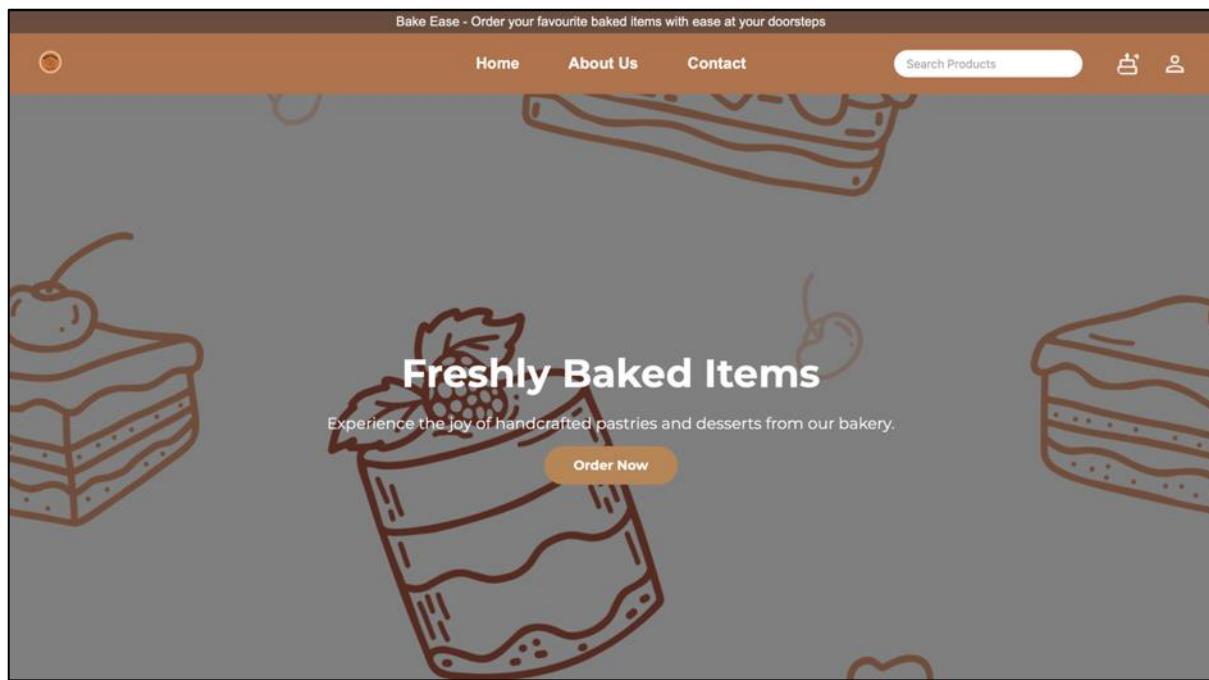


Figure 13: Home page design (i)

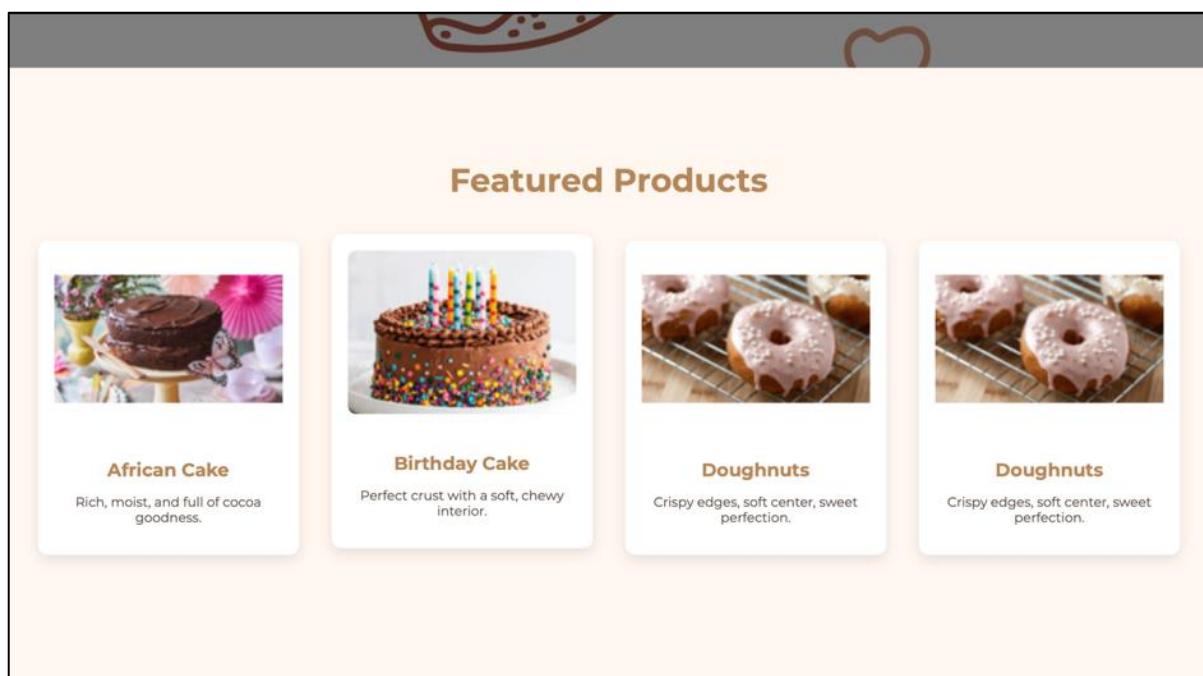


Figure 14: home page design (ii)

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.)

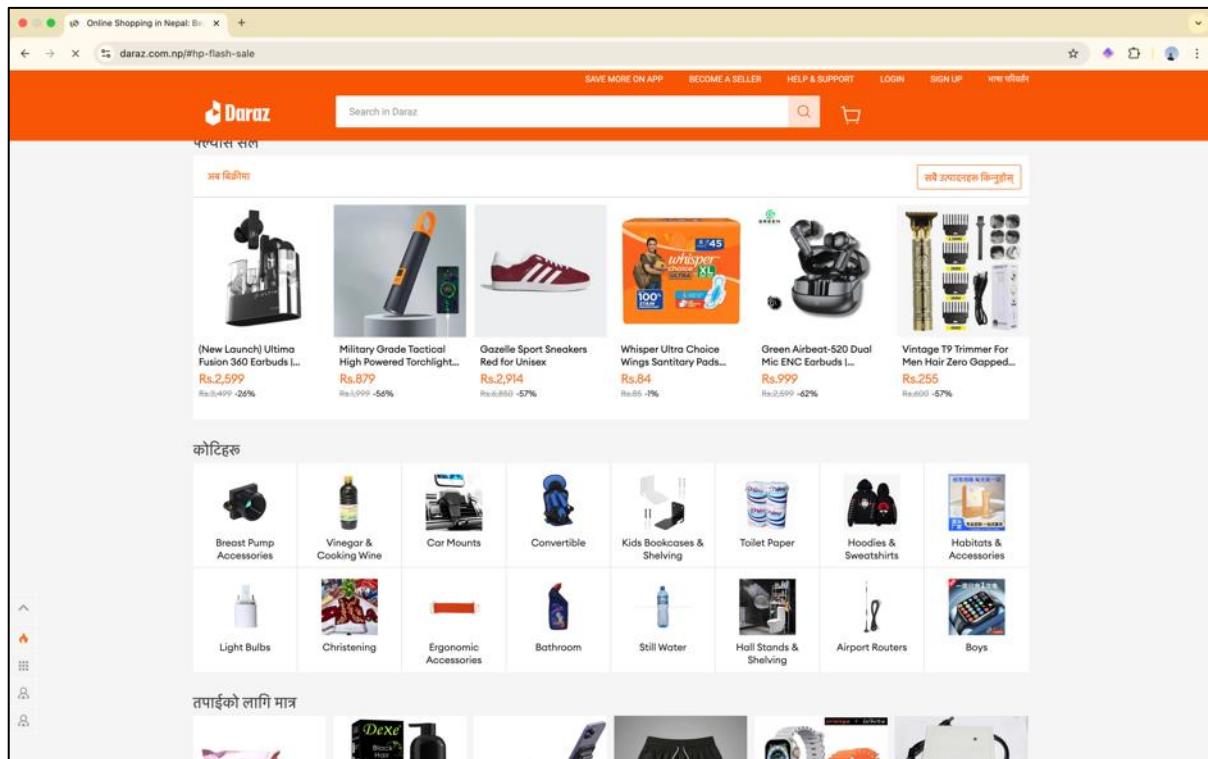


Figure 15: reference for product page

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

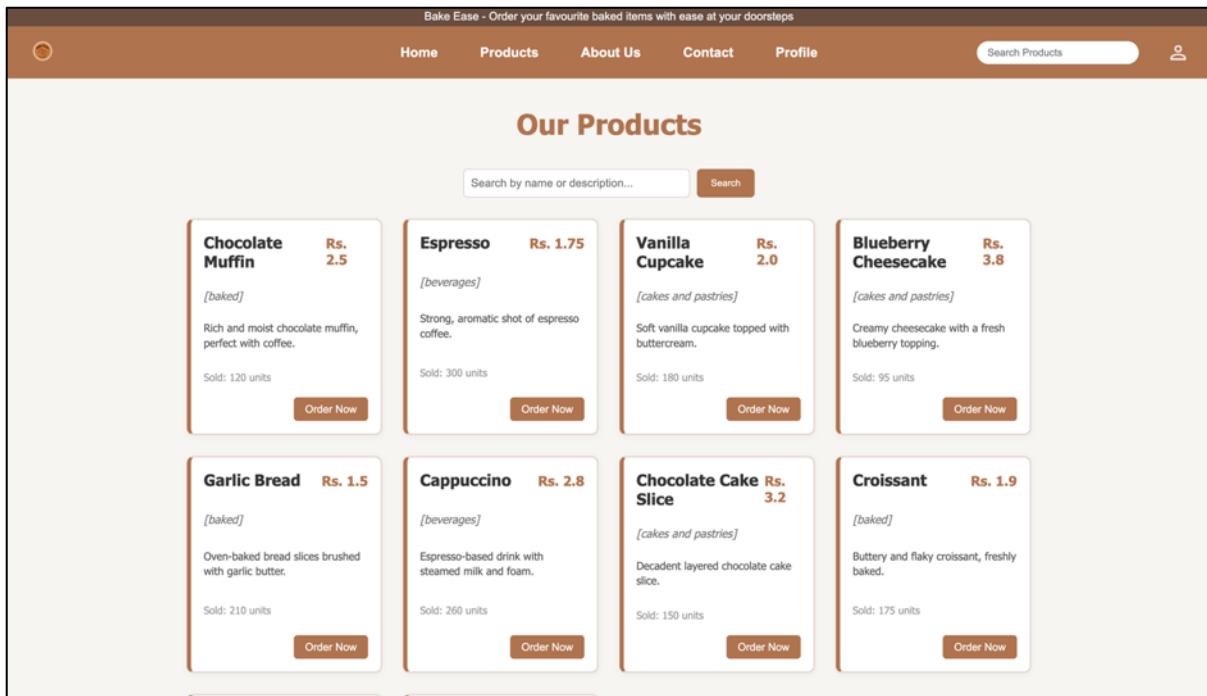


Figure 16: Actual product page design

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

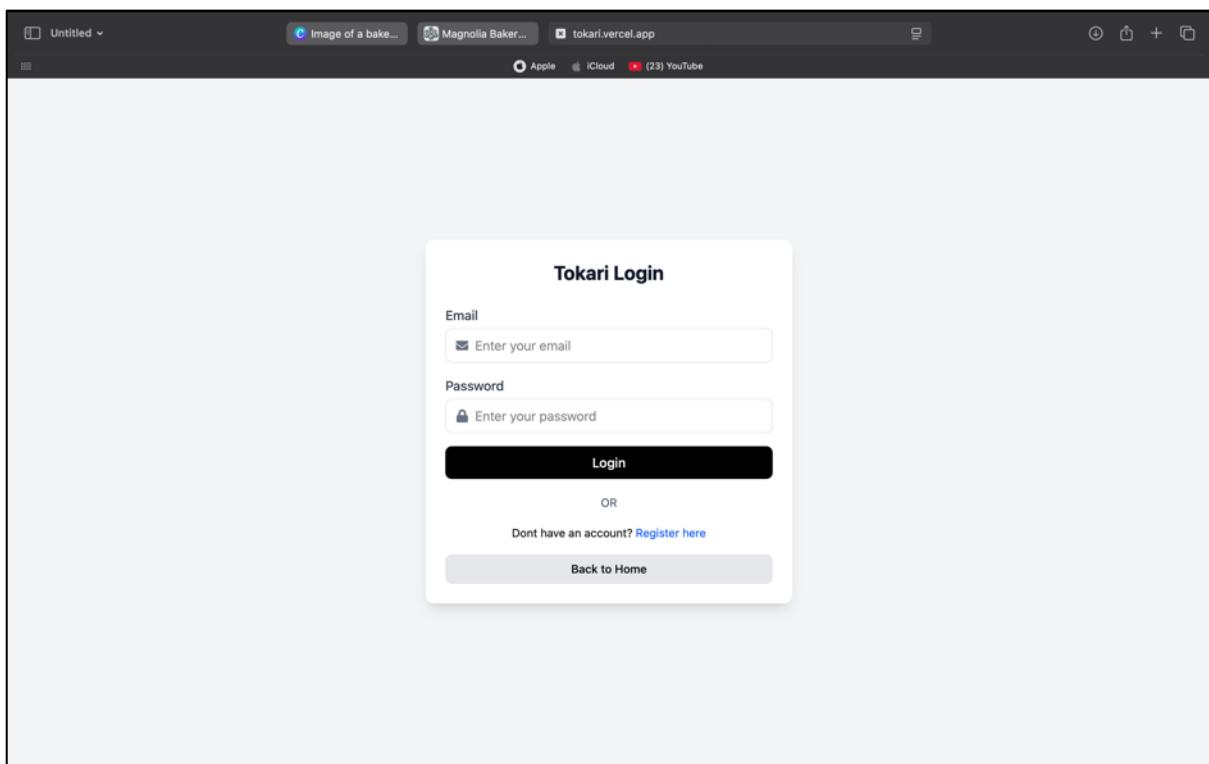


Figure 17: research reference for login/register

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

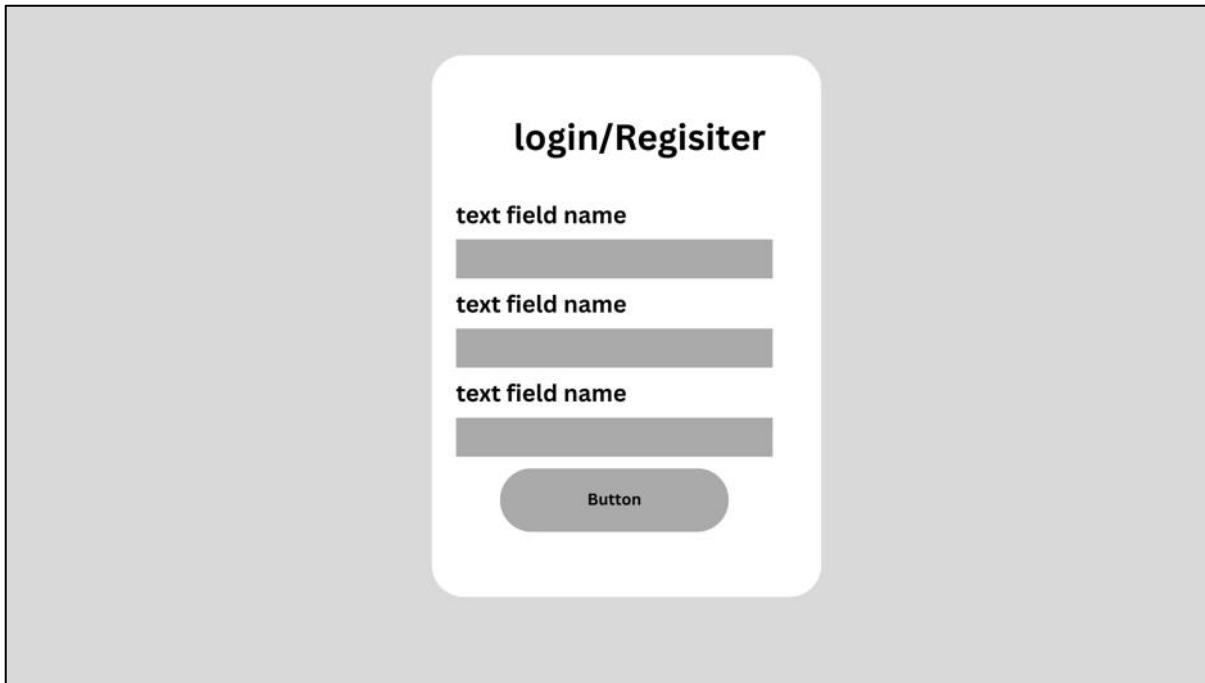


Figure 18: wireframe for login/register

3.4.3. Design Development

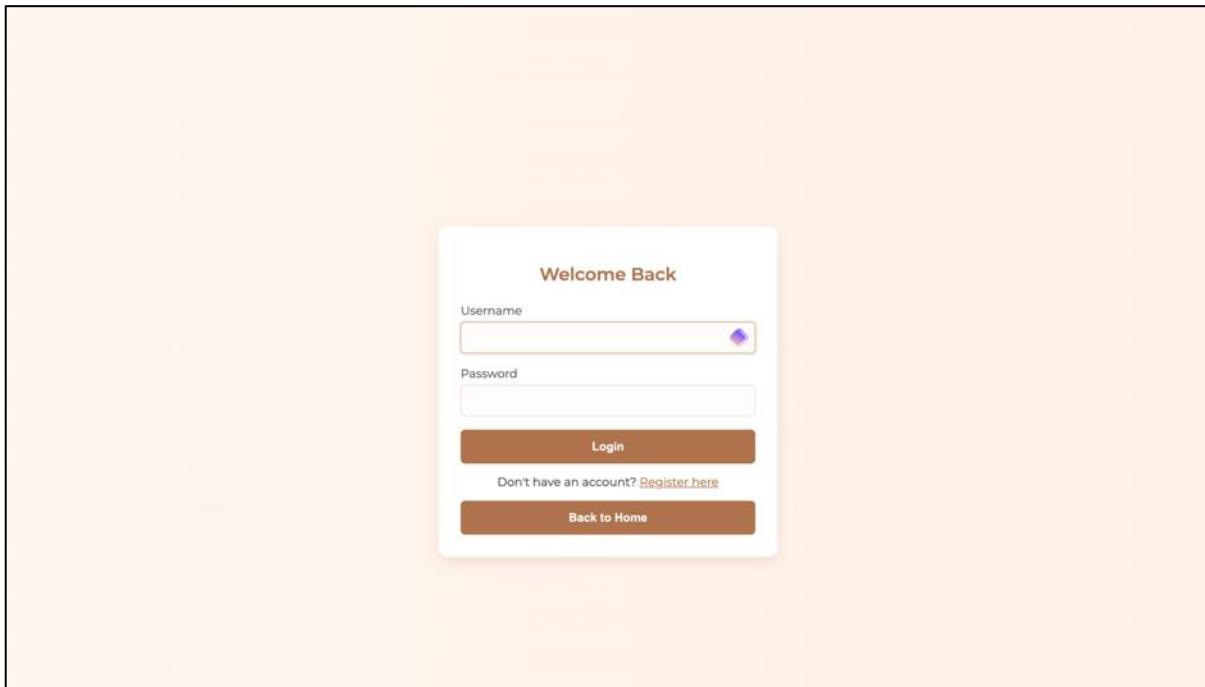


Figure 19: login page

The screenshot shows a registration form titled "Create Account". The form consists of several input fields: "Username", "Email", "Password", "Confirm Password", and "Phone Number". Below these is a "Profile Image" field with a file selection button ("Choose File") showing "no file selected". At the bottom of the form are two buttons: "Register" and "Back to Home". A link "Already have an account? [Login here](#)" is also present.

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.)

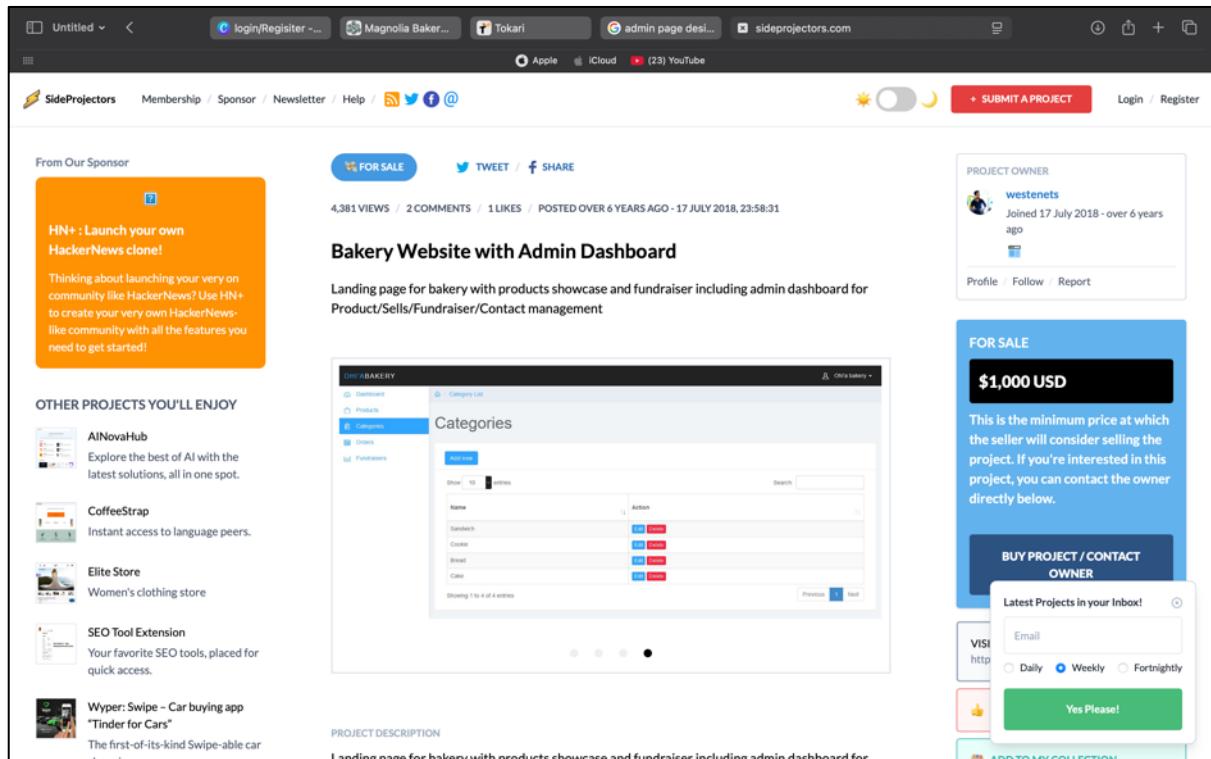


Figure 21: research reference of admin page

The reasons why I chose this as my reference are:

- Design Layout
- Options layout in the dashboard

3.5.2. Wireframe Design

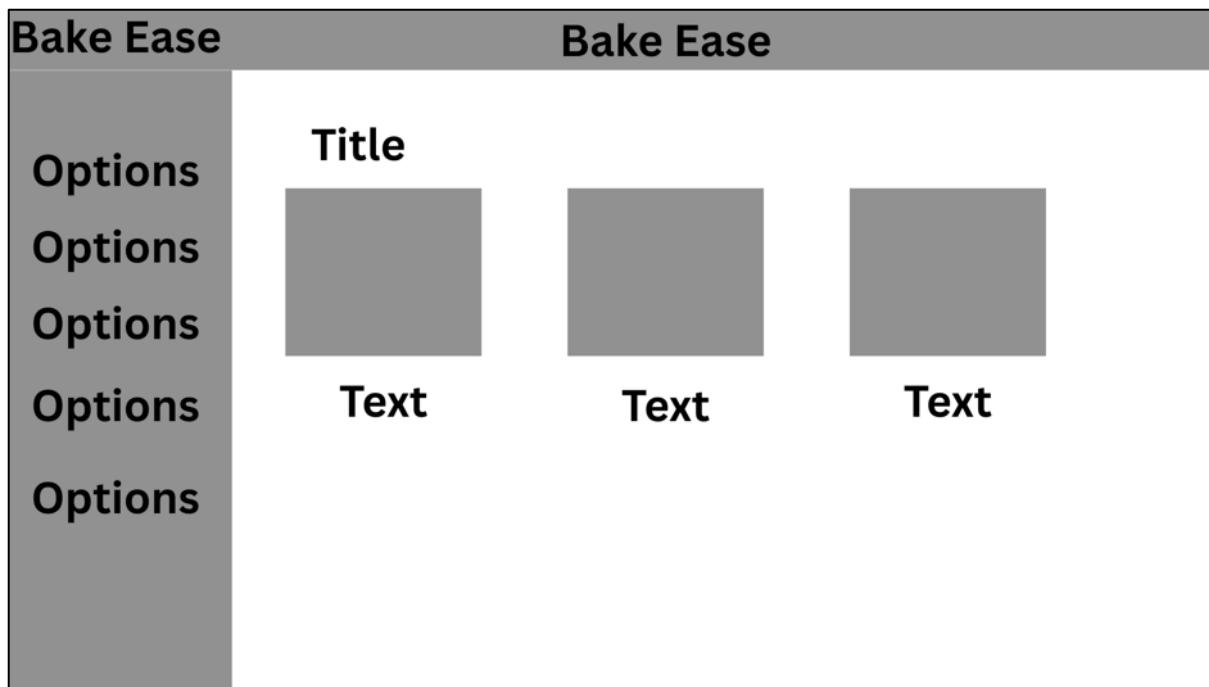


Figure 22: wireframe of admin page

3.5.3. Design Development

The screenshot shows a web browser window titled "Admin Dashboard" with the URL "localhost:8080/BakeEase/admin". On the left, there is a dark sidebar with a user profile picture of "piyush" and a "Admin Control" section containing buttons for "Dashboard", "Manage Products", and "Add Products". The main content area is titled "Product Overview" and displays the following statistics:

- Total Sales: Rs. 2440.0
- Total Products: 12
- Product Categories: 3
- Registered Customers: 2
- Best Selling Item: Tea

Below these stats is a table showing 13 products with columns for ID, Name, Description, Price, Category, and Total Sales. The table data is as follows:

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, cinnamon 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

At the bottom, there is a chart titled "Product Sales Overview" showing a single bar for "Total Sales" reaching approximately 500 on the y-axis.

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

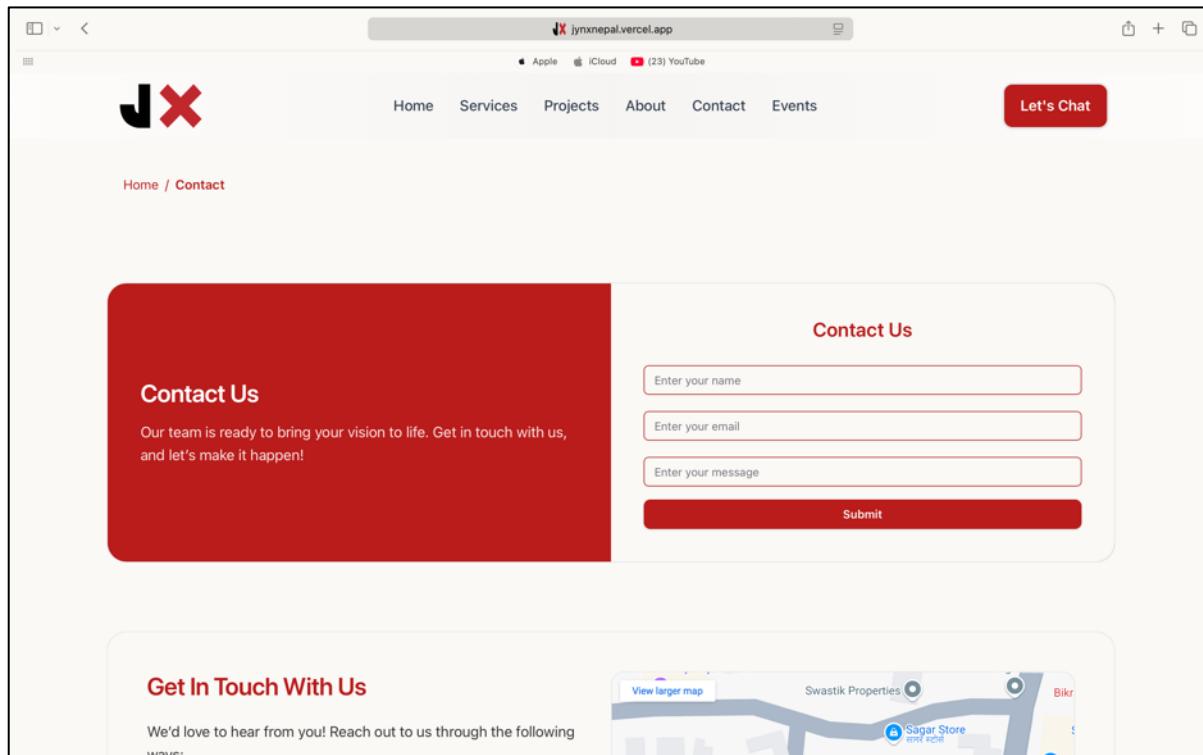


Figure 24: research reference of contact page

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

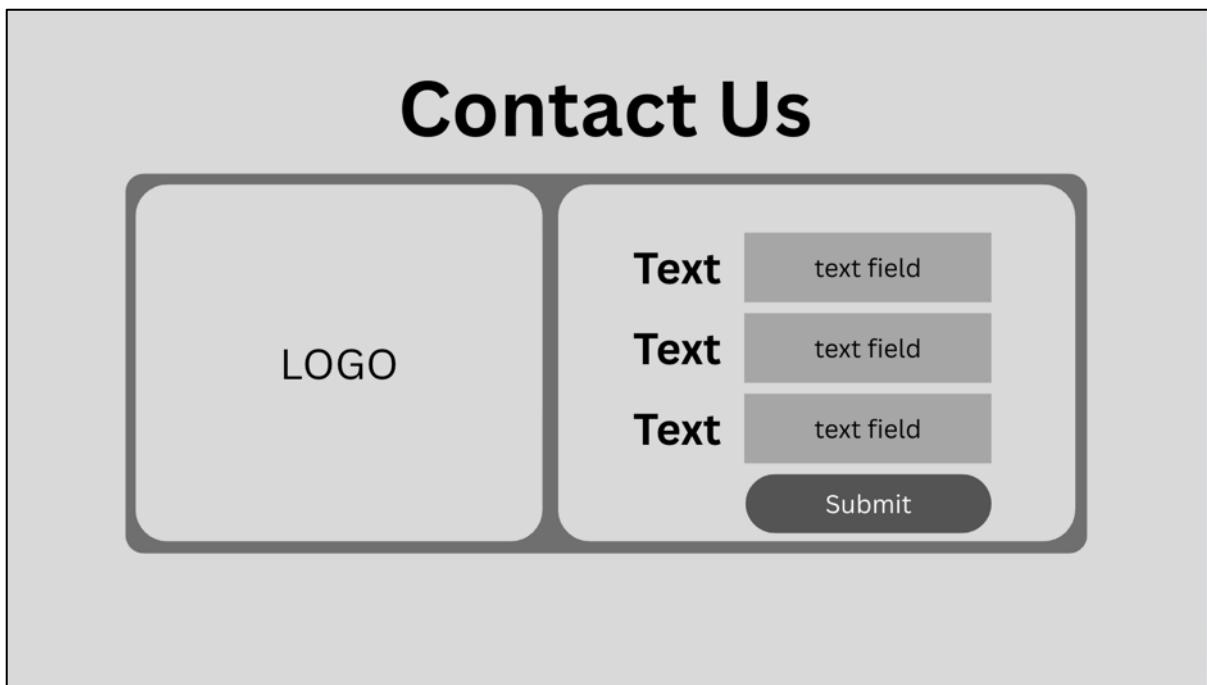


Figure 25: Wireframe of contact page

3.6.3. Design development

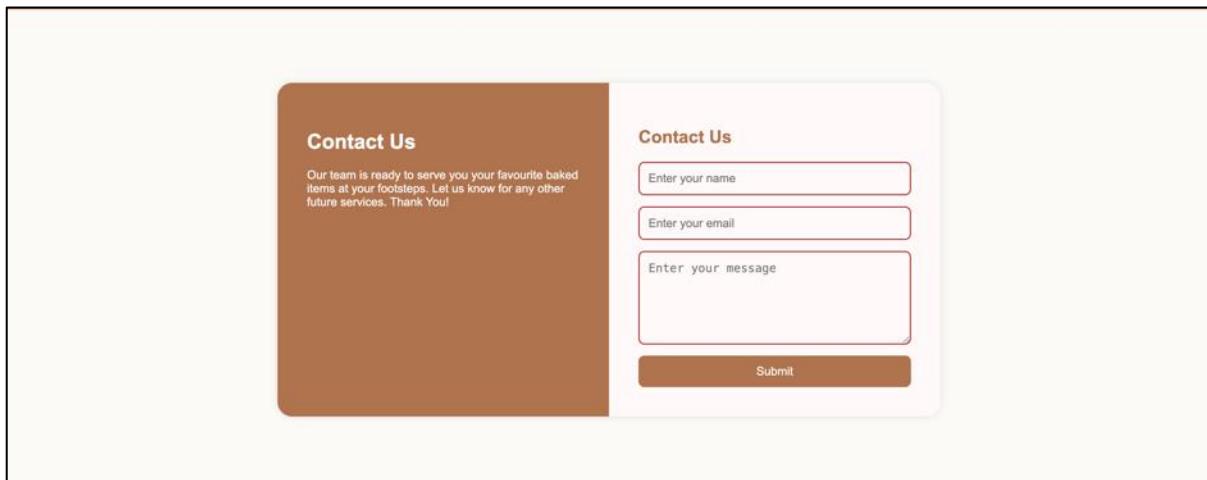


Figure 26: design of contact page

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

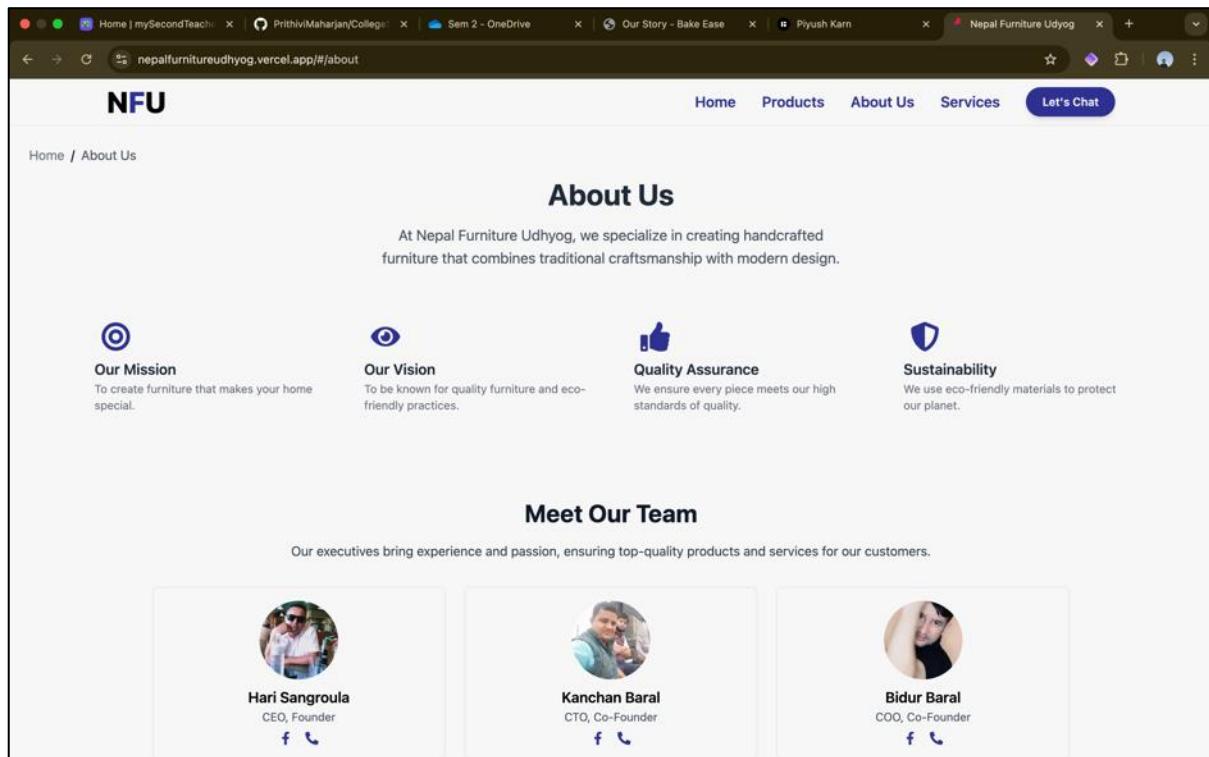


Figure 27: research reference of about page

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

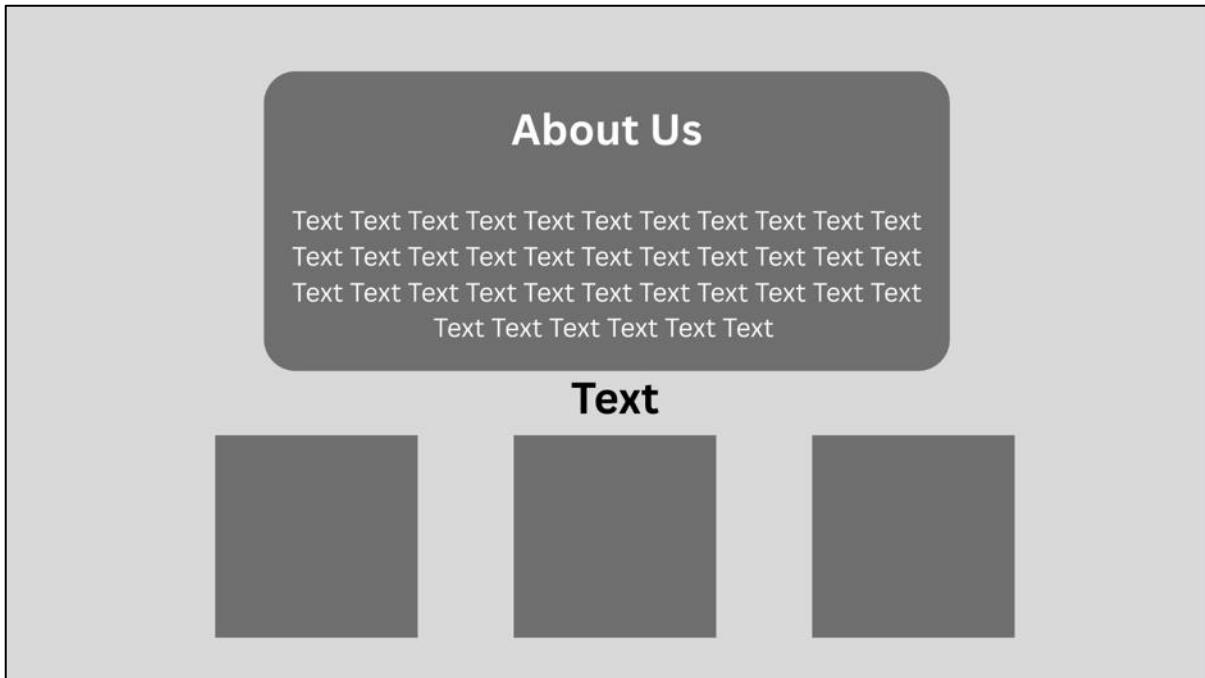


Figure 28: wireframe design of about us page

3.7.3. Design Development

Bake Ease - Order your favourite baked items with ease at your doorsteps

Home Products About Us Contact Profile

Our Story

At Bake Ease, we're more than just a bakery — we're a family-run business with a love for bringing smiles through the power of freshly baked goods. Our mission is to deliver delicious, handcrafted products that warm hearts and homes alike.

Rooted in tradition and passion, we value quality, consistency, and connection with our community. Every recipe we bake is a labor of love, and every customer, a part of our story.

Meet Our Team

Our team at Bake Ease combines culinary expertise, creative flair, and warm hospitality to deliver delightful experiences.



Figure 29: design of about page (i)

Meet Our Team

Our team at Bake Ease combines culinary expertise, creative flair, and warm hospitality to deliver delightful experiences.



Piyush Karn
CEO
[View Portfolio](#)



Aashutosh Dhakal
Operations Manager
[View Portfolio](#)



Arpan Nepal
Food Testing Manager
[View Portfolio](#)

Bake Ease
Delighting your taste buds with fresh baked goods, every single day.

Quick Links

- [Home](#)
- [About](#)
- [Products](#)
- [Contact](#)

Contact Us

Email: support@bakeease.com
Phone: +977-9898989898
Location: Kathmandu, Nepal

© 2025 Bake Ease. All rights reserved.

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.

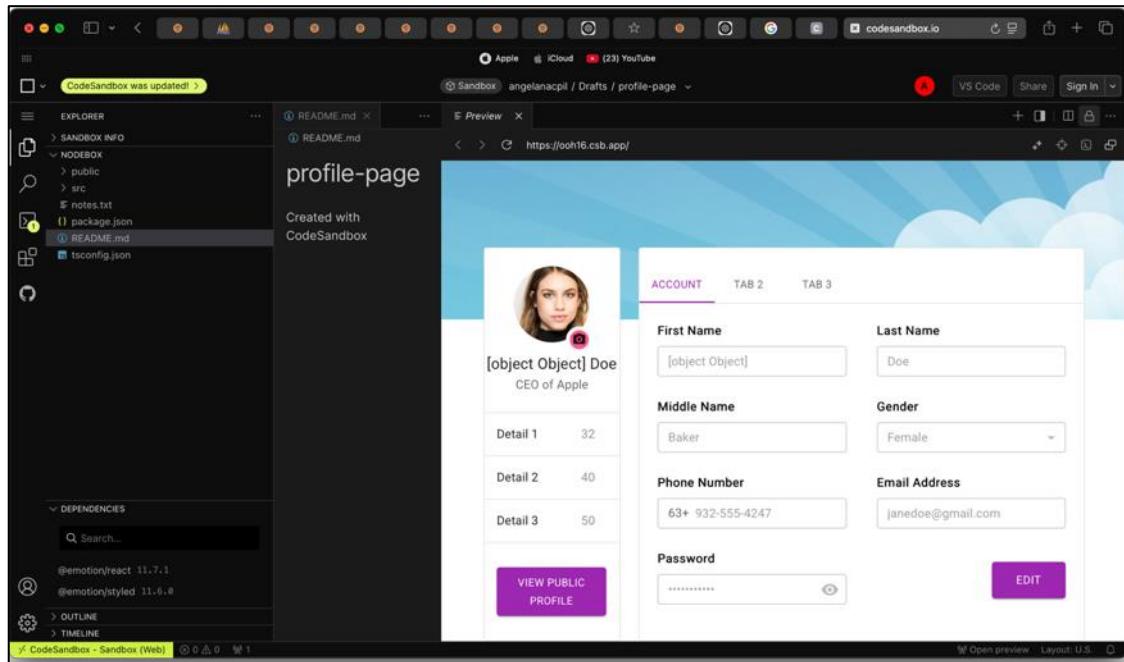


Figure 31: research reference for profile page

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

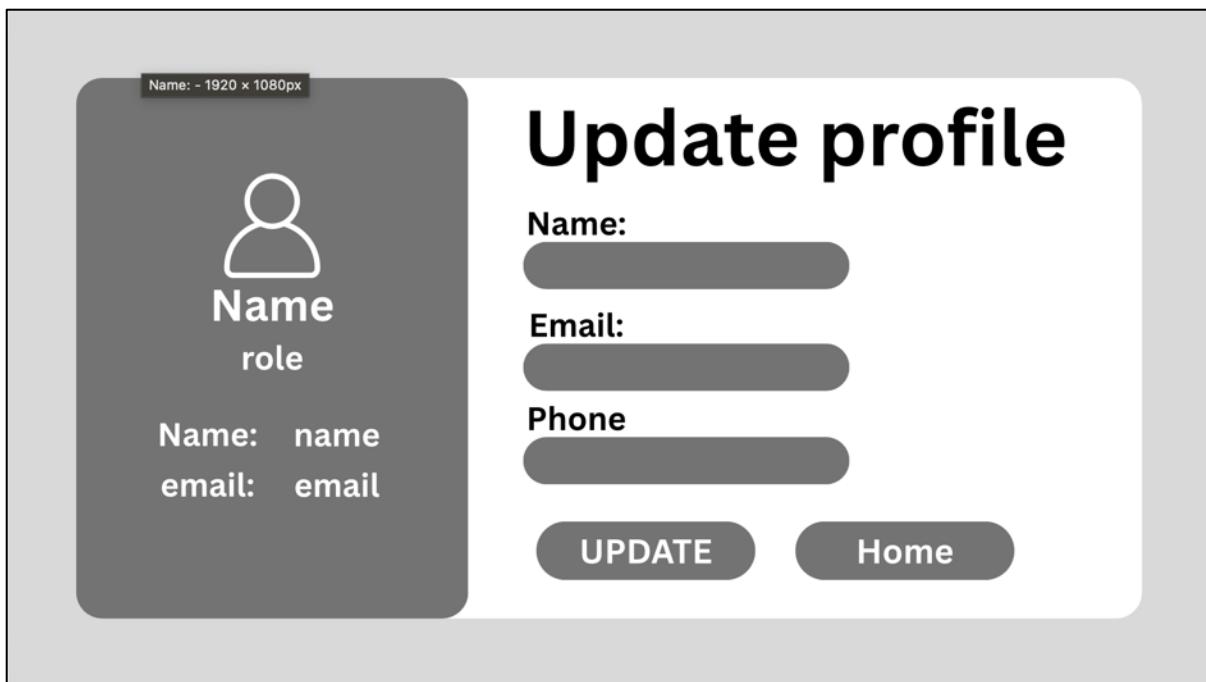


Figure 32: wireframe of profile page

3.8.3.Design Development

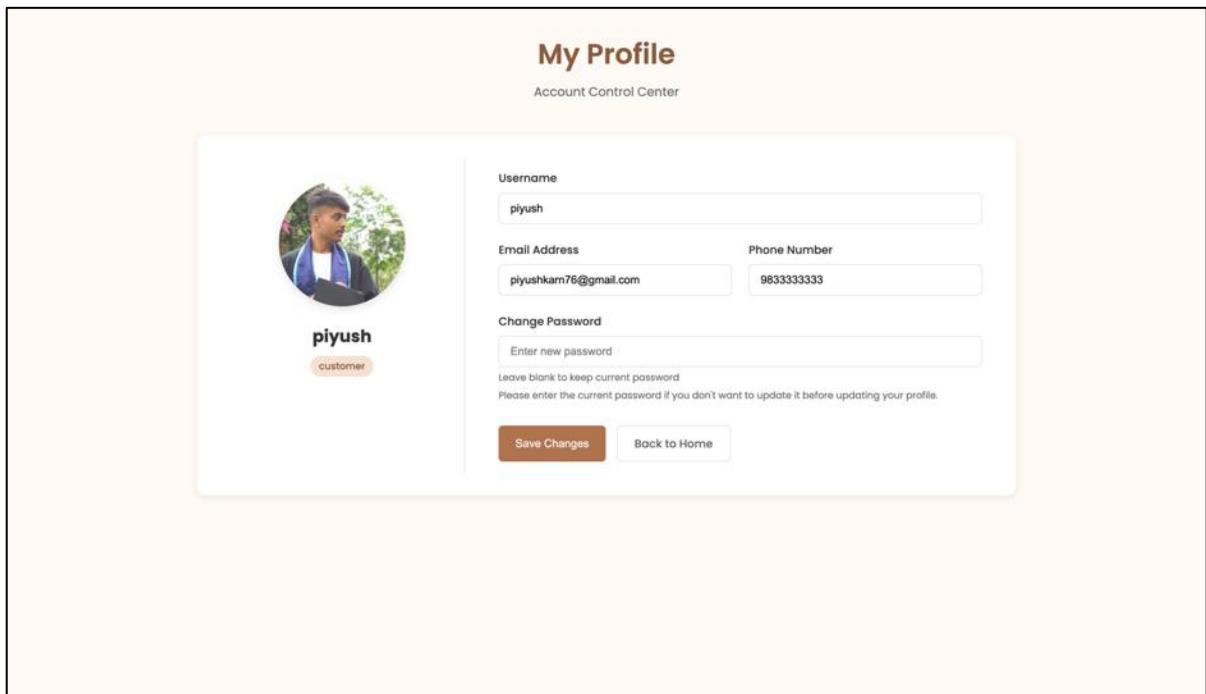


Figure 33: actual design of profile page

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.)

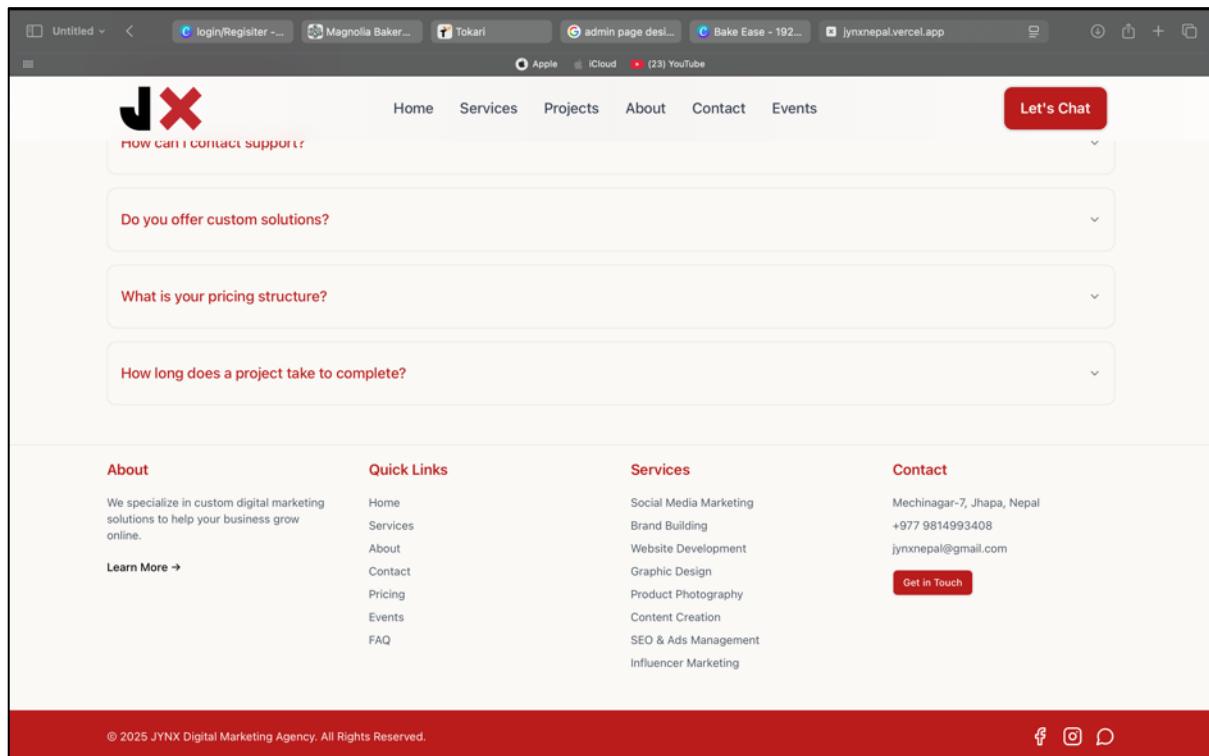


Figure 34: Research reference for footer

3.9.2. Wireframe Design

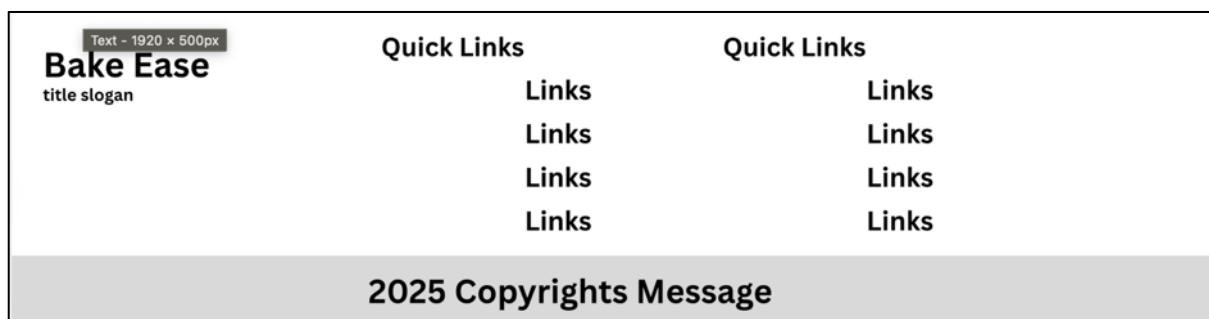


Figure 35: Wireframe design of footer

3.9.3.Design Development

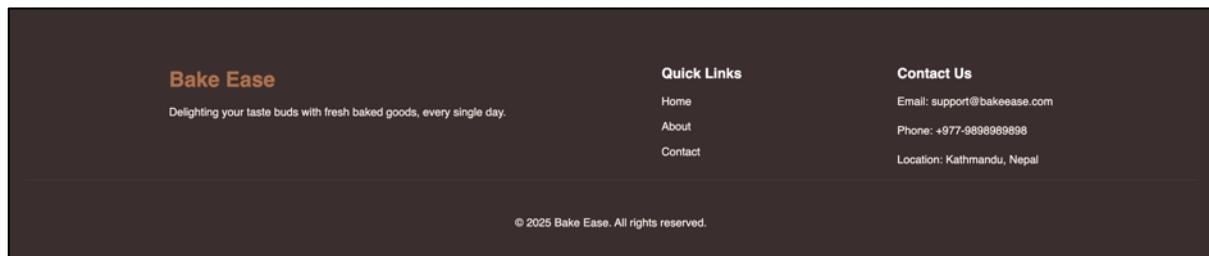


Figure 36: design of footer

4. Java Classes and Methods

4.1. Overall Class Diagram

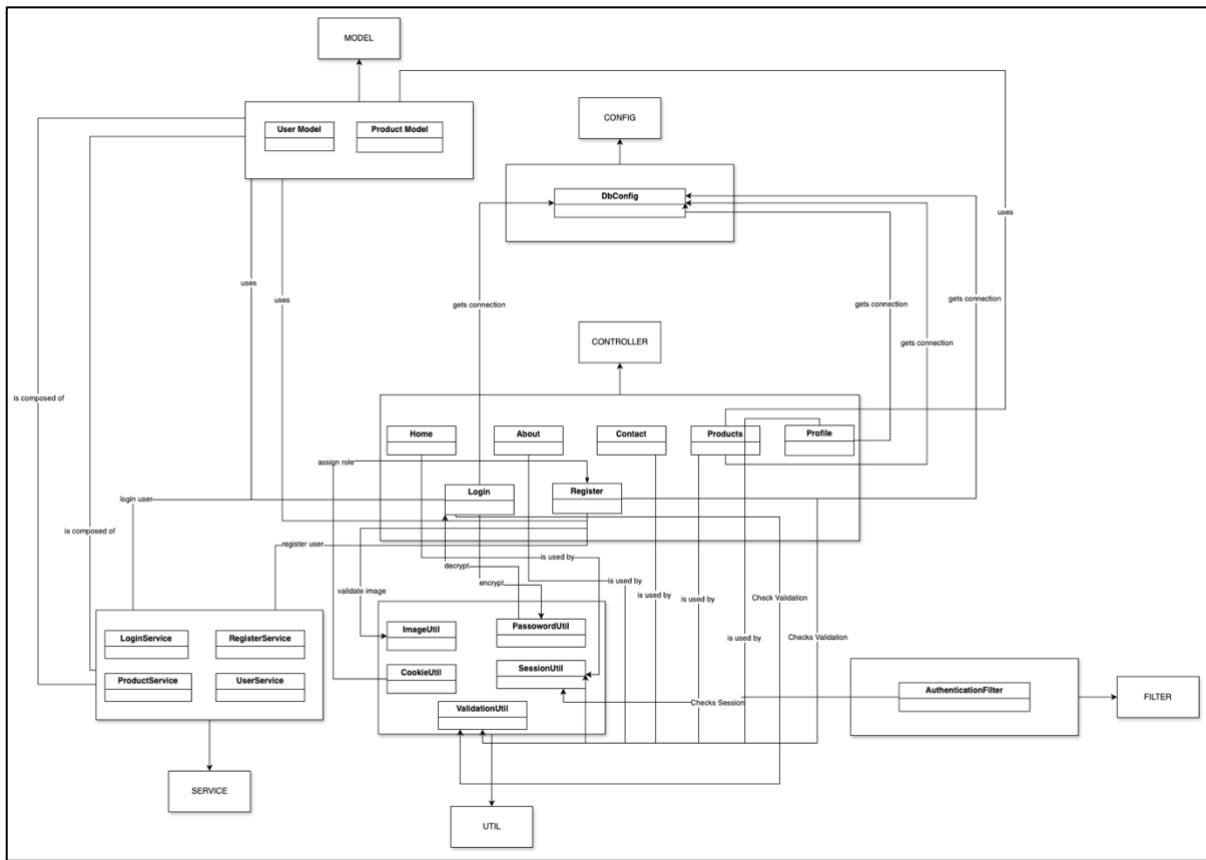


Figure 37: overall class diagram

4.2. Individual Class Diagram and Methods Description

4.2.1.DbConfig

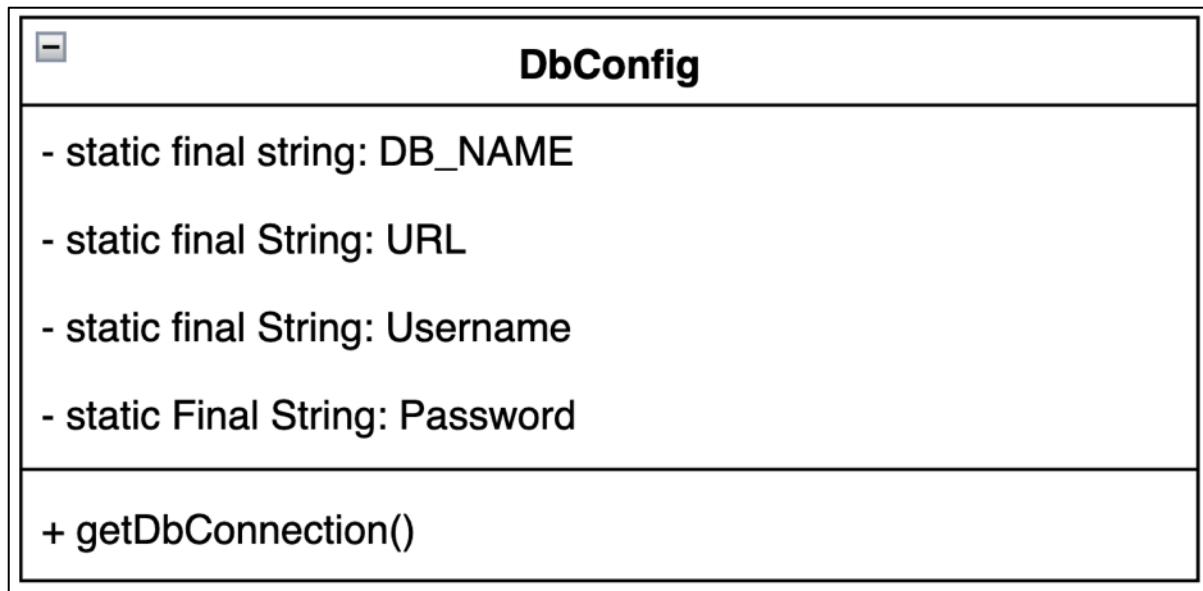


Figure 38: class diagram of DbConfig

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.

```

1 package com.bakeease.config;
2
3 import java.sql.Connection;
4
5 /**
6  * Database configuration class for BakeEase application.
7  * Provides method to establish a connection with the database.
8 */
9 public class DbConfig {
10
11     // Database configuration information
12     private static final String DB_NAME = "BakeEaseDb"; // Your database name
13     private static final String URL = "jdbc:mysql://localhost:3306/" + DB_NAME;
14     private static final String USERNAME = "root"; // Your DB username
15     private static final String PASSWORD = ""; // Your DB password
16
17     /**
18      * Establishes a connection to the database.
19      *
20      * @return Connection object for the database
21      * @throws SQLException if a database access error occurs
22      * @throws ClassNotFoundException if the JDBC driver class is not found
23      */
24     public static Connection getDbConnection() throws SQLException, ClassNotFoundException {
25         Class.forName("com.mysql.cj.jdbc.Driver");
26         return DriverManager.getConnection(URL, USERNAME, PASSWORD);
27     }
28 }
29
30 }
  
```

The code is a Java class named 'DbConfig' located in the package 'com.bakeease.config'. It contains static final variables for database name, URL, username, and password. It has a public static method 'getDbConnection()' that establishes a connection using the MySQL JDBC driver. The code uses JavaDoc comments to describe the class and its methods.

Figure 39: code for DbConfig file

4.2.2. About.java controller

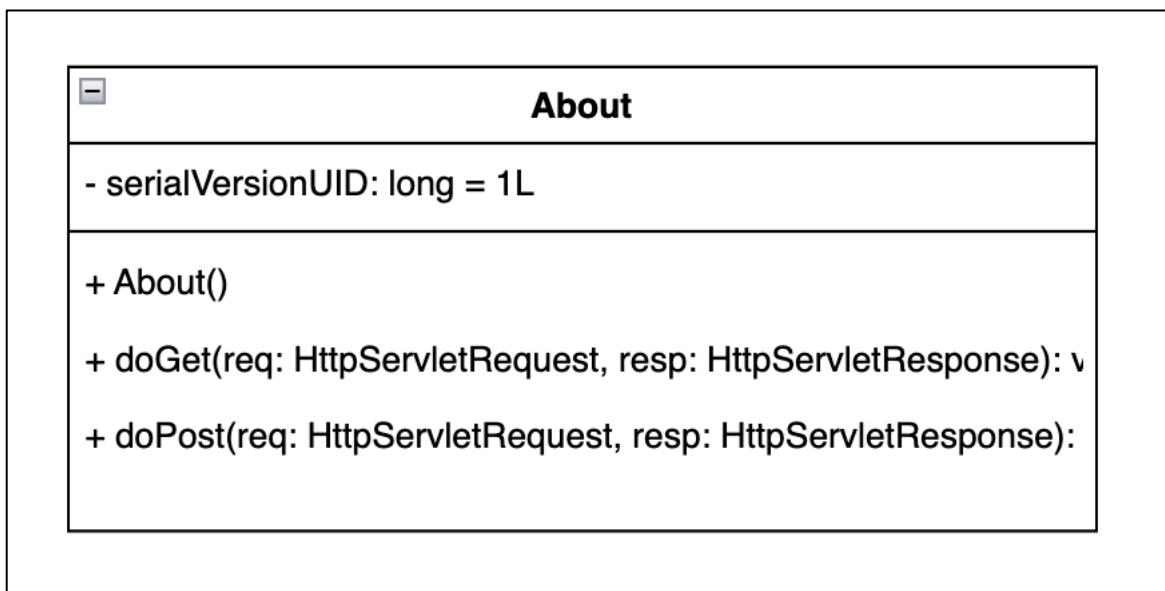


Figure 40: method description for about controller

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.

```

1 * The ABOUT servlet handles HTTP requests for the "/about" URL.
2 * It is used to forward users to the "about.jsp" page located in the "WEB-INF/pages" directory.
3 *
4 * This servlet responds to both GET and POST requests.
5 */
6 @WebServlet(asyncSupported = true, urlPatterns = { "/about" })
7 public class About extends HttpServlet {
8
9     private static final long serialVersionUID = 1L;
10
11    /**
12     * Default constructor.
13     * Calls the superclass constructor (HttpServlet).
14     */
15    public About() {
16        super();
17        // Default constructor
18    }
19
20    /**
21     * Handles GET requests for the "/about" URL.
22     * Forwards the request to the "about.jsp" page.
23     *
24     * @param req the HttpServletRequest object that contains the request the client made to the servlet
25     * @param resp the HttpServletResponse object that contains the response the servlet returns to the client
26     * @throws ServletException if the request could not be handled
27     * @throws IOException if an input or output error is detected when the servlet handles the request
28     */
29    @Override
30    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
31        req.getRequestDispatcher("WEB-INF/pages/about.jsp").forward(req, resp);
32    }
33
34    /**
35     * Handles POST requests for the "/about" URL.
36     * Delegates handling to the doGet() method, treating POST requests the same as GET requests.
37     *
38     * @param request the HttpServletRequest object that contains the request the client made to the servlet
39     * @param response the HttpServletResponse object that contains the response the servlet returns to the client
40     * @throws ServletException if the request could not be handled
41     * @throws IOException if an input or output error is detected when the servlet handles the request
42     */
43    @Override
44    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
45        doGet(request, response);
46    }
47
48 }
49
50 }
51
52 }
53
54 }
55
56 }
57 }

```

Figure 41: code for doGet and doPost method

4.2.3. Admin.java Controller

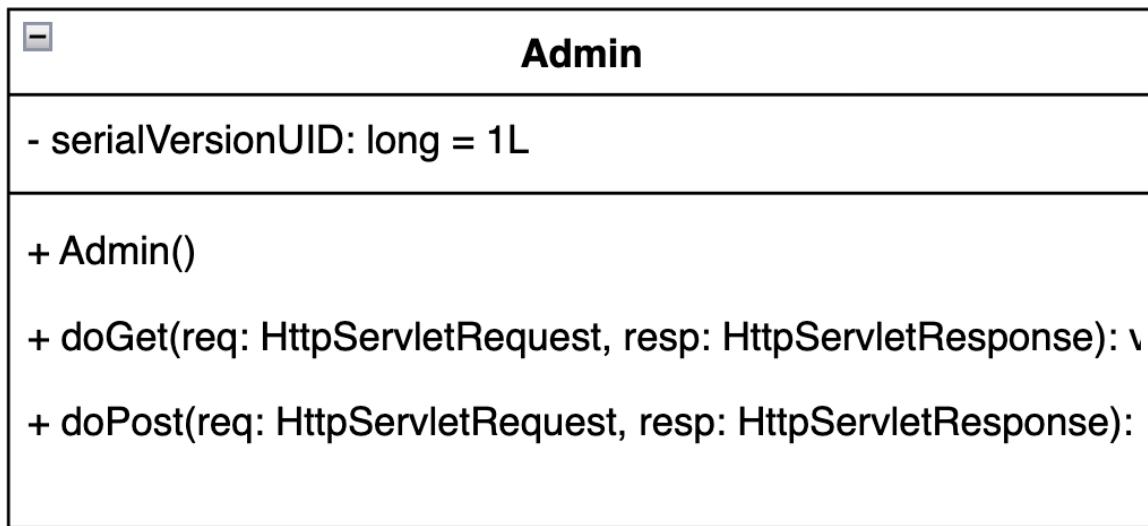


Figure 42: class diagram of about controller

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.

```

1 Admin.java X
42 * It also checks for a valid session and loads user-specific and product data.
43 *
44 * @param request the HttpServletRequest object that contains the request the client made to the servlet
45 * @param response the HttpServletResponse object that contains the response the servlet returns to the client
46 * @throws ServletException if an error occurs during request forwarding
47 * @throws IOException if an I/O error occurs
48 */
49 @Override
50 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
51     // Check session for authentication
52     HttpSession session = request.getSession(false);
53     if (session == null || session.getAttribute("username") == null) {
54         response.sendRedirect(request.getContextPath() + "/login");
55         return;
56     }
57     // Retrieve logged-in user details
58     String username = (String) session.getAttribute("username");
59     UserModel user = userService.getUserByUsername(username);
60     if (user != null) {
61         request.setAttribute("user", user);
62     }
63
64     // Handle different actions: show addProduct or viewProducts page
65     String action = request.getParameter("action");
66
67     if ("addProduct".equals(action)) {
68         request.getRequestDispatcher("/WEB-INF/pages/addProduct.jsp").forward(request, response);
69     } else if ("viewProducts".equals(action)) {
70         List<ProductModel> products = productService.getAllProducts();
71         request.setAttribute("products", products);
72         request.getRequestDispatcher("/WEB-INF/pages/viewProducts.jsp").forward(request, response);
73     } else {
74         // Load default admin dashboard data
75         List<ProductModel> products = productService.getAllProducts();
76
77         double totalSales = products.stream().mapToDouble(ProductModel::getTotalSales).sum();
78         int items = products.size();
79         int categories = 3; // Hardcoded; can be dynamically determined if needed
80         int customers = userService.countCustomers();
81         int orders = products.size(); // Assuming each product counts as one order
82         String getBestProductName = productService.getBestSellingProduct();
83
84         // Set attributes for admin dashboard display
85         request.setAttribute("totalSales", totalSales);
86         request.setAttribute("items", items);
87         request.setAttribute("categories", categories);
88         request.setAttribute("customers", customers);
89         request.setAttribute("orders", orders);
90         request.setAttribute("products", products);
91         request.setAttribute("bestSellingProduct", getBestProductName);
92
93     }
94 }
95 }
96 }
```

Figure 43: code for do get method

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

```
1 Admin.java x
2
3 108 @Override
4 109     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5 110         String action = request.getParameter("action");
6 111
7 112         try {
8 113             // Add a new product
9 114             if ("add".equals(action)) {
10 115                 String name = request.getParameter("name");
11 116                 String description = request.getParameter("description");
12 117                 double price = Double.parseDouble(request.getParameter("price"));
13 118                 String category = request.getParameter("category");
14 119                 int totalSales = Integer.parseInt(request.getParameter("total_sales"));
15
16 120                 ProductModel product = new ProductModel();
17 121                 product.setName(name);
18 122                 product.setDescription(description);
19 123                 product.setPrice(price);
20 124                 product.setCategory(category);
21 125                 product.setTotalSales(totalSales);
22
23 126                 productService.addProduct(product);
24 127                 request.setAttribute("successMessage", "Product added successfully!");
25
26 128             // Update an existing product
27 129             } else if ("update".equals(action)) {
28 130                 int id = Integer.parseInt(request.getParameter("id"));
29 131                 String name = request.getParameter("name");
30 132                 String description = request.getParameter("description");
31 133                 double price = Double.parseDouble(request.getParameter("price"));
32 134                 String category = request.getParameter("category");
33 135                 int totalSales = Integer.parseInt(request.getParameter("total_sales"));
34
35 136                 ProductModel product = new ProductModel();
36 137                 product.setId(id);
37 138                 product.setName(name);
38 139                 product.setDescription(description);
39 140                 product.setPrice(price);
40 141                 product.setCategory(category);
41 142                 product.setTotalSales(totalSales);
42
43 143                 productService.updateProduct(product);
44 144                 request.setAttribute("successMessage", "Product updated successfully!");
45
46 145             // Delete a product
47 146             } else if ("delete".equals(action)) {
48 147                 int id = Integer.parseInt(request.getParameter("id"));
49 148                 productService.deleteProduct(id);
50 149                 request.setAttribute("successMessage", "Product deleted successfully!");
51
52 150         }
53
54 155     } catch (Exception e) {
55 156         e.printStackTrace();
56 157         request.setAttribute("successMessage", "An error occurred while processing the request.");
57
58 158     }
59
60 161 }
```

Figure 44: code for doPost method

4.2.4. Contact.java controller

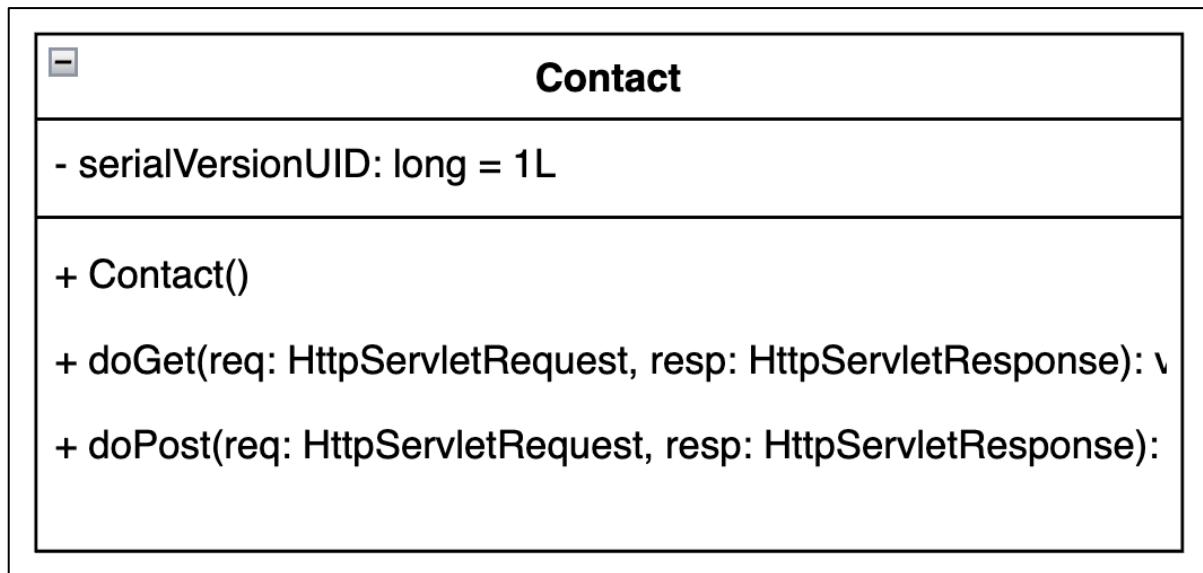
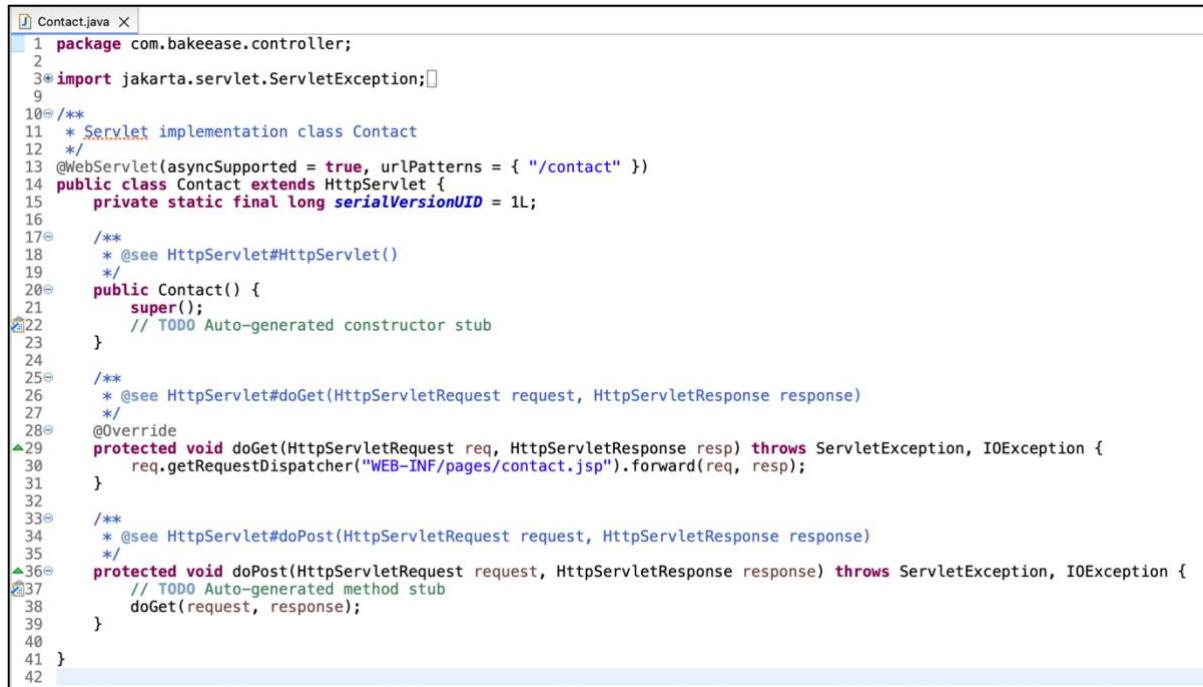


Figure 45: class diagram of contact controller

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.



```
1 package com.bakeease.controller;
2
3 import jakarta.servlet.ServletException;
4
5 /**
6  * Servlet implementation class Contact
7  */
8 @WebServlet(asyncSupported = true, urlPatterns = { "/contact" })
9 public class Contact extends HttpServlet {
10     private static final long serialVersionUID = 1L;
11
12     /**
13      * @see HttpServlet#HttpServlet()
14      */
15     public Contact() {
16         super();
17         // TODO Auto-generated constructor stub
18     }
19
20     /**
21      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
22      */
23     @Override
24     protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
25         req.getRequestDispatcher("WEB-INF/pages/contact.jsp").forward(req, resp);
26     }
27
28     /**
29      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
30      */
31     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
32         // TODO Auto-generated method stub
33         doGet(request, response);
34     }
35 }
36
37 }
```

Figure 46: code for doGet and doPost method

4.2.5. Home.java controller

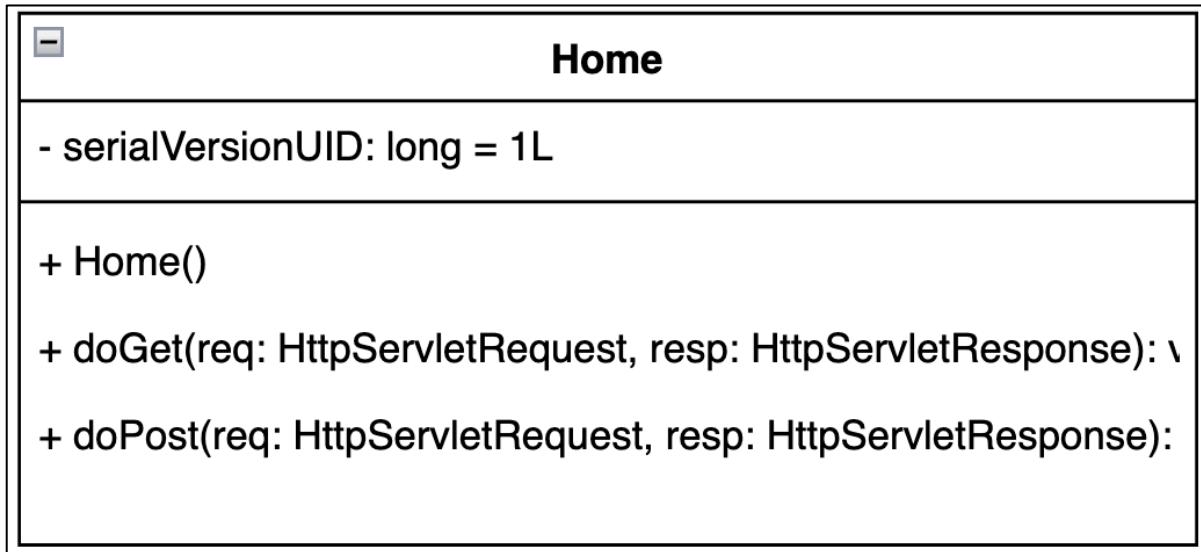


Figure 47: class diagram of home controller

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 whereas the doPost method is used to handle the Get HTTP request of the home path.

```

Home.java X
1 package com.bakeease.controller;
2
3 import jakarta.servlet.ServletException;
4
5 /**
6  * Servlet implementation class Home
7  * Handles requests for the home page.
8  *
9  * @author
10 * Piyush Karn
11 */
12 @WebServlet(asyncSupported = true, urlPatterns = {"/home", "/"})
13 public class Home extends HttpServlet {
14     private static final long serialVersionUID = 1L;
15
16     /**
17      * Handles GET requests and forwards to home.jsp.
18      */
19     @Override
20     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
21         request.getRequestDispatcher("WEB-INF/pages/home.jsp").forward(request, response);
22     }
23
24     /**
25      * Handles POST requests by retrieving form data and forwarding to home.jsp.
26      */
27     @Override
28     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
29         // Example: Processing form inputs
30         String username = request.getParameter("username");
31         String message = request.getParameter("message");
32
33         // Setting attributes to send back to JSP
34         request.setAttribute("username", username);
35         request.setAttribute("message", message);
36
37         // Forwarding to home.jsp with attributes
38         request.getRequestDispatcher("WEB-INF/pages/home.jsp").forward(request, response);
39     }
40 }
41
42
43
44
45
46

```

Figure 48: code for doGet and doPost

4.2.6.Login.java controller

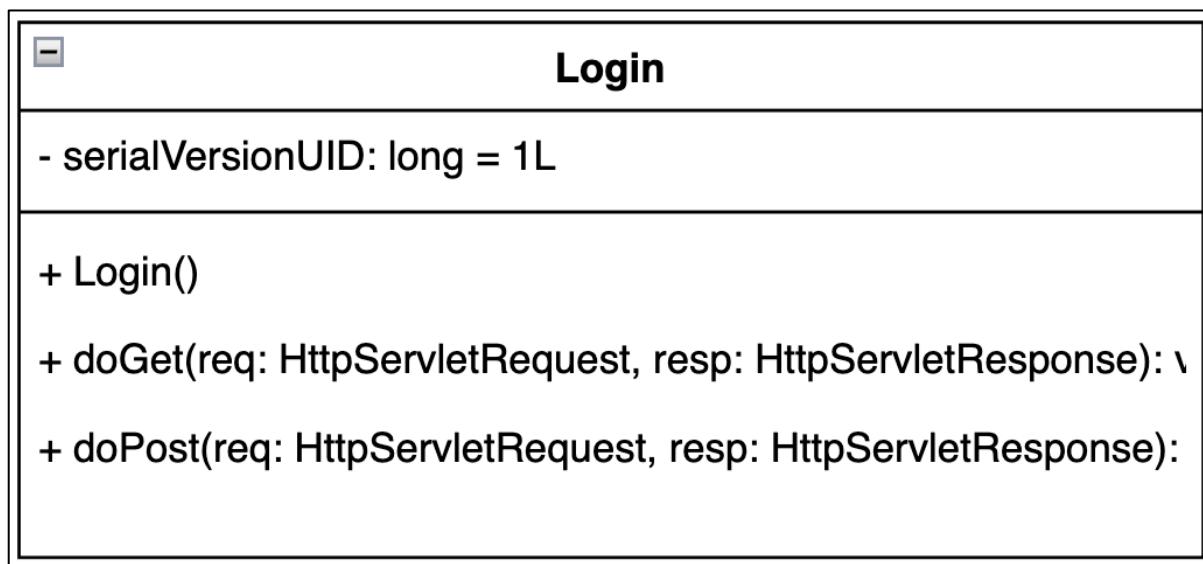


Figure 49: class diagram of login controller

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.

```

1 package com.bakeease.controller;
2
3 import com.bakeease.service.LoginService;
4
5 public class Login extends HttpServlet {
6     private static final long serialVersionUID = 1L;
7
8     private final LoginService loginService = new LoginService();
9
10    @Override
11    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
12        req.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(req, resp);
13    }
14
15    @Override
16    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
17        request.setCharacterEncoding("UTF-8");
18
19        String username = request.getParameter("username");
20        String inputPassword = request.getParameter("password");
21
22        String role = loginService.authenticate(username, inputPassword);
23
24        if (role != null) {
25            // Login successful
26            HttpSession session = request.getSession();
27            session.setAttribute("username", username);
28
29            // Set role cookie
30            CookieUtil.addCookie(response, "role", role, 60 * 60); // 1 hour validity
31
32            if ("admin".equalsIgnoreCase(role)) {
33                response.sendRedirect(request.getContextPath() + "/admin");
34            } else {
35                response.sendRedirect(request.getContextPath() + "/home");
36            }
37        }
38
39        // Login failed
40        request.setAttribute("error", "Invalid username or password.");
41        request.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(request, response);
42    }
43}

```

Figure 50: code for doGet and doPost

4.2.7. Product.java controller

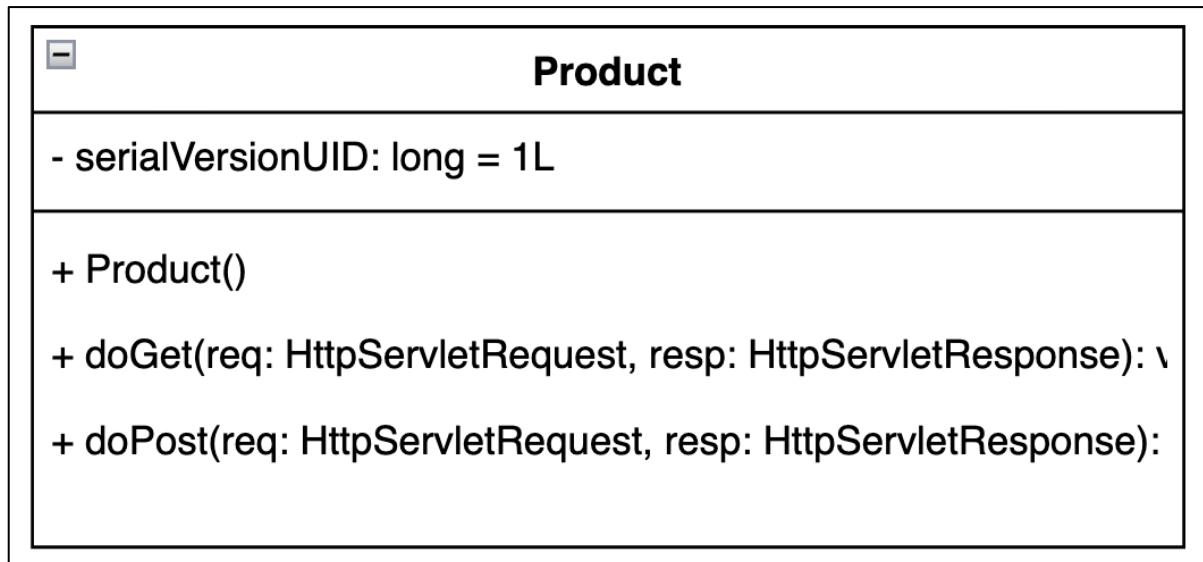
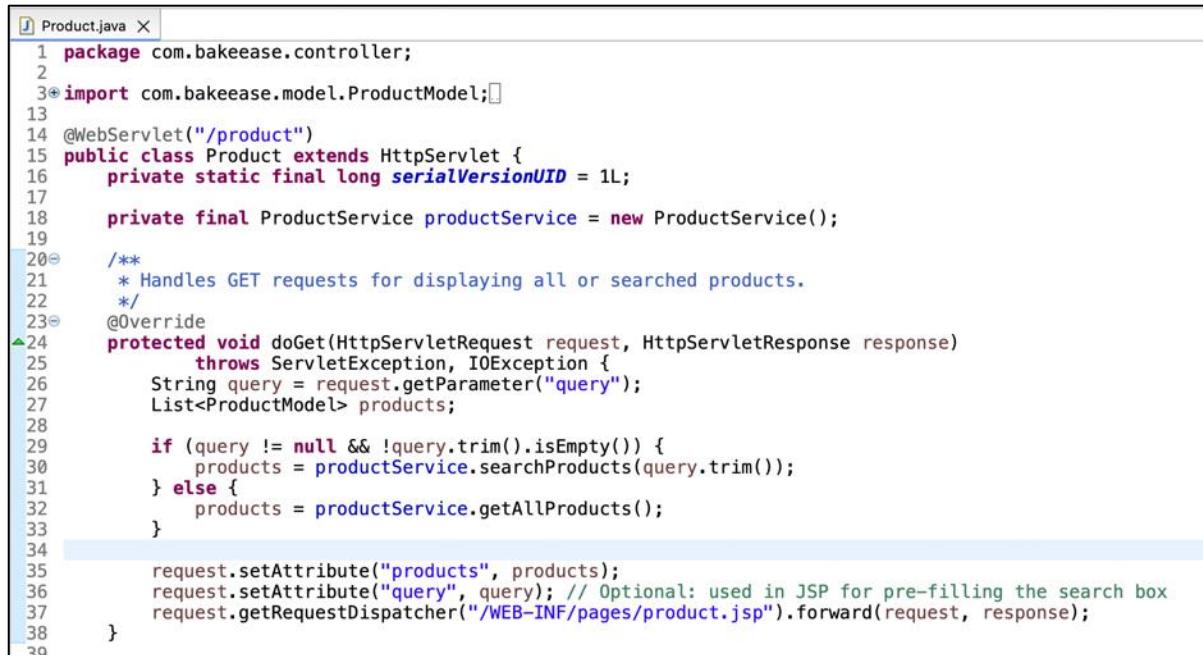


Figure 51: class diagram of product controller

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.



```

1 package com.bakeease.controller;
2
3 import com.bakeease.model.ProductModel;
4
5 @WebServlet("/product")
6 public class Product extends HttpServlet {
7     private static final long serialVersionUID = 1L;
8
9     private final ProductService productService = new ProductService();
10
11    /**
12     * Handles GET requests for displaying all or searched products.
13     */
14    @Override
15    protected void doGet(HttpServletRequest request, HttpServletResponse response)
16         throws ServletException, IOException {
17         String query = request.getParameter("query");
18         List<ProductModel> products;
19
20         if (query != null && !query.trim().isEmpty()) {
21             products = productService.searchProducts(query.trim());
22         } else {
23             products = productService.getAllProducts();
24         }
25
26         request.setAttribute("products", products);
27         request.setAttribute("query", query); // Optional: used in JSP for pre-filling the search box
28         request.getRequestDispatcher("/WEB-INF/pages/product.jsp").forward(request, response);
29     }
30 }
```

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.

```

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

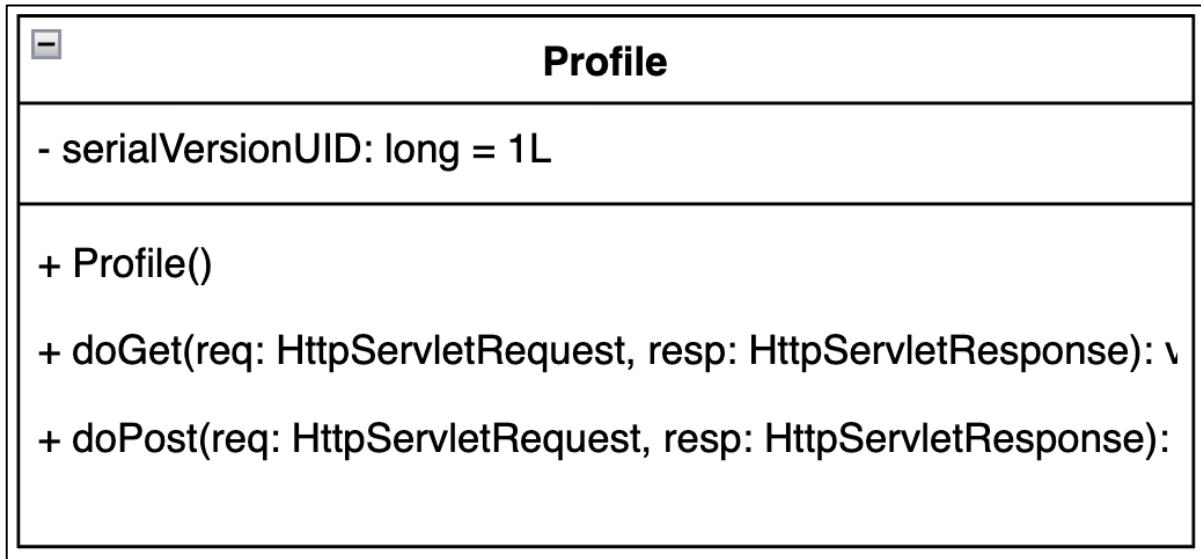


Figure 54: class diagram of profile controller

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 if user or session is not found in the profile.jsp file in the web page.

```

Profile.java X
1 package com.bakeease.controller;
2
3 import com.bakeease.model.UserModel;
4
5 @WebServlet("/profile")
6 @MultipartConfig
7 public class Profile extends HttpServlet {
8     private static final long serialVersionUID = 1L;
9
10    private final UserService userService = new UserService();
11
12    @Override
13    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
14        HttpSession session = request.getSession(false);
15
16        if (session == null || session.getAttribute("username") == null) {
17            response.sendRedirect(request.getContextPath() + "/login");
18            return;
19        }
20
21        String username = (String) session.getAttribute("username");
22        UserModel user = userService.getUserByUsername(username);
23
24        if (user != null) {
25            request.setAttribute("user", user);
26            request.getRequestDispatcher("/WEB-INF/pages/profile.jsp").forward(request, response);
27        } else {
28            request.setAttribute("error", "User not found.");
29            request.getRequestDispatcher("/WEB-INF/pages/error.jsp").forward(request, response);
30        }
31    }
32
33 }

```

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

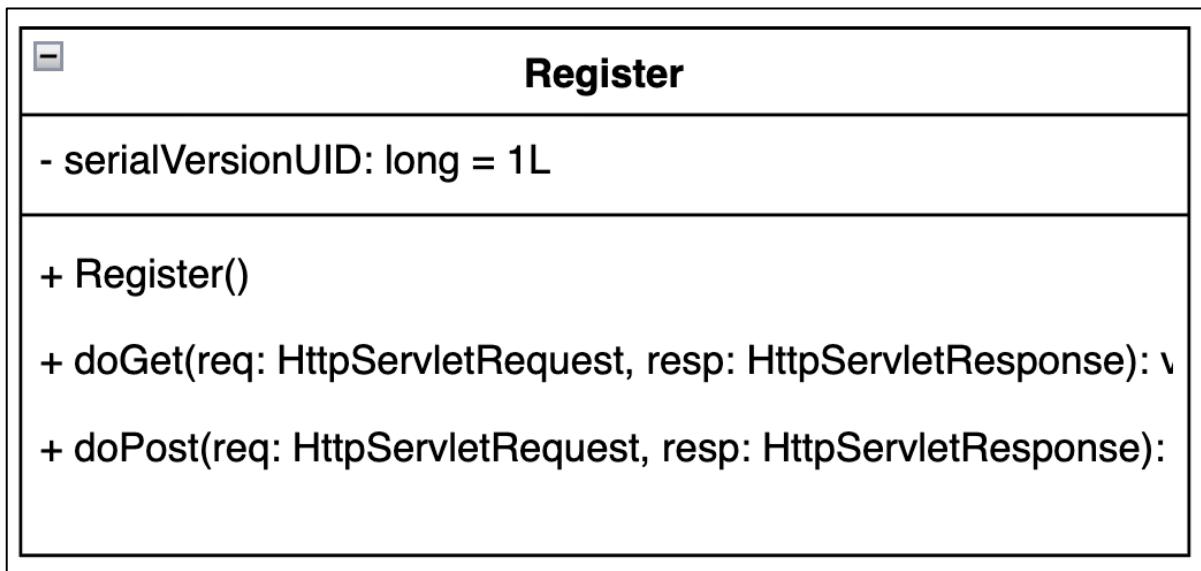


Figure 57: class diagram of register controller

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 X
34
35@override
36protected 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.

```
40 @Override
41 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
42     request.setCharacterEncoding("UTF-8");
43
44     String username = request.getParameter("username");
45     String email = request.getParameter("email");
46     String password = request.getParameter("password");
47     String confirmPassword = request.getParameter("confirmPassword");
48     String phone = request.getParameter("phone");
49     Part image = request.getPart("image");
50
51     boolean isValid = true;
52
53     // Validation checks
54     if (Validationutil.isNullOrEmpty(username)) {
55         request.setAttribute("usernameError", "Username is required.");
56         isValid = false;
57     } else if (!username.matches("^[a-zA-Z][a-zA-Z0-9_]{2,19}$")) {
58         request.setAttribute("usernameError", "Username must start with a letter and be 3-20 characters long.");
59         isValid = false;
60     }
61
62     if (Validationutil.isNullOrEmpty(email)) {
63         request.setAttribute("emailError", "Email is required.");
64         isValid = false;
65     } else if (!email.matches("[a-zA-Z0-9_.+?@\\.[a-zA-Z0-9_.+?@]+[a-zA-Z]{2,7}$")) {
66         request.setAttribute("emailError", "Please enter a valid email address.");
67         isValid = false;
68     }
69
70     if (Validationutil.isNullOrEmpty(password)) {
71         request.setAttribute("passwordError", "Password is required.");
72         isValid = false;
73     } else if (password.length() < 6) {
74         request.setAttribute("passwordError", "Password must be at least 6 characters long.");
75         isValid = false;
76     }
77
78     if (Validationutil.isNullOrEmpty(confirmPassword)) {
79         request.setAttribute("confirmPasswordError", "Confirm Password is required.");
80         isValid = false;
81     } else if (!password.equals(confirmPassword)) {
82         request.setAttribute("confirmPasswordError", "Password and Confirm Password do not match.");
83         isValid = false;
84     }
85
86     if (Validationutil.isNullOrEmpty(phone)) {
87         request.setAttribute("phoneError", "Phone number is required.");
88         isValid = false;
89     } else if (!phone.matches("^98[0-9]{8}$")) {
90         request.setAttribute("phoneError", "Phone number must start with '98' and be 10 digits.");
91         isValid = false;
92     }
93
94     String fileName = null;
95     if (image == null || image.getSize() == 0) {
96         request.setAttribute("imageError", "Image field is empty. Please upload an image.");
97     }
98 }
```

```

Register.java X
1 if (ValidationUtil.isNullOrEmpty(phone)) {
2     request.setAttribute("phoneError", "Phone number is required.");
3     isValid = false;
4 } else if (!phone.matches("^98[0-9]{8}$")) {
5     request.setAttribute("phoneError", "Phone number must start with '98' and be 10 digits.");
6     isValid = false;
7 }
8
9 String fileName = null;
10 if (image == null || image.getSize() == 0) {
11     request.setAttribute("imageError", "Image field is empty. Please upload an image.");
12     isValid = false;
13 } else if (!ValidationUtil.isValidImageExtension(image)) {
14     request.setAttribute("imageError", "Only upload png, jpg and jpeg image.");
15     isValid = false;
16 } else {
17     fileName = imageUtil.getImageNameFromPart(image);
18     boolean uploadSuccess = imageUtil.uploadImage(image, getServletContext().getRealPath(""), "/customer-images");
19     if (!uploadSuccess) {
20         request.setAttribute("imageError", "Oops! Couldn't upload the image. Give it another shot.");
21         isValid = false;
22     }
23 }
24
25 // Check for duplicate entries
26 if (isValid && registerService.isDuplicateUsernameOrEmail(username, email)) {
27     request.setAttribute("duplicateError", "Duplicate entries found. Please choose another username and email.");
28     isValid = false;
29 }
30
31 if (!isValid) {
32     request.setAttribute("username", username);
33     request.setAttribute("email", email);
34     request.setAttribute("phone", phone);
35     request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
36     return;
37 }
38
39 String encryptedPassword = PasswordUtil.encrypt(username, password);
40 if (encryptedPassword == null) {
41     request.setAttribute("error", "Error encrypting password. Try again.");
42     request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
43     return;
44 }
45
46 String role = "customer";
47 UserModel user = new UserModel(username, email, phone, encryptedPassword, role, fileName);
48
49 boolean isRegistered = registerService.addUser(user);
50 if (isRegistered) {
51     response.sendRedirect(request.getContextPath() + "/login");
52 } else {
53     request.setAttribute("error", "Registration failed. Try again.");
54     request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
55 }
56 }
57 }
58
59 }
60
61 }
62
63 }
64
65 }
66
67 }
68
69 }
70
71 }
72
73 }
74
75 }
76
77 }
78
79 }
80
81 }
82
83 }
84
85 }
86
87 }
88
89 }
90
91 }
92
93 }
94
95 }
96
97 }
98
99 }
100
101 }
102
103 }
104
105 }
106
107 }
108
109 }
110
111 }
112
113 }
114
115 }
116
117 }
118
119 }
120
121 }
122
123 }
124
125 }
126
127 }
128
129 }
130
131 }
132
133 }
134
135 }
136
137 }
138
139 }
140
141 }
142 }

```

Figure 59: code for doPost method

4.2.10. AuthenticationFilter.java Filter

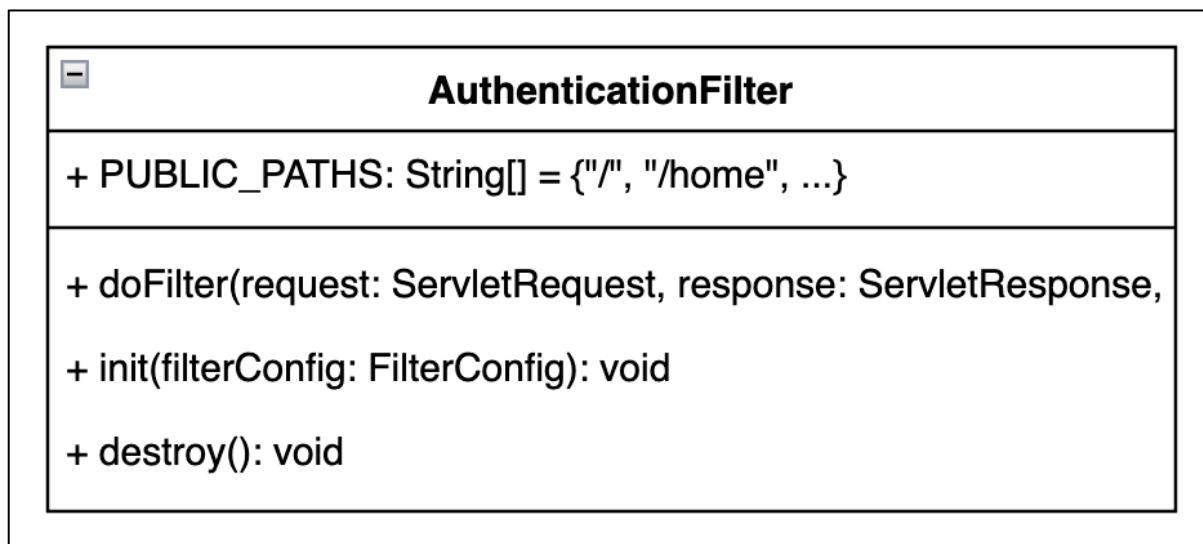


Figure 60: class diagram of authentication filter

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.

```

J AuthenticationFilter.java X
11 @WebFilter(asyncSupported = true, urlPatterns = "/*")
12 public class AuthenticationFilter implements Filter {
13
14     private static final String[] PUBLIC_PATHS = {
15         "/", "/home", "/about", "/login", "/register", "/contact", "/product"
16     };
17
18     @Override
19     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
20             throws IOException, ServletException {
21
22         HttpServletRequest req = (HttpServletRequest) request;
23         HttpServletResponse res = (HttpServletResponse) response;
24         String uri = req.getRequestURI();
25
26         boolean isLoggedIn = SessionUtil.getAttribute(req, "username") != null;
27         String role = CookieUtil.getCookie(req, "role") != null
28             ? CookieUtil.getCookie(req, "role").getValue()
29             : null;
30
31         // Allow static resources (CSS, JS, images, fonts, etc.)
32         if (uri.matches(".*(\.css|\.\js|\.\png|\.\jpg|\.\jpeg|\.\gif|\.\svg|\.\woff|\.\woff2|\.\ttf|\.\eot)$")) {
33             chain.doFilter(request, response);
34             return;
35         }
36
37         // Allow public paths to everyone
38         for (String path : PUBLIC_PATHS) {
39             if (uri.equals(req.getContextPath() + path) || uri.equals(path)) {
40                 chain.doFilter(request, response);
41                 return;
42             }
43         }
44
45         // Admin-only path
46         if (uri.equals(req.getContextPath() + "/admin")) {
47             if (isLoggedIn && "admin".equals(role)) {
48                 chain.doFilter(request, response);
49             } else {
50                 res.sendRedirect(req.getContextPath() + "/login");
51             }
52             return;
53         }
54
55         // Profile page - allow both admin and customer
56         if (uri.equals(req.getContextPath() + "/profile")) {
57             if (isLoggedIn && ("admin".equals(role) || "customer".equals(role))) {
58                 chain.doFilter(request, response);
59             } else {
60                 res.sendRedirect(req.getContextPath() + "/login");
61             }
62             return;
63         }
64
65         // Default case - block and redirect to login
66         res.sendRedirect(req.getContextPath() + "/login");
67     }
}

```

Figure 61: doFilter code for authentication filter

4.2.11. ProductModel.java Model

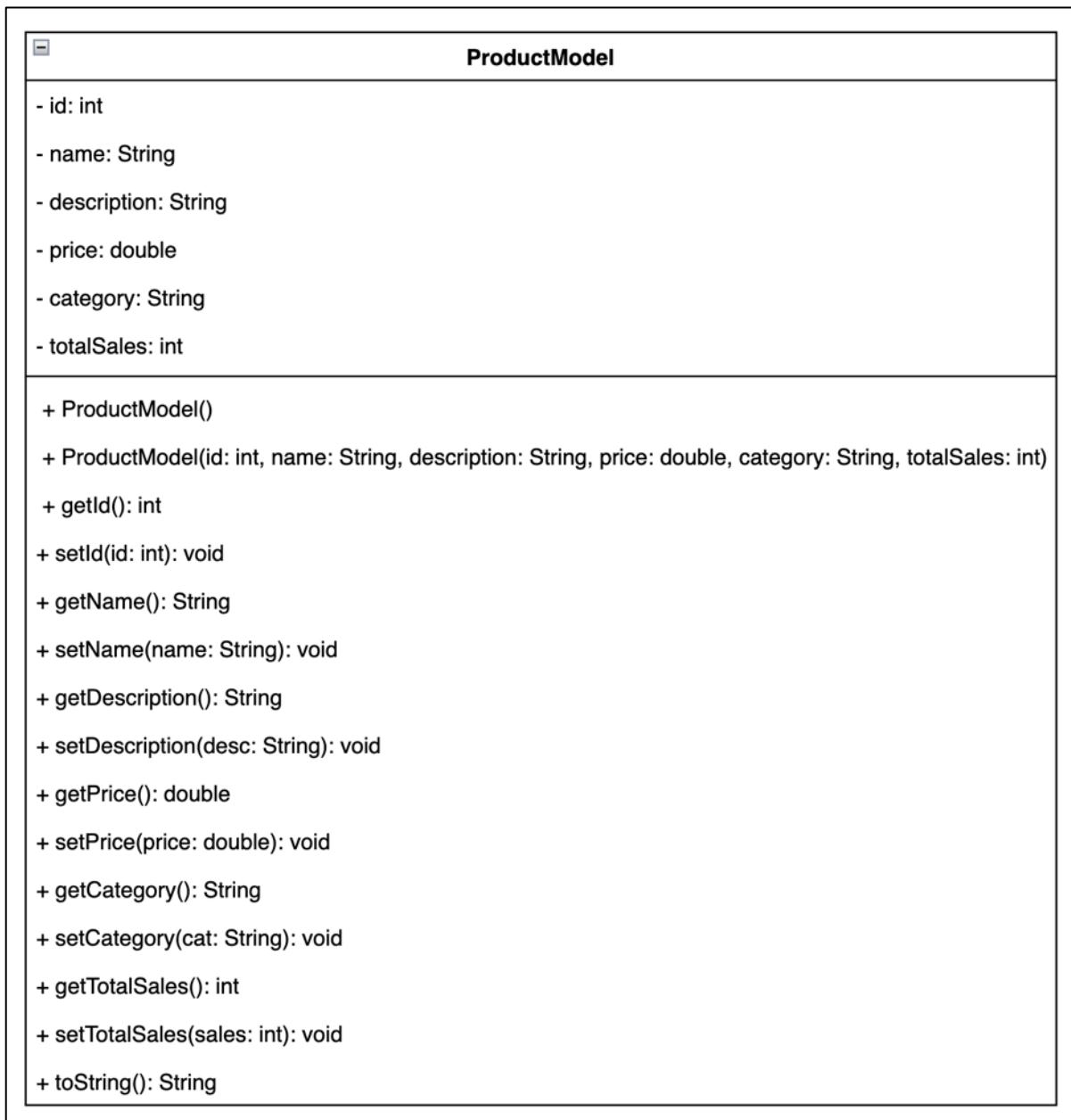


Figure 62: class diagram of product model

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.



```

1 package com.bakeease.model;
2
3 public class ProductModel {
4     private int id;
5     private String name;
6     private String description;
7     private double price;
8     private String category;
9     private int totalSales;
10
11    // Default Constructor
12    public ProductModel() {}
13
14    // Constructor with all parameters
15    public ProductModel(int id, String name, String description, double price, String category, int totalSales) {
16        this.id = id;
17        this.name = name;
18        this.description = description;
19        this.price = price;
20        this.category = category;
21        this.totalSales = totalSales;
22    }
23
24    // Getters and Setters
25    public int getId() { return id; }
26    public void setId(int id) { this.id = id; }
27
28    public String getName() { return name; }
29    public void setName(String name) { this.name = name; }
30
31    public String getDescription() { return description; }
32    public void setDescription(String description) { this.description = description; }
33
34    public double getPrice() { return price; }
35    public void setPrice(double price) { this.price = price; }
36
37    public String getCategory() { return category; }
38    public void setCategory(String category) { this.category = category; }
39
40    public int getTotalSales() { return totalSales; }
41    public void setTotalSales(int totalSales) { this.totalSales = totalSales; }
42
43    @Override
44    public String toString() {
45        return "ProductModel{" +
46            "id=" + id +
47            ", name='" + name + '\'' +
48            ", description='" + description + '\'' +
49            ", price=" + price +
50            ", category='" + category + '\'' +
51            ", totalSales=" + totalSales +
52            '}';
53    }
}

```

Figure 63: code of product model

4.2.12. UserModel.java Model

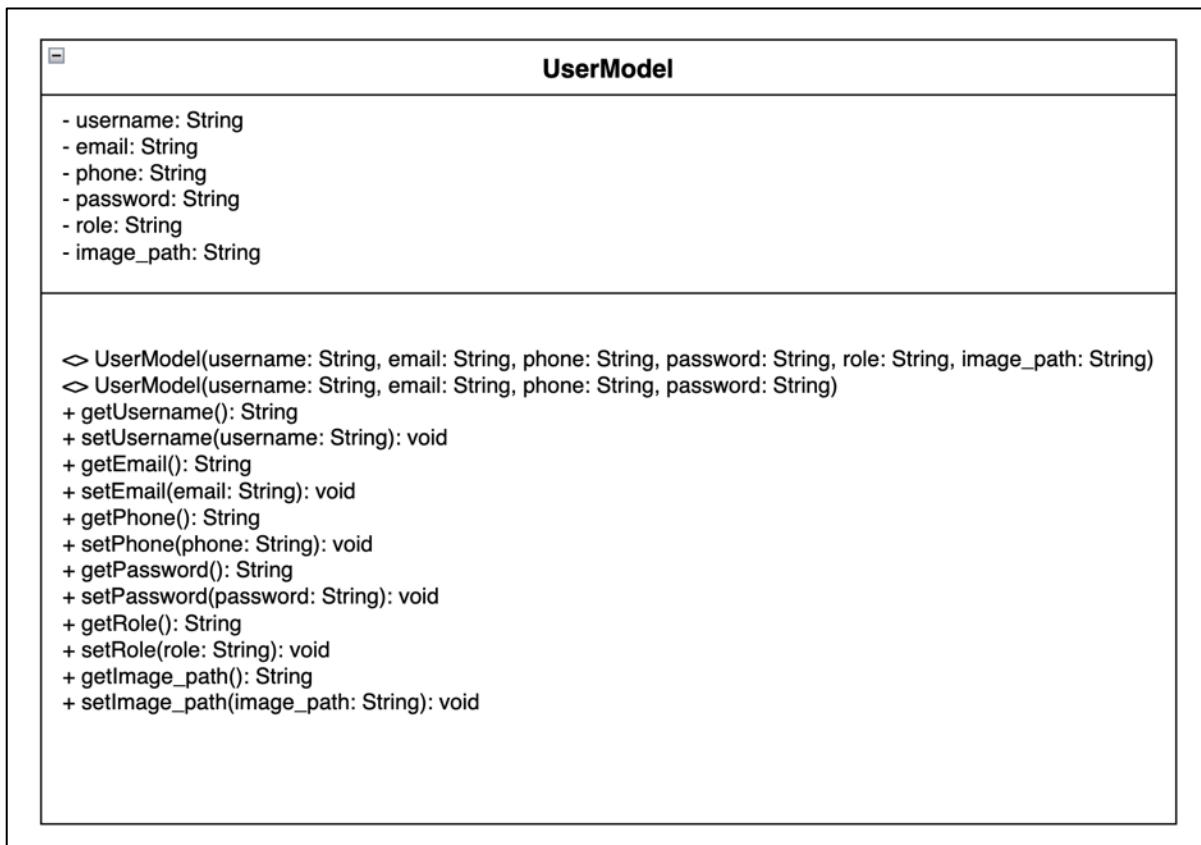
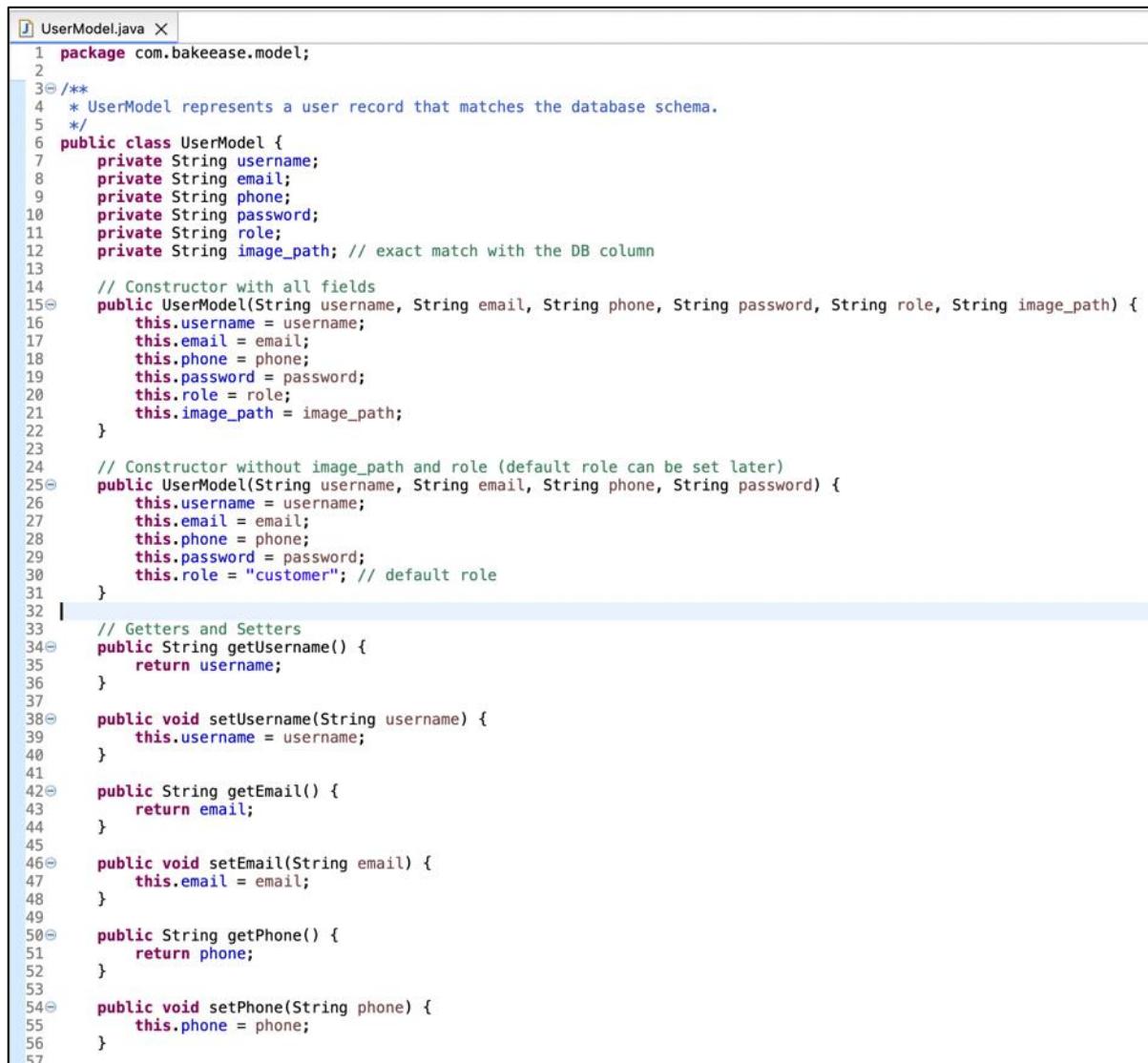


Figure 64: class diagram for user model

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.



```

1 package com.bakeease.model;
2
3 /**
4  * UserModel represents a user record that matches the database schema.
5 */
6 public class UserModel {
7     private String username;
8     private String email;
9     private String phone;
10    private String password;
11    private String role;
12    private String image_path; // exact match with the DB column
13
14    // Constructor with all fields
15    public UserModel(String username, String email, String phone, String password, String role, String image_path) {
16        this.username = username;
17        this.email = email;
18        this.phone = phone;
19        this.password = password;
20        this.role = role;
21        this.image_path = image_path;
22    }
23
24    // Constructor without image_path and role (default role can be set later)
25    public UserModel(String username, String email, String phone, String password) {
26        this.username = username;
27        this.email = email;
28        this.phone = phone;
29        this.password = password;
30        this.role = "customer"; // default role
31    }
32
33    // Getters and Setters
34    public String getUsername() {
35        return username;
36    }
37
38    public void setUsername(String username) {
39        this.username = username;
40    }
41
42    public String getEmail() {
43        return email;
44    }
45
46    public void setEmail(String email) {
47        this.email = email;
48    }
49
50    public String getPhone() {
51        return phone;
52    }
53
54    public void setPhone(String phone) {
55        this.phone = phone;
56    }
57

```

Figure 65: code for user model

4.2.13. LoginService.java Service

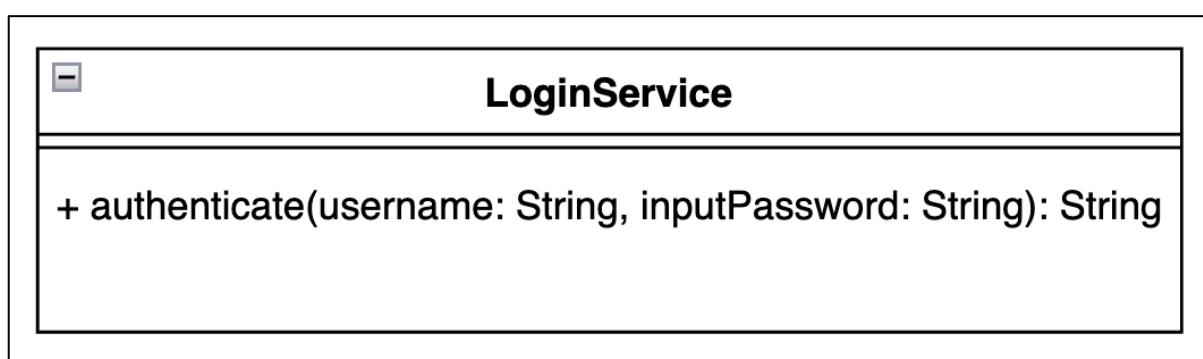
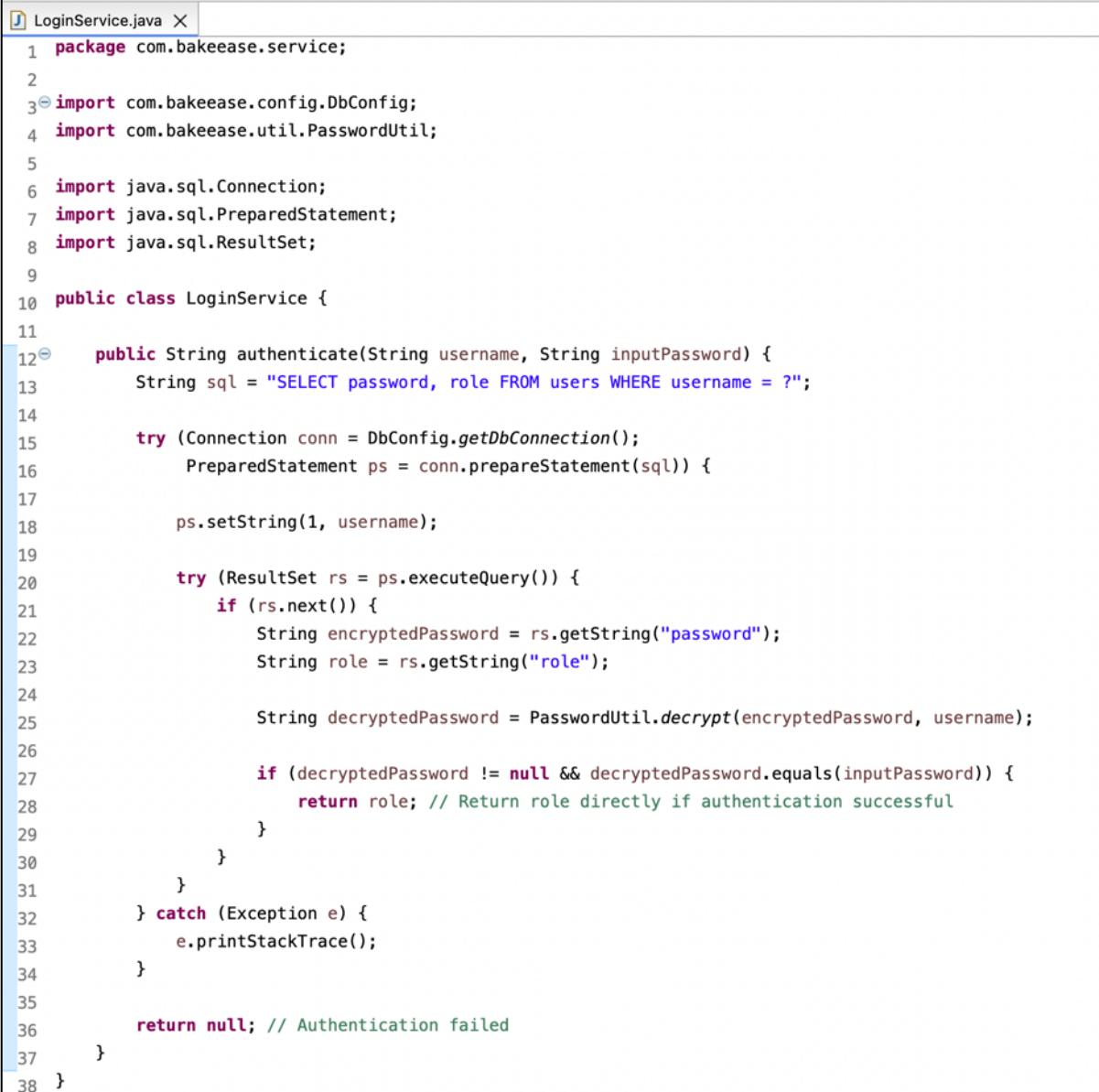


Figure 66: class diagram of login service

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.



```

1 package com.bakeease.service;
2
3 import com.bakeease.config.DbConfig;
4 import com.bakeease.util.PasswordUtil;
5
6 import java.sql.Connection;
7 import java.sql.PreparedStatement;
8 import java.sql.ResultSet;
9
10 public class LoginService {
11
12     public String authenticate(String username, String inputPassword) {
13         String sql = "SELECT password, role FROM users WHERE username = ?";
14
15         try (Connection conn = DbConfig.getDbConnection();
16              PreparedStatement ps = conn.prepareStatement(sql)) {
17
18             ps.setString(1, username);
19
20             try (ResultSet rs = ps.executeQuery()) {
21                 if (rs.next()) {
22                     String encryptedPassword = rs.getString("password");
23                     String role = rs.getString("role");
24
25                     String decryptedPassword = PasswordUtil.decrypt(encryptedPassword, username);
26
27                     if (decryptedPassword != null && decryptedPassword.equals(inputPassword)) {
28                         return role; // Return role directly if authentication successful
29                     }
30                 }
31             }
32         } catch (Exception e) {
33             e.printStackTrace();
34         }
35
36         return null; // Authentication failed
37     }
38 }
```

Figure 67: code for login service

4.2.14. **ProductService.java Service**

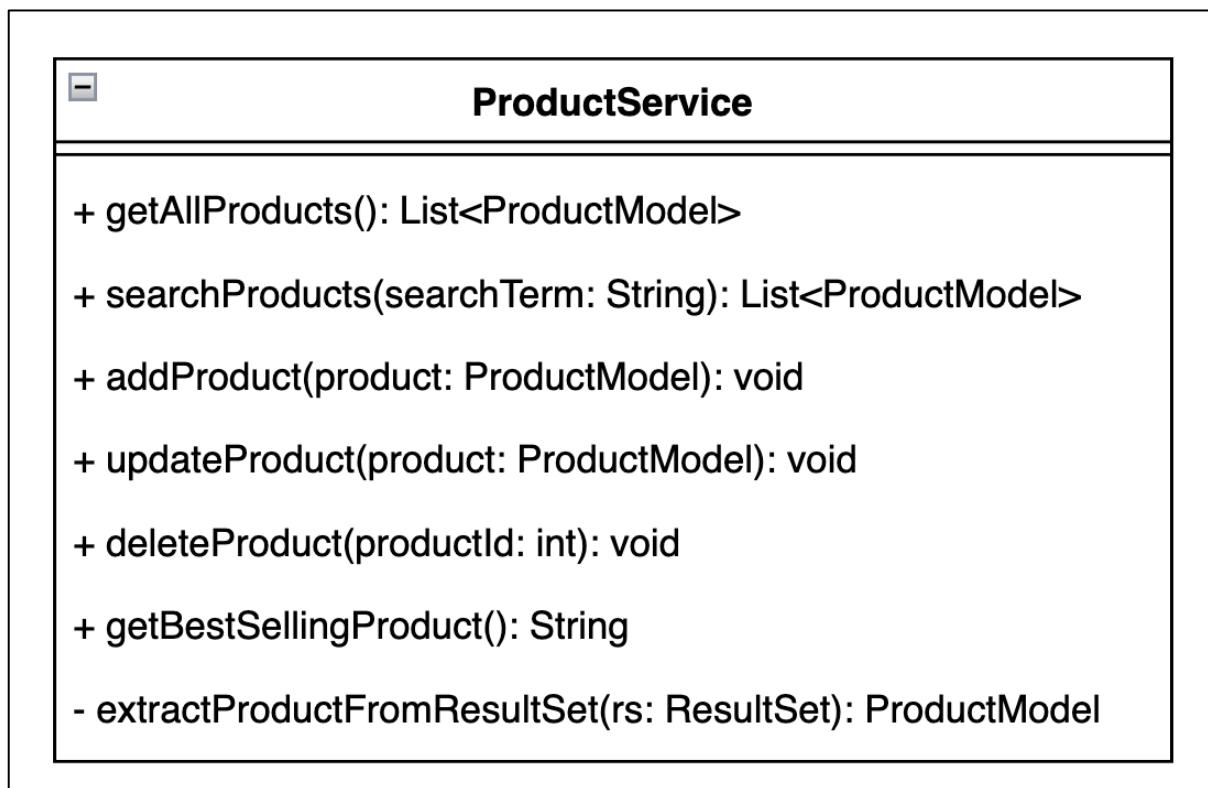
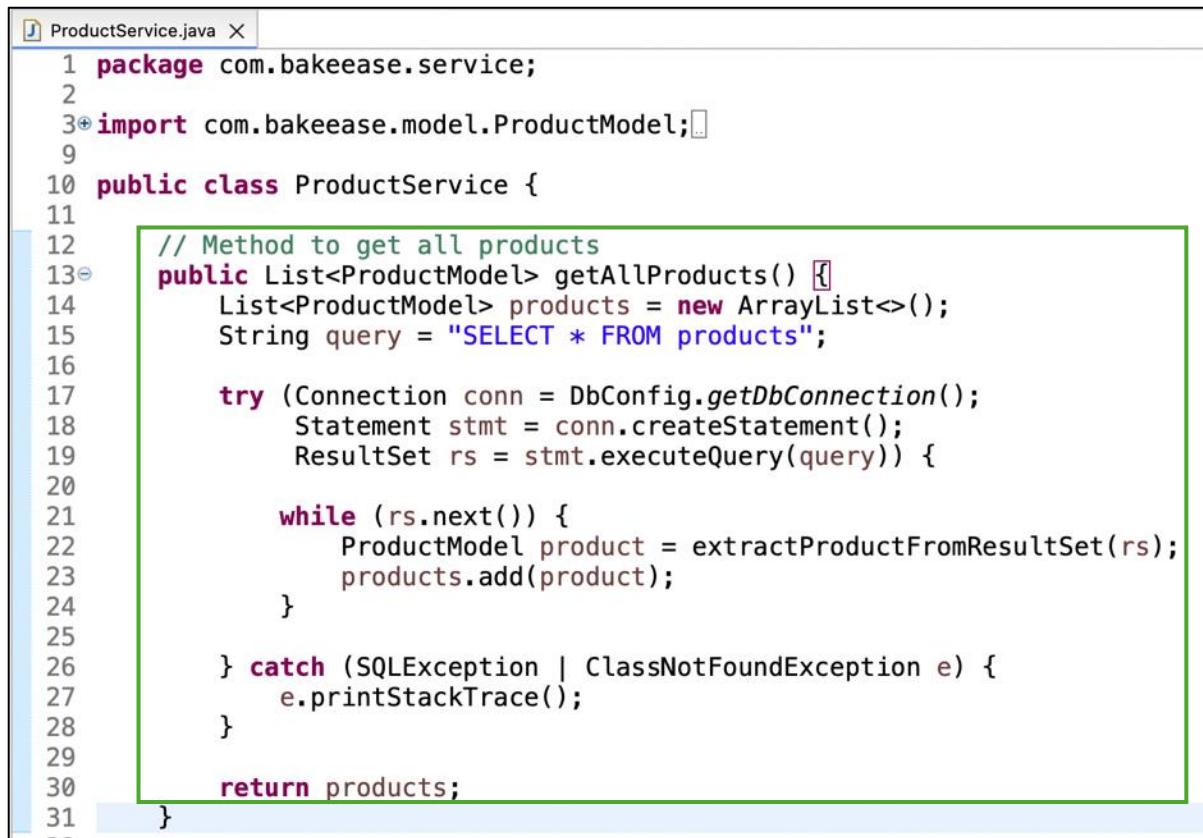


Figure 68: class diagram for product service



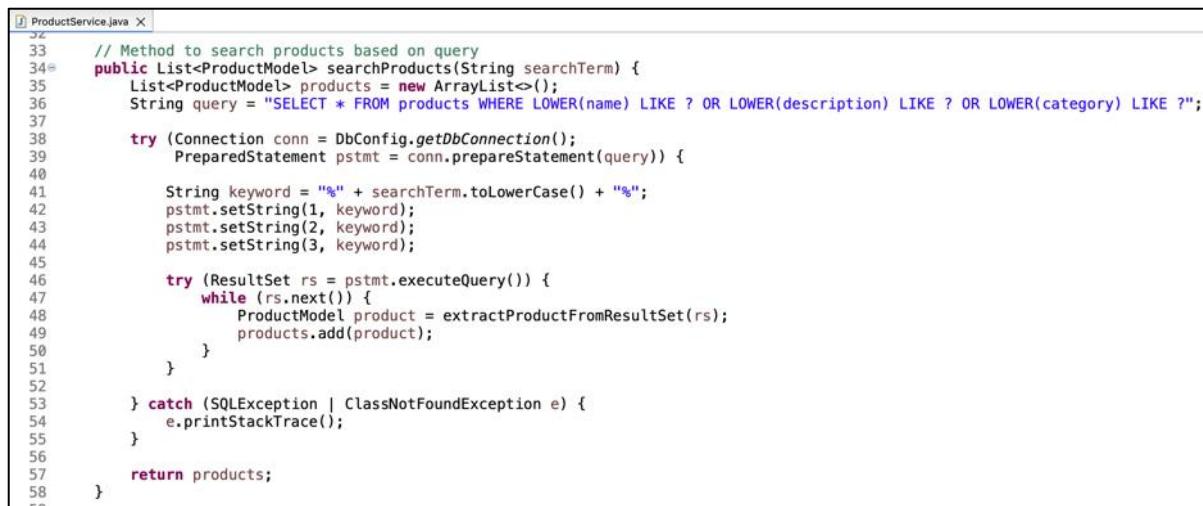
```

1 package com.bakeease.service;
2
3+ import com.bakeease.model.ProductModel;...
4
5
6 public class ProductService {
7
8     // Method to get all products
9     public List<ProductModel> getAllProducts() {
10         List<ProductModel> products = new ArrayList<>();
11         String query = "SELECT * FROM products";
12
13         try (Connection conn = DbConfig.getDbConnection();
14              Statement stmt = conn.createStatement();
15              ResultSet rs = stmt.executeQuery(query)) {
16
17             while (rs.next()) {
18                 ProductModel product = extractProductFromResultSet(rs);
19                 products.add(product);
20             }
21
22         } catch (SQLException | ClassNotFoundException e) {
23             e.printStackTrace();
24         }
25
26         return products;
27     }
28
29
30 }
31

```

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.



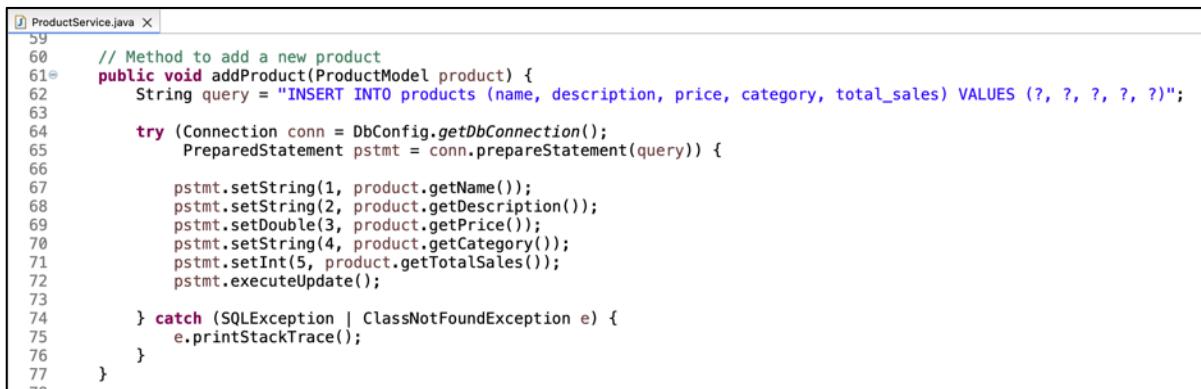
```

33     // Method to search products based on query
34     public List<ProductModel> searchProducts(String searchTerm) {
35         List<ProductModel> products = new ArrayList<>();
36         String query = "SELECT * FROM products WHERE LOWER(name) LIKE ? OR LOWER(description) LIKE ? OR LOWER(category) LIKE ?";
37
38         try (Connection conn = DbConfig.getDbConnection();
39              PreparedStatement pstmt = conn.prepareStatement(query)) {
40
41             String keyword = "%" + searchTerm.toLowerCase() + "%";
42             pstmt.setString(1, keyword);
43             pstmt.setString(2, keyword);
44             pstmt.setString(3, keyword);
45
46             try (ResultSet rs = pstmt.executeQuery()) {
47                 while (rs.next()) {
48                     ProductModel product = extractProductFromResultSet(rs);
49                     products.add(product);
50                 }
51             }
52
53         } catch (SQLException | ClassNotFoundException e) {
54             e.printStackTrace();
55         }
56
57         return products;
58     }

```

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.



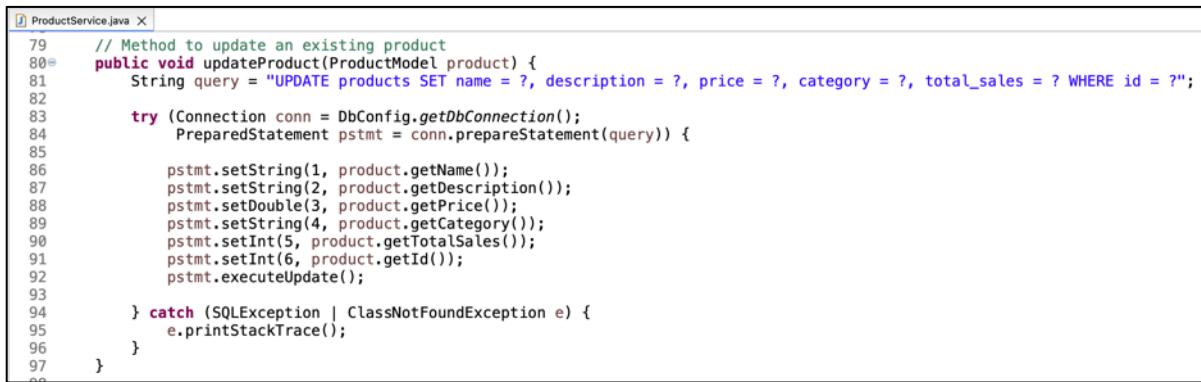
```

59
60    // Method to add a new product
61@  public void addProduct(ProductModel product) {
62      String query = "INSERT INTO products (name, description, price, category, total_sales) VALUES (?, ?, ?, ?, ?)";
63
64      try (Connection conn = DbConfig.getDbConnection();
65           PreparedStatement pstmt = conn.prepareStatement(query)) {
66
67          pstmt.setString(1, product.getName());
68          pstmt.setString(2, product.getDescription());
69          pstmt.setDouble(3, product.getPrice());
70          pstmt.setString(4, product.getCategory());
71          pstmt.setInt(5, product.getTotalSales());
72          pstmt.executeUpdate();
73
74      } catch (SQLException | ClassNotFoundException e) {
75          e.printStackTrace();
76      }
77  }

```

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.



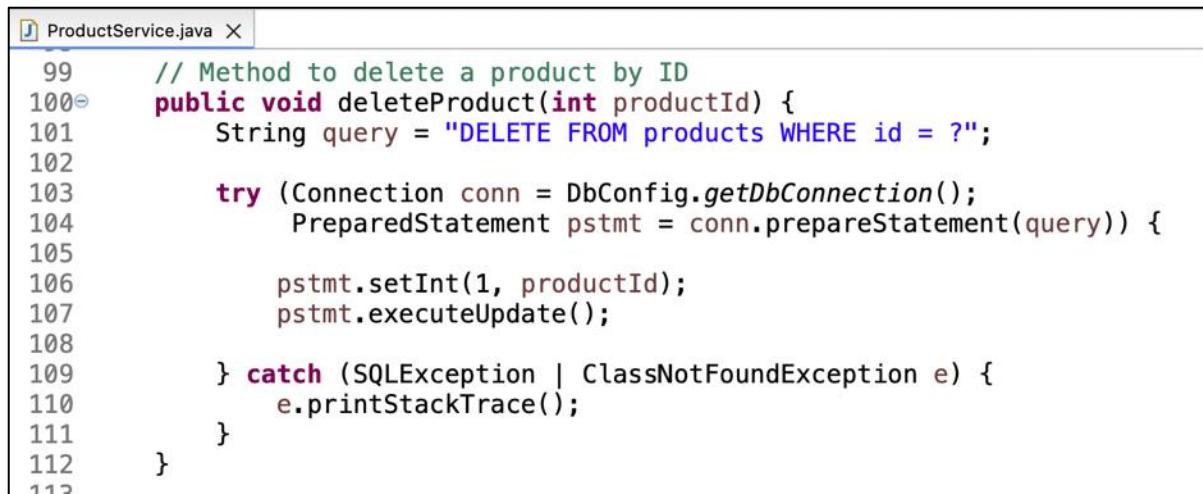
```

79
80  // Method to update an existing product
81@  public void updateProduct(ProductModel product) {
82      String query = "UPDATE products SET name = ?, description = ?, price = ?, category = ?, total_sales = ? WHERE id = ?";
83
84      try (Connection conn = DbConfig.getDbConnection();
85           PreparedStatement pstmt = conn.prepareStatement(query)) {
86
87          pstmt.setString(1, product.getName());
88          pstmt.setString(2, product.getDescription());
89          pstmt.setDouble(3, product.getPrice());
90          pstmt.setString(4, product.getCategory());
91          pstmt.setInt(5, product.getTotalSales());
92          pstmt.setInt(6, product.getId());
93          pstmt.executeUpdate();
94
95      } catch (SQLException | ClassNotFoundException e) {
96          e.printStackTrace();
97      }
98  }

```

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.



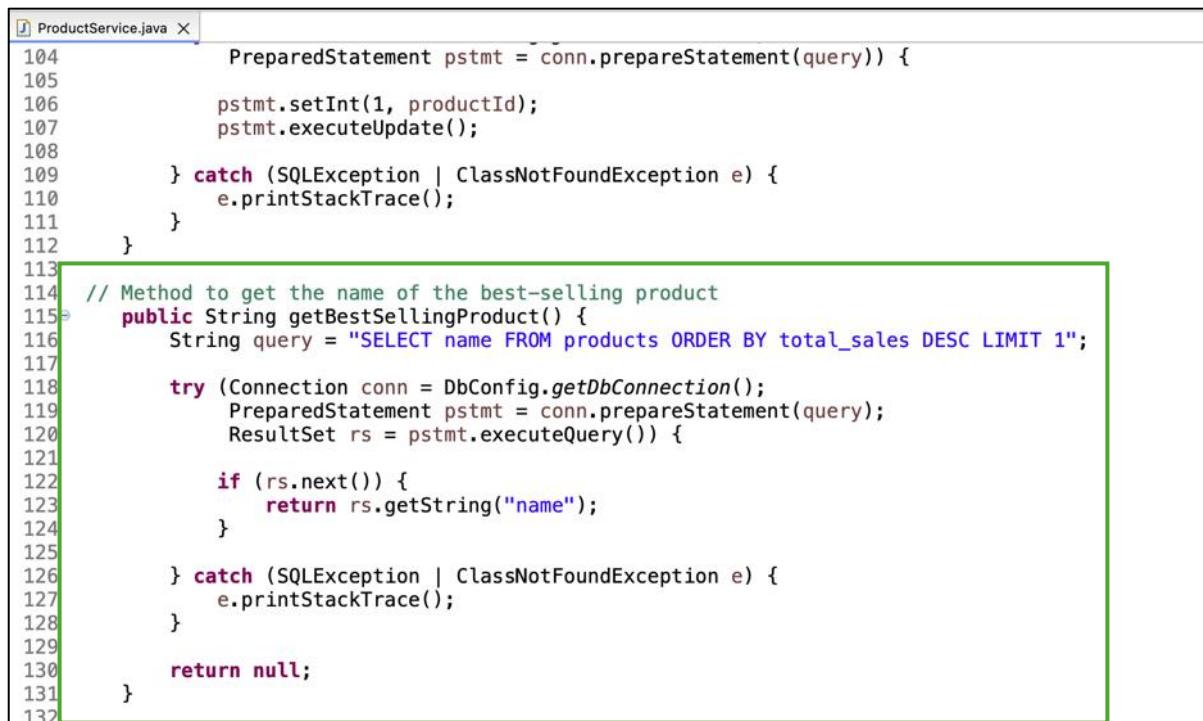
```

99     // Method to delete a product by ID
100    public void deleteProduct(int productId) {
101        String query = "DELETE FROM products WHERE id = ?";
102
103        try (Connection conn = DbConfig.getDbConnection();
104             PreparedStatement pstmt = conn.prepareStatement(query)) {
105
106            pstmt.setInt(1, productId);
107            pstmt.executeUpdate();
108
109        } catch (SQLException | ClassNotFoundException e) {
110            e.printStackTrace();
111        }
112    }
113

```

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.



```

104        PreparedStatement pstmt = conn.prepareStatement(query) {
105
106            pstmt.setInt(1, productId);
107            pstmt.executeUpdate();
108
109        } catch (SQLException | ClassNotFoundException e) {
110            e.printStackTrace();
111        }
112    }
113
114    // Method to get the name of the best-selling product
115    public String getBestSellingProduct() {
116        String query = "SELECT name FROM products ORDER BY total_sales DESC LIMIT 1";
117
118        try (Connection conn = DbConfig.getDbConnection();
119             PreparedStatement pstmt = conn.prepareStatement(query);
120             ResultSet rs = pstmt.executeQuery()) {
121
122            if (rs.next()) {
123                return rs.getString("name");
124            }
125
126        } catch (SQLException | ClassNotFoundException e) {
127            e.printStackTrace();
128        }
129
130        return null;
131    }
132

```

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

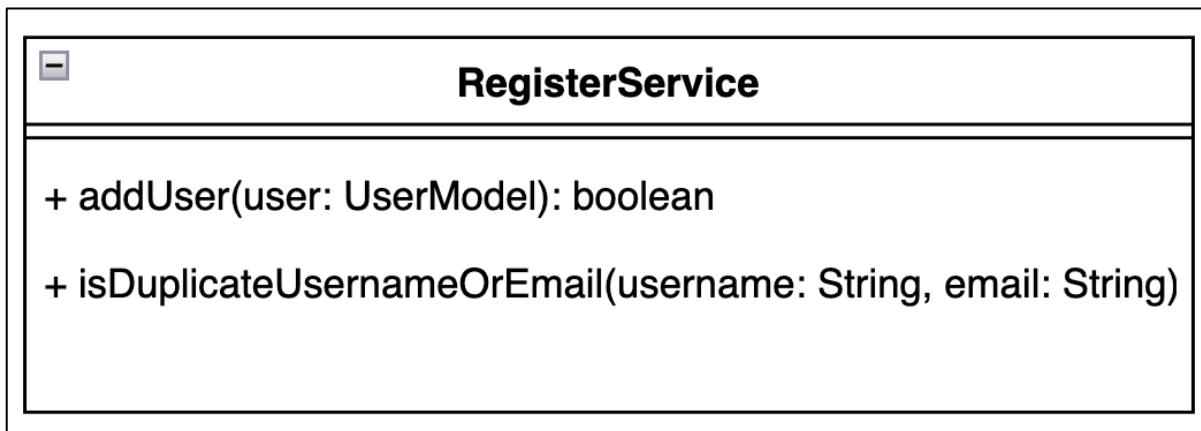


Figure 75: class diagram of register service

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

```

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

The code shows the implementation of the 'addUser' method. It imports the 'UserModel' class and defines the method signature. It constructs an SQL insert statement and uses a try-with-resources block to execute it via a prepared statement. The method returns a boolean indicating if the insertion was successful. A green box highlights the entire method body.

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

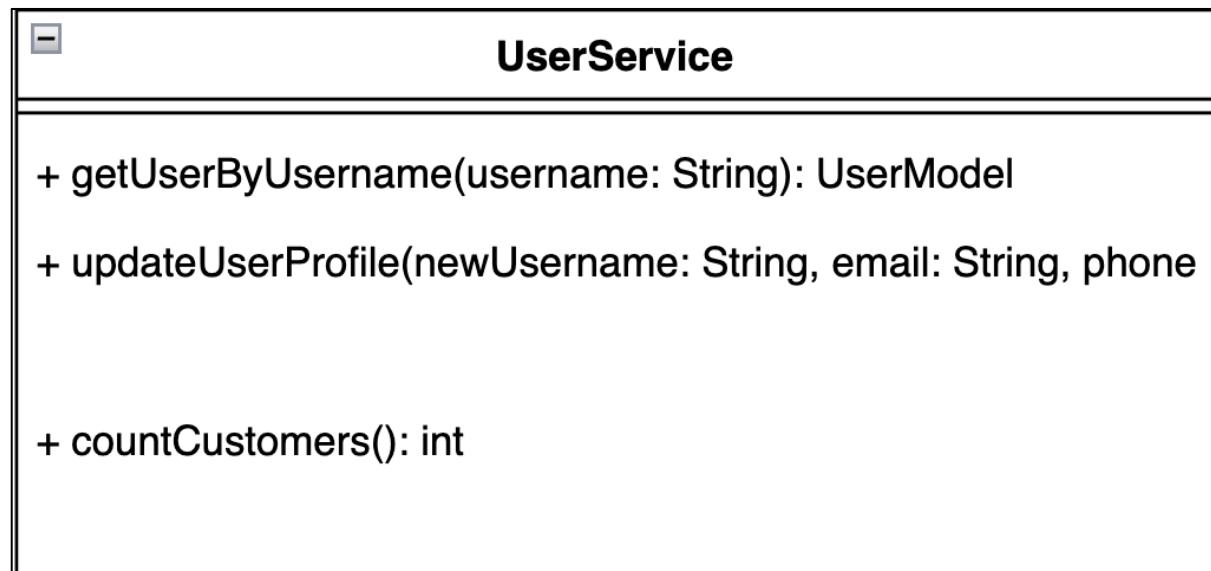
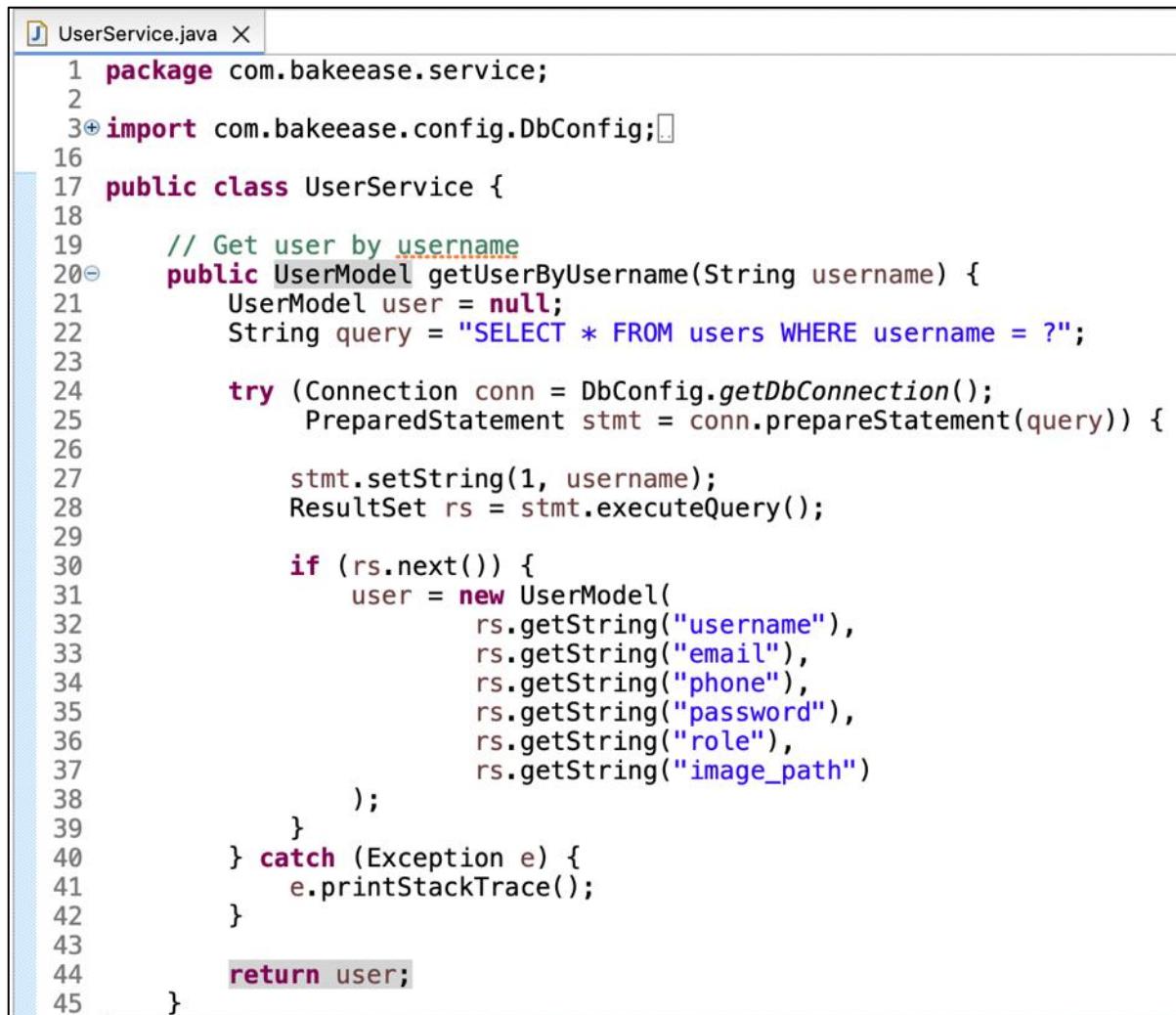


Figure 78: class diagram of user service

This class contains 3 methods which are further discussed below.



```

1 package com.bakeease.service;
2
3+import com.bakeease.config.DbConfig;[]
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.



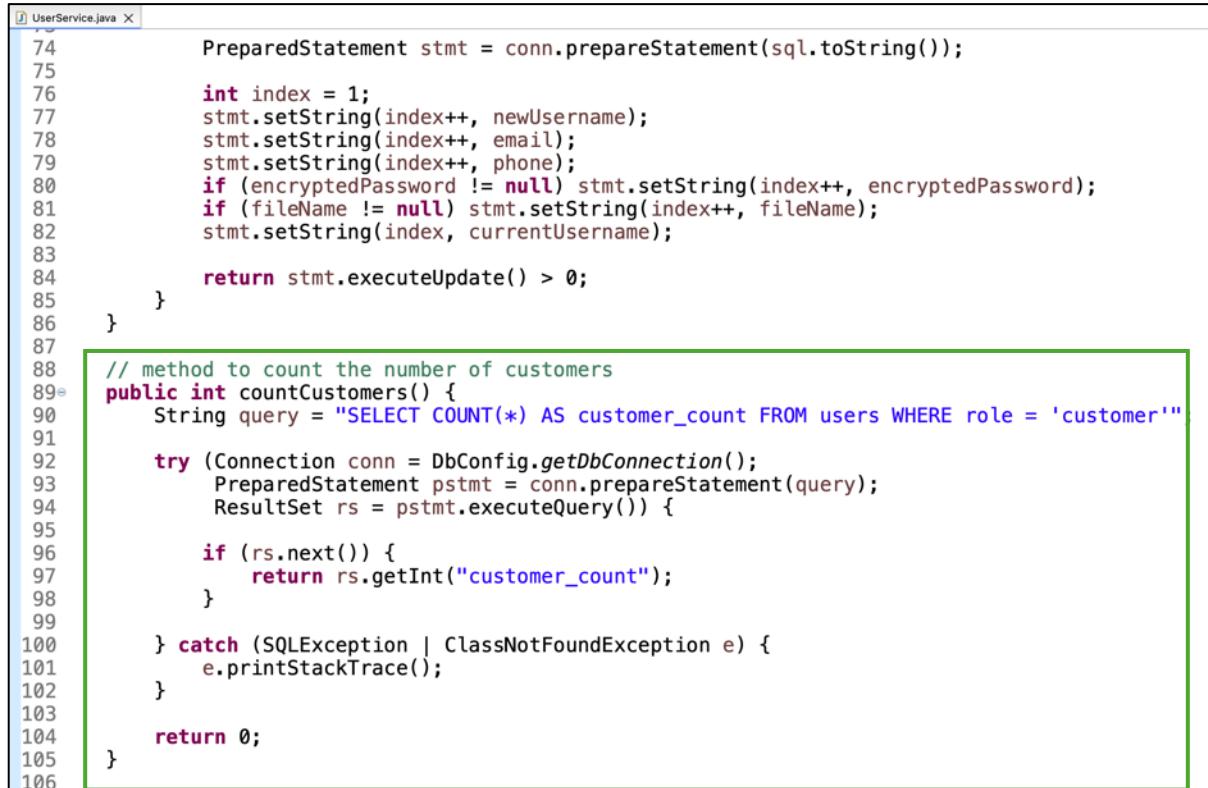
```

46     // Update user profile
47     public boolean updateUserProfile(String newUsername, String email, String phone, String password, Part imagePart, String currentUsername, String uploadPath)
48     throws SQLException, IOException, ClassNotFoundException {
49
50         String encryptedPassword = null;
51         if (password != null && !password.trim().isEmpty()) {
52             encryptedPassword = PasswordUtil.encrypt(newUsername, password);
53         }
54
55         String fileName = null;
56         if (imagePart != null && imagePart.getSize() > 0) {
57             fileName = Paths.get(imagePart.getSubmittedFileName()).getFileName().toString();
58             File uploadsDir = new File(uploadPath);
59             if (!uploadsDir.exists()) uploadsDir.mkdirs();
60
61             File file = new File(uploadsDir, fileName);
62             try (InputStream input = imagePart.getInputStream()) {
63                 Files.copy(input, file.toPath(), StandardCopyOption.REPLACE_EXISTING);
64             }
65         }
66
67         try (Connection conn = DbConfig.getDbConnection()) {
68             StringBuilder sql = new StringBuilder("UPDATE users SET username = ?, email = ?, phone = ?");
69             if (encryptedPassword != null) sql.append(", password = ?");
70             if (fileName != null) sql.append(", image_path = ?");
71             sql.append(" WHERE username = ?");
72
73             PreparedStatement stmt = conn.prepareStatement(sql.toString());
74
75             int index = 1;
76             stmt.setString(index++, newUsername);
77             stmt.setString(index++, email);
78             stmt.setString(index++, phone);
79             if (encryptedPassword != null) stmt.setString(index++, encryptedPassword);
80             if (fileName != null) stmt.setString(index++, fileName);
81             stmt.setString(index, currentUsername);
82
83             return stmt.executeUpdate() > 0;
84         }
85     }
86 }

```

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.



```

74         PreparedStatement stmt = conn.prepareStatement(sql.toString());
75
76         int index = 1;
77         stmt.setString(index++, newUsername);
78         stmt.setString(index++, email);
79         stmt.setString(index++, phone);
80         if (encryptedPassword != null) stmt.setString(index++, encryptedPassword);
81         if (fileName != null) stmt.setString(index++, fileName);
82         stmt.setString(index, currentUsername);
83
84         return stmt.executeUpdate() > 0;
85     }
86 }
87
88 // method to count the number of customers
89 public int countCustomers() {
90     String query = "SELECT COUNT(*) AS customer_count FROM users WHERE role = 'customer'";
91
92     try (Connection conn = DbConfig.getDbConnection();
93          PreparedStatement pstmt = conn.prepareStatement(query);
94          ResultSet rs = pstmt.executeQuery()) {
95
96         if (rs.next()) {
97             return rs.getInt("customer_count");
98         }
99
100    } catch (SQLException | ClassNotFoundException e) {
101        e.printStackTrace();
102    }
103
104    return 0;
105}
106

```

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**

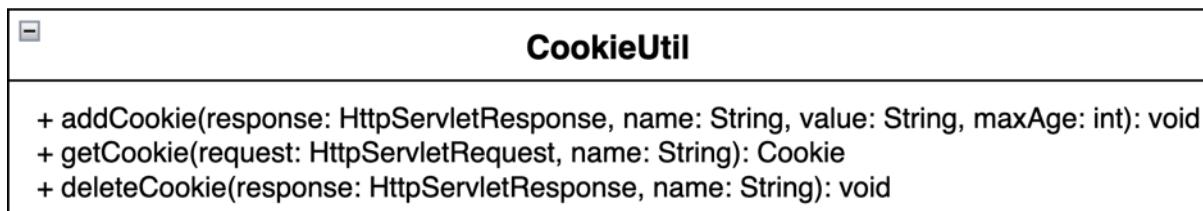


Figure 82: class diagram for cookie util

This class contains 3 methods which are discussed below.

```

CookieUtil.java
1 package com.bakeease.util;
2
3 import jakarta.servlet.http.Cookie;
4
5 public class CookieUtil {
6
7     public static void addCookie(HttpServletRequest response, String name, String value, int maxAge) {
8         Cookie cookie = new Cookie(name, value);
9         cookie.setMaxAge(maxAge);
10        cookie.setPath("/");
11        response.addCookie(cookie);
12    }
13}

```

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.

```

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(HttpServletRequest 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

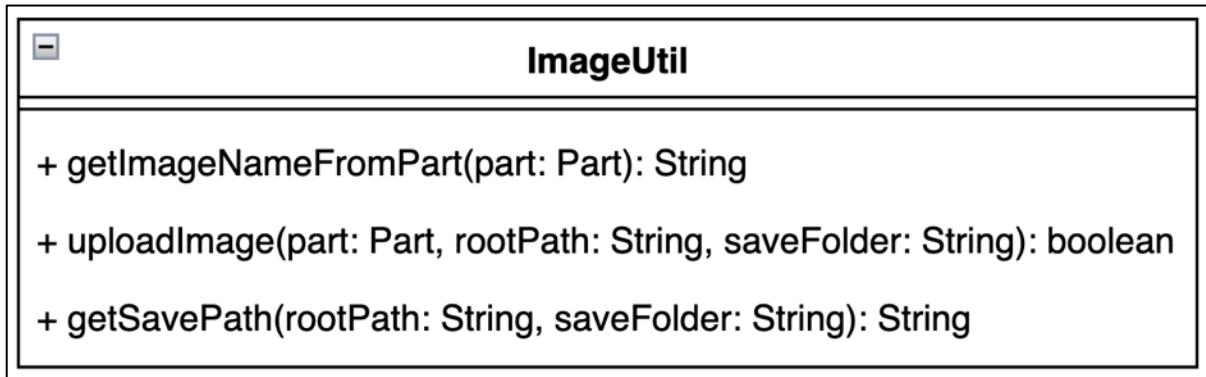
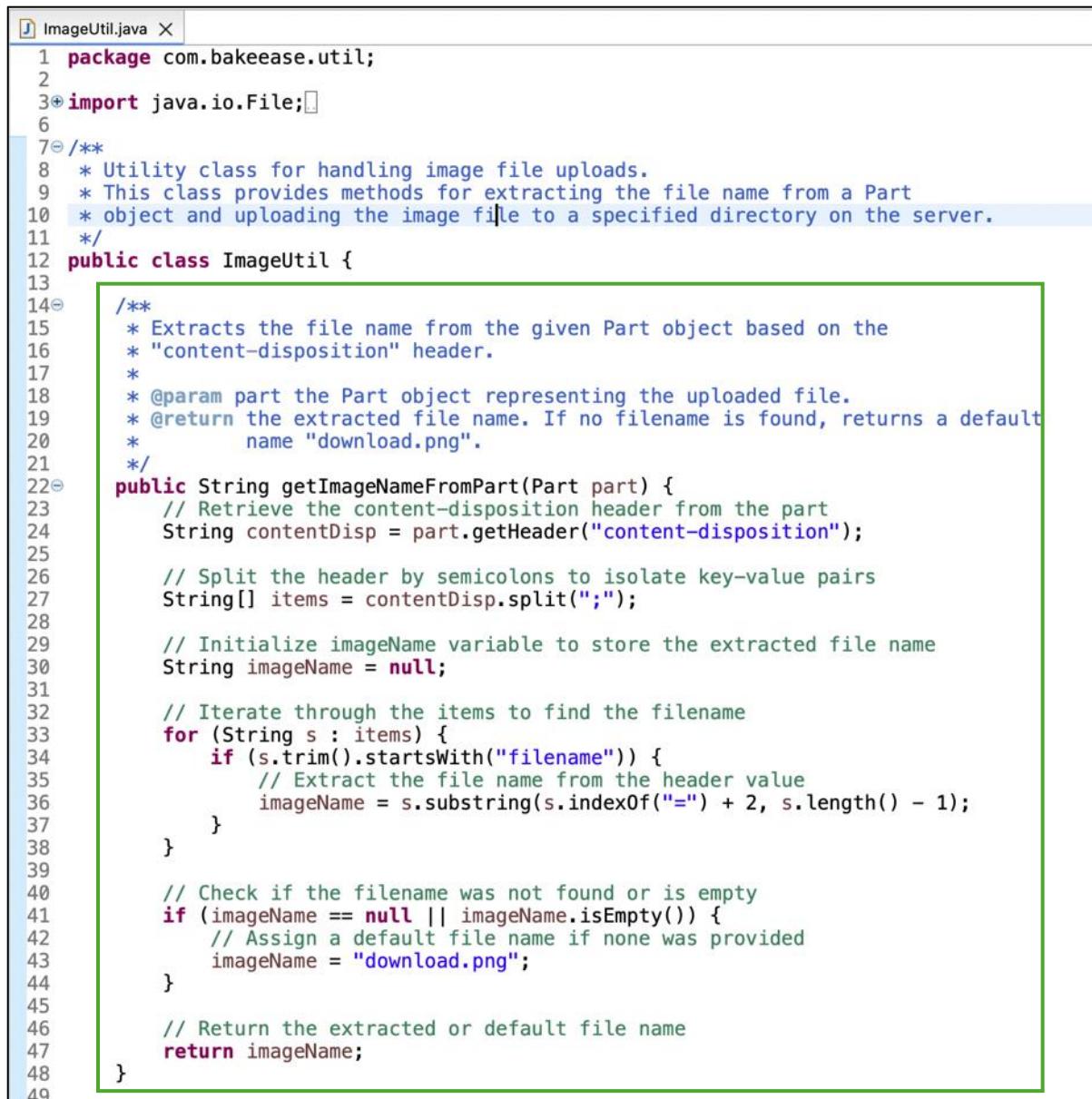


Figure 86: class diagram for image util

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



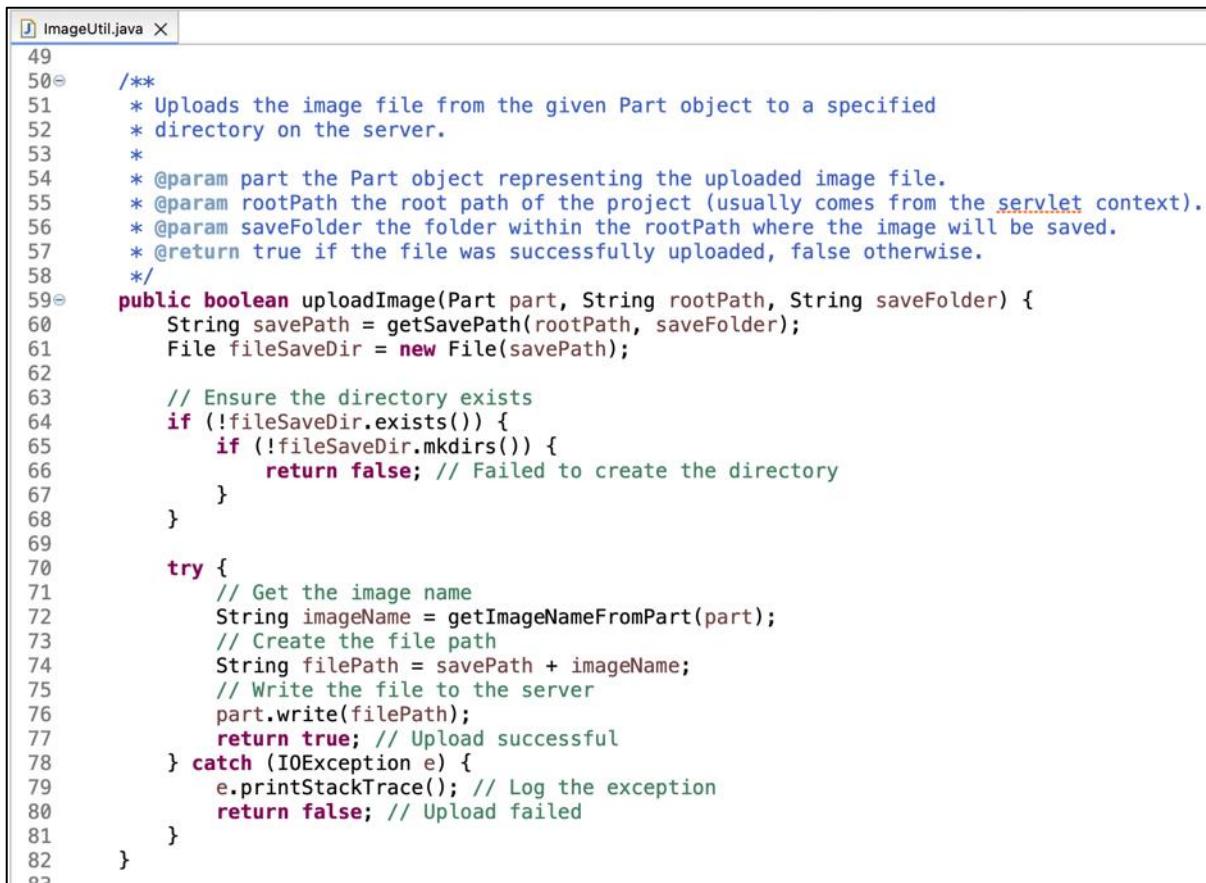
```

1 package com.bakeease.util;
2
3+import java.io.File;[]
4
5 /**
6  * Utility class for handling image file uploads.
7  * This class provides methods for extracting the file name from a Part
8  * object and uploading the image file to a specified directory on the server.
9  */
10
11
12 public class ImageUtil {
13
14 /**
15  * Extracts the file name from the given Part object based on the
16  * "content-disposition" header.
17  *
18  * @param part the Part object representing the uploaded file.
19  * @return the extracted file name. If no filename is found, returns a default
20  *         name "download.png".
21  */
22
23     public String getImageNameFromPart(Part part) {
24         // Retrieve the content-disposition header from the part
25         String contentDisp = part.getHeader("content-disposition");
26
27         // Split the header by semicolons to isolate key-value pairs
28         String[] items = contentDisp.split(";");
29
30         // Initialize imageName variable to store the extracted file name
31         String imageName = null;
32
33         // Iterate through the items to find the filename
34         for (String s : items) {
35             if (s.trim().startsWith("filename")) {
36                 // Extract the file name from the header value
37                 imageName = s.substring(s.indexOf("=") + 2, s.length() - 1);
38             }
39
40             // Check if the filename was not found or is empty
41             if (imageName == null || imageName.isEmpty()) {
42                 // Assign a default file name if none was provided
43                 imageName = "download.png";
44             }
45
46             // Return the extracted or default file name
47             return imageName;
48         }
49     }

```

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.



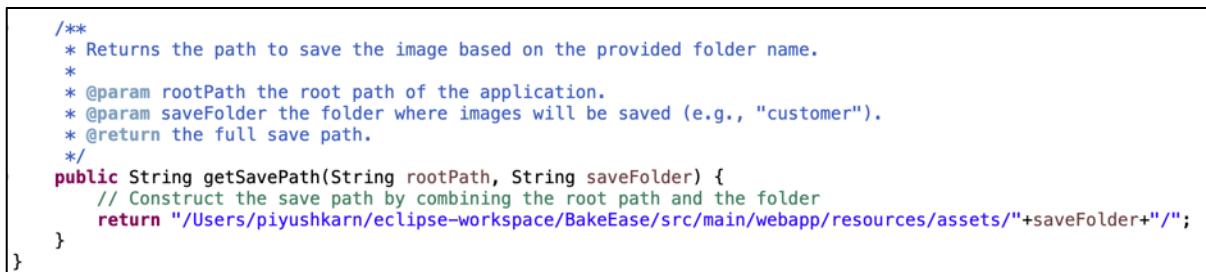
```

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*      */
59public 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}
83

```

Figure 88: code for upload image

This method is 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

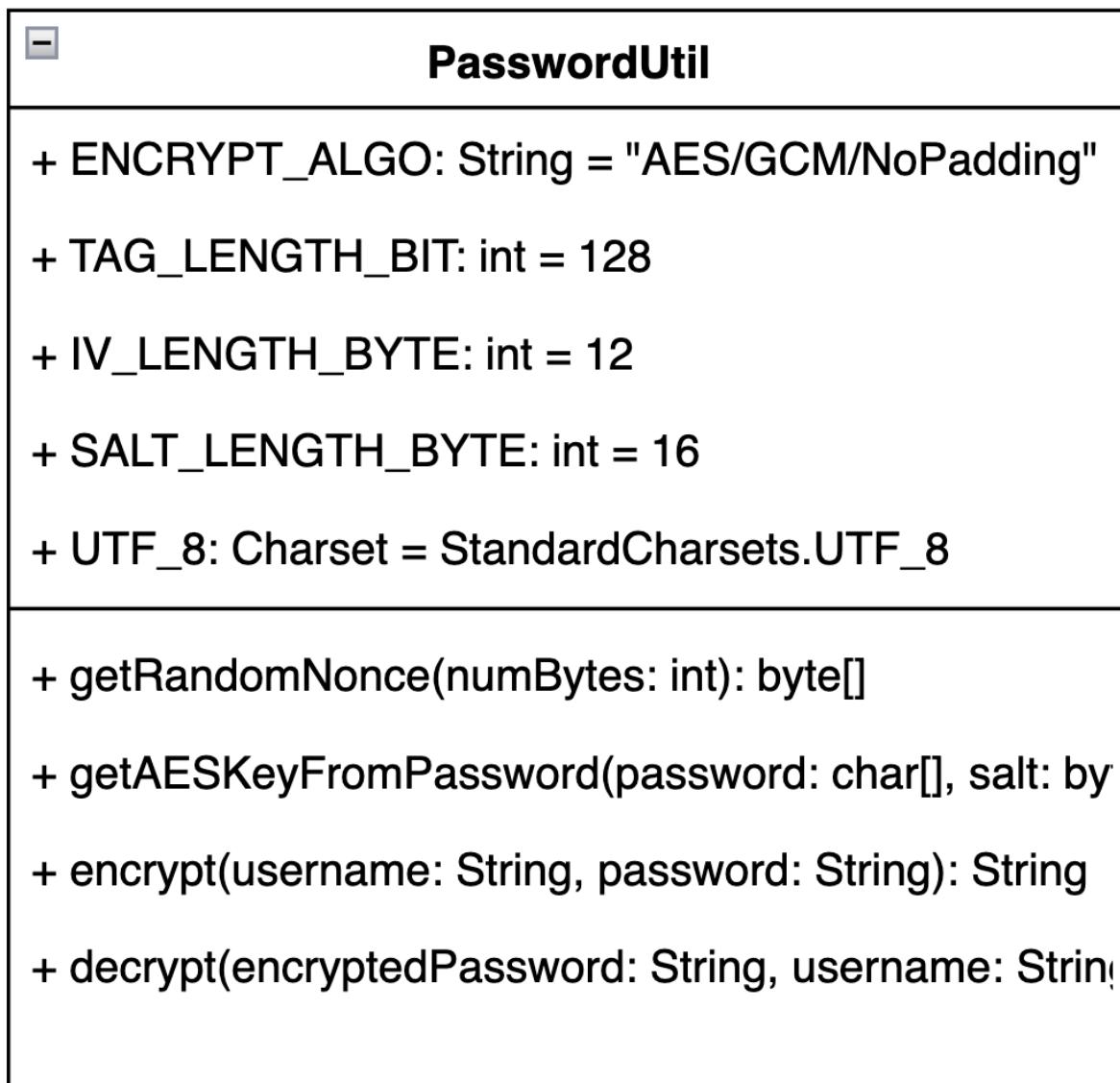


Figure 90: class diagram for password util

This class contains 4 major methods which are discussed below.



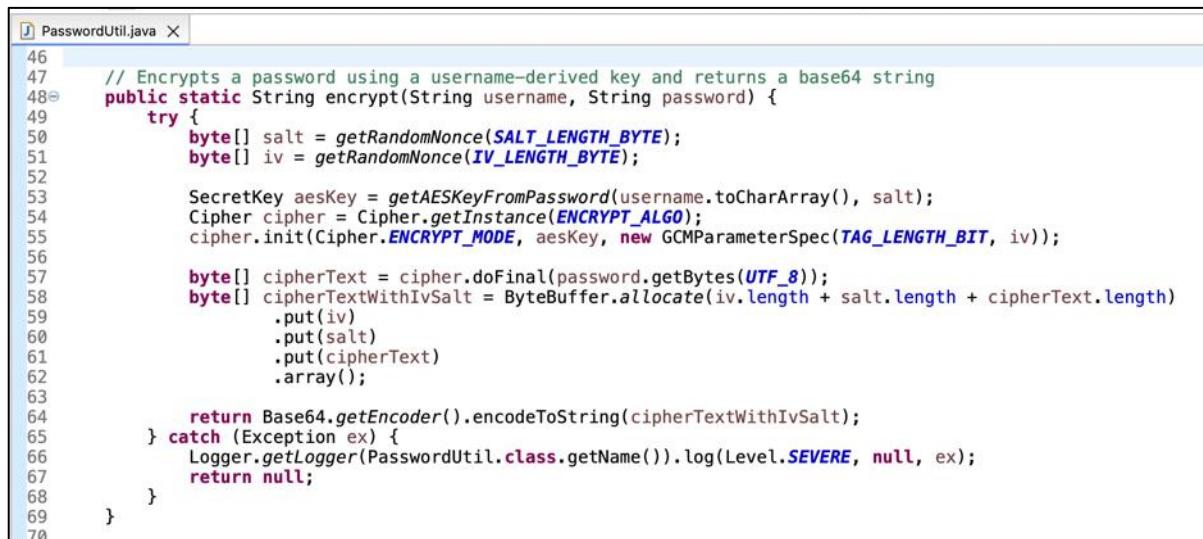
```

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.



```

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

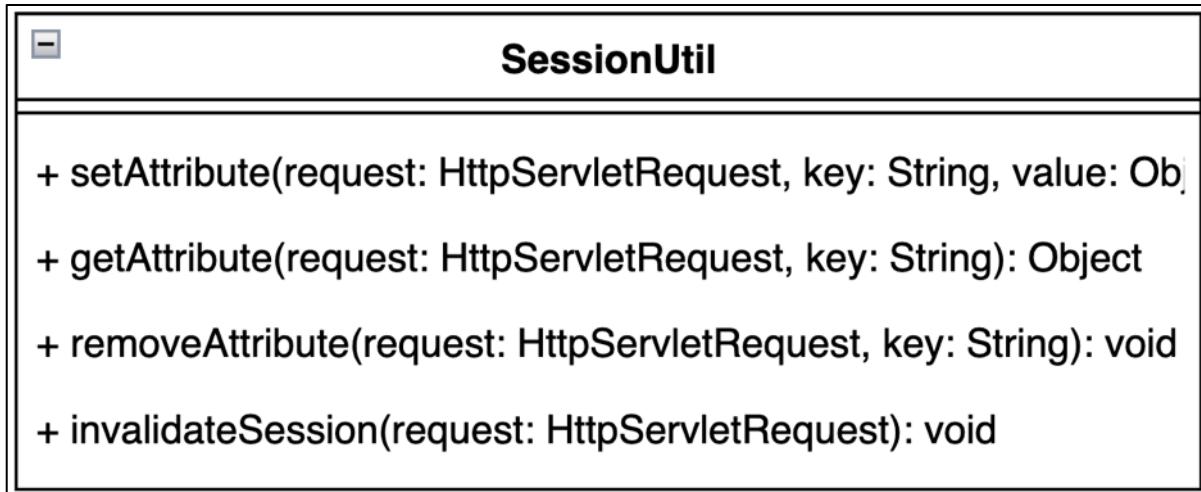


Figure 94: class diagram for session util

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

```
J SessionUtil.java X
1 package com.bakeease.util;
2
3+ import jakarta.servlet.http.HttpServletRequest;
4
5
6 public class SessionUtil {
7
8+     public static void setAttribute(HttpServletRequest request, String key, Object value) {
9         HttpSession session = request.getSession();
10        session.setAttribute(key, value);
11    }
12}
```

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.

```
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.

```
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.

```
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

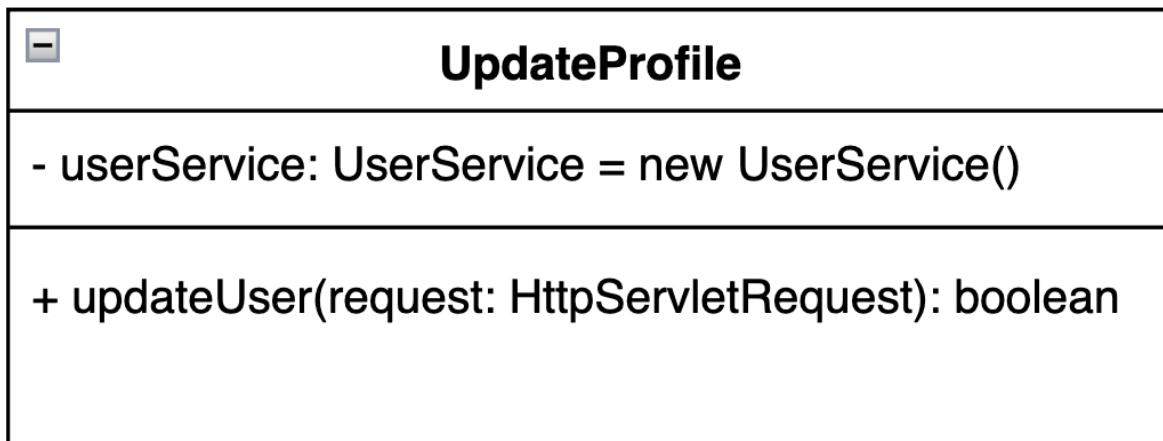


Figure 99: class diagram of update profile

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.

```

1 package com.bakeease.util;
2
3 import com.bakeease.service.UserService;
4
5 public class UpdateProfile {
6     private static final UserService userService = new UserService();
7
8     public static boolean updateUser(HttpServletRequest request) throws Exception {
9         String currentUsername = (String) SessionUtil.getAttribute(request, "username");
10
11         if (currentUsername == null) {
12             return false;
13         }
14
15         String newUsername = request.getParameter("username");
16         String email = request.getParameter("email");
17         String phone = request.getParameter("phone");
18         String password = request.getParameter("password");
19         Part imagePart = request.getPart("profileImage");
20
21         String uploadPath = request.getServletContext().getRealPath("/") + File.separator + "resources" +
22             File.separator + "assets" + File.separator + "customer-images";
23
24         boolean success = userService.updateUserProfile(newUsername, email, phone, password, imagePart, currentUsername, uploadPath);
25
26         if (success) {
27             SessionUtil.setAttribute(request, "username", newUsername);
28         }
29
30         return success;
31     }
32 }

```

Figure 100: code for update profile

4.2.22. ValidationUtil.java Util

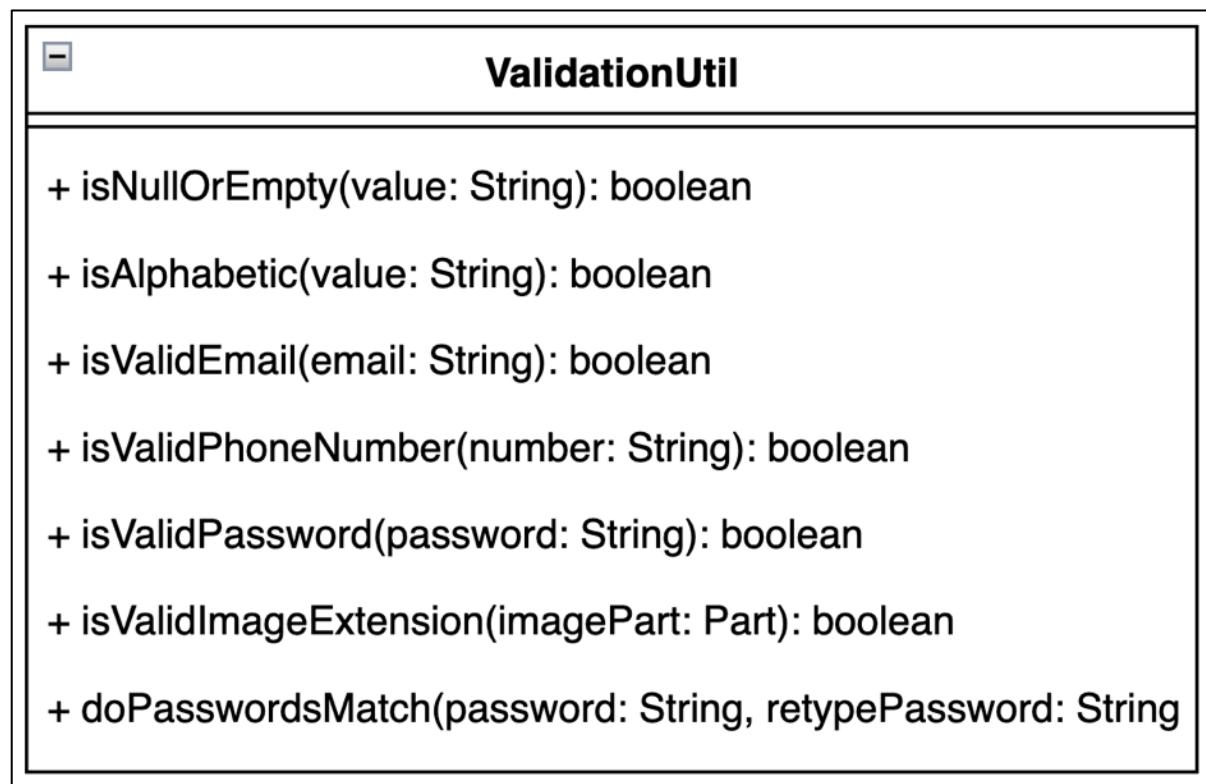


Figure 101: class diagram of validation util

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 isAlphanumericString

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");
}

// A Validator is passed and returns a success or fail message
```

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 if 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**

The screenshot shows a 'Create Account' form with several input fields and their validation messages:

- Username:** The field contains an empty value. The validation message is "Username is required."
- Email:** The field contains an empty value. The validation message is "Email is required."
- Password:** The field contains an empty value. The validation message is "Password is required."
- Confirm Password:** The field contains an empty value. The validation message is "Confirm Password is required."
- Phone Number:** The field contains an empty value. The validation message is "Phone number is required."
- Profile Image:** The field shows "Choose File" and "No file chosen". The validation message is "Image field is empty. Please upload an image."

At the bottom, there is a "Register" button and a link "Already have an account? [Login here](#)".

Figure 109: empty fields validation in register page

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

- **Incorrect details validation**

The screenshot shows a 'Create Account' form with validation messages for incorrect details:

- Username:** The field contains "783465394". The validation message is "Username must start with a letter and be 3-20 characters long."
- Email:** The field contains "sbkuhfg@gmail.com".
- Password:** The field contains an empty value. The validation message is "Password and Confirm Password do not match."
- Confirm Password:** The field contains an empty value.
- Phone Number:** The field contains "78678678786". The validation message is "Phone number must start with '98' and be 10 digits."
- Profile Image:** The field shows "Choose File" and "No file chosen". The validation message is "Image field is empty. Please upload an image."

At the bottom, there is a "Register" button, a link "Already have an account? [Login here](#)", and a "Back to Home" button.

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**

Create Account

Username

Username must start with a letter and be 3-20 characters long.

Email

Password

Confirm Password

Password and Confirm Password do not match.

Phone Number

Phone number must start with '98' and be 10 digits.

Profile Image

Image field is empty. Please upload an image.

Register

Already have an account? [Login here](#)

Back to Home

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

Server: localhost - Database: BakeEaseDb - Table: users

Browse Structure SQL Search Insert Export Import Privileges Operations Tracking Triggers

Showing rows 0 - 0 (1 total, Query took 0.0001 seconds.)

SELECT * FROM `users`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Restore column order | Number of rows: 25 Filter rows: Search this table

Extra options

	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	id	username	email	password	image_path	phone	role
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	piyush	piyushkarm76@gmail.com	gQZT5ojQd2POJV+J EhI6U4iRM0izZ1l zOj5dpm4J785UWsYZ...	piyush.png	9825914530	customer

With selected:

Show all | Restore column order | Number of rows: 25 Filter rows: Search this table

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

The image shows a login form titled "Welcome Back". At the top, there is a red error message box containing the text "Invalid username or password.". Below this is a "Username" input field containing the text "piyush". Underneath the input field is a "Password" input field containing several dots. To the right of the password field is a small circular icon with a purple dot. Below the input fields is a "Login" button in a brown rectangle. At the bottom of the form, there is a link "Don't have an account? [Register here](#)" and a "Back to Home" button.

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**

The image shows a login form titled "Welcome Back". The "Username" input field contains the text "piyush". The "Password" input field contains several dots. To the right of the password field is a small circular icon with a purple dot. Below the input fields is a "Login" button in a brown rectangle. At the bottom of the form, there is a link "Don't have an account? [Register here](#)" and a "Back to Home" button.

Figure 115: before entering correct details in the web page

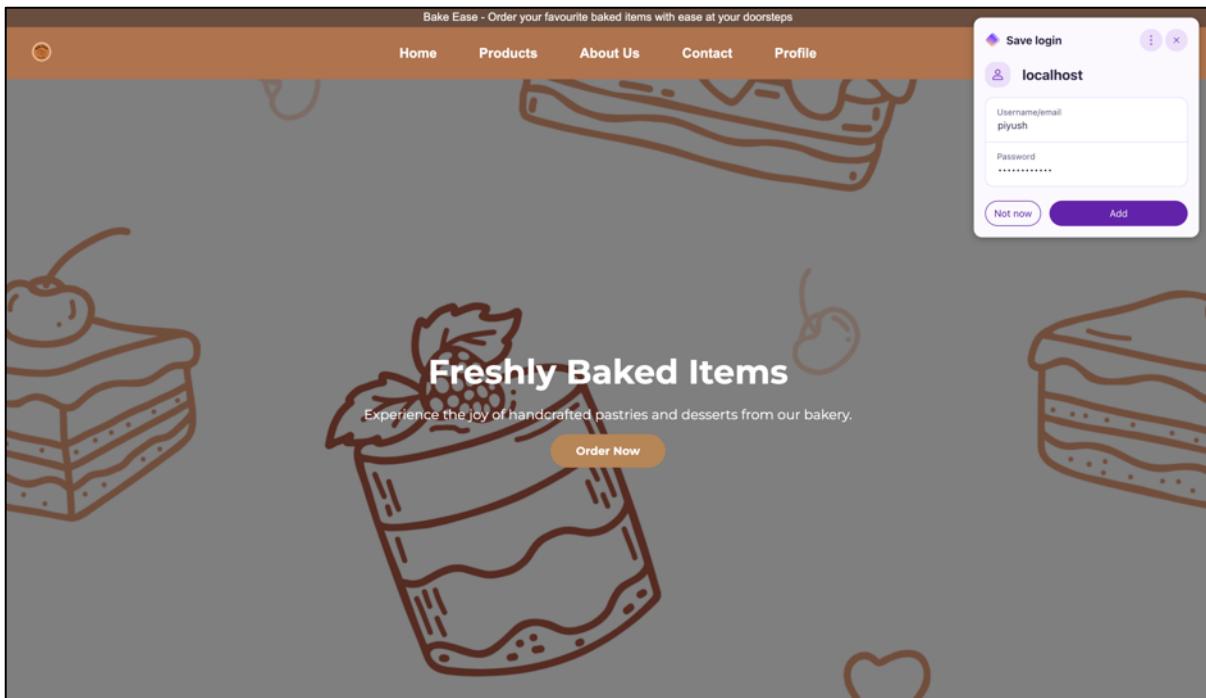


Figure 116: after entering the correct details in the web page

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.

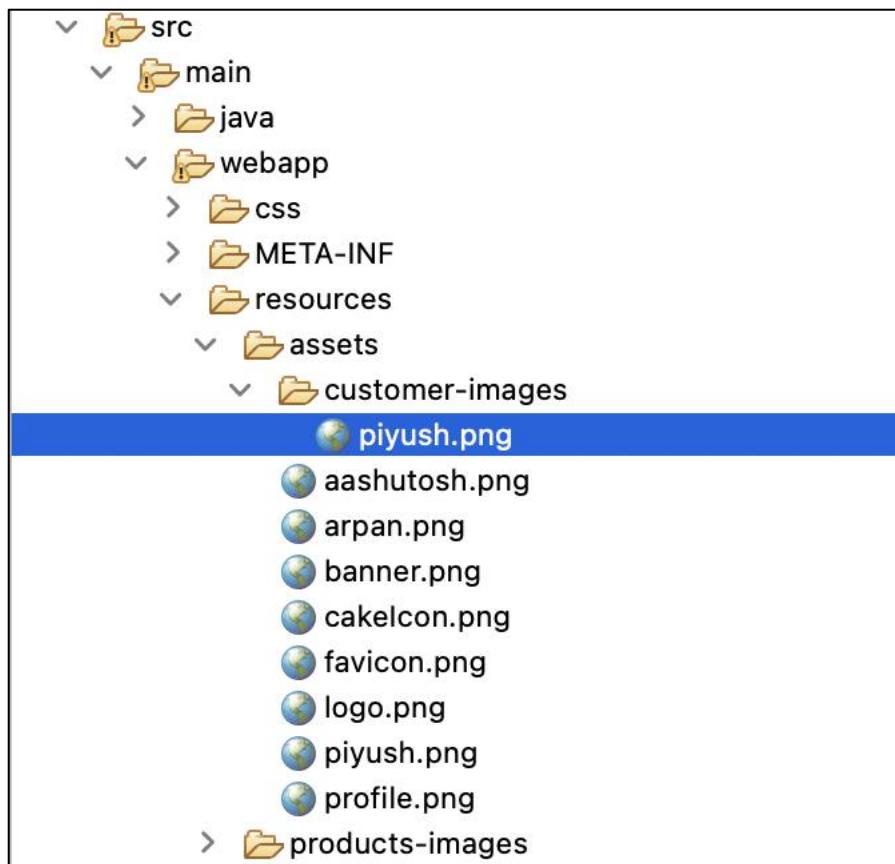


Figure 117: image of the logged in user in the system

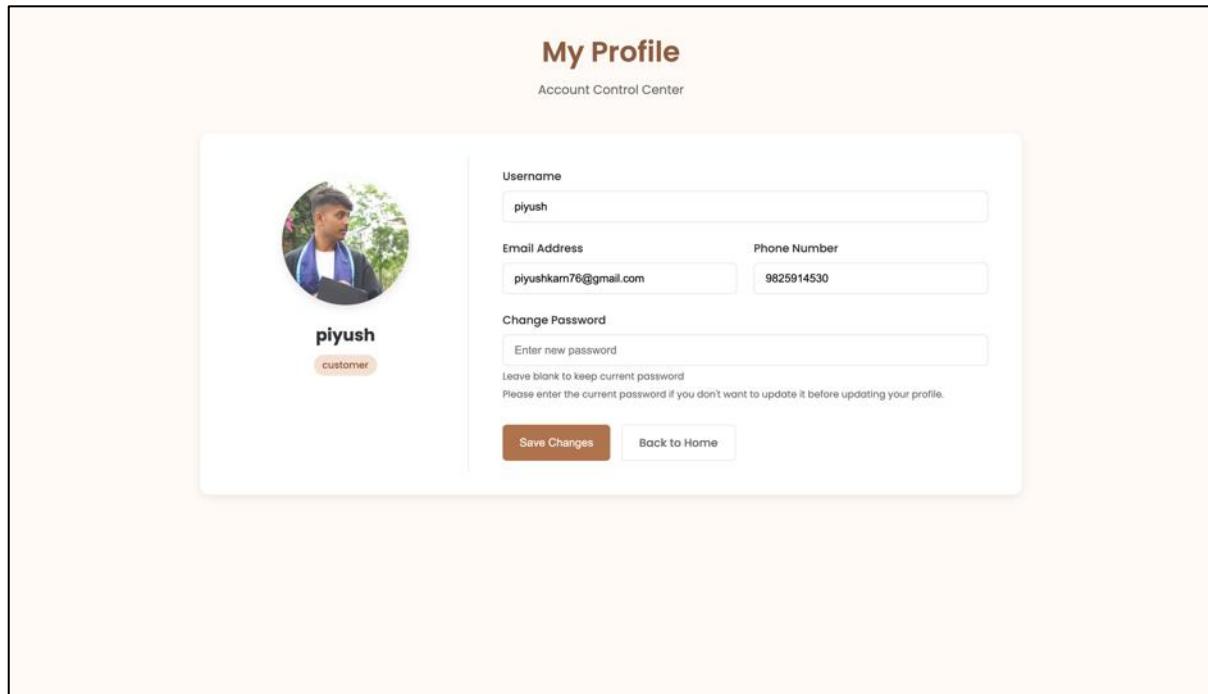


Figure 118: user profile of the currently logged in user

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

The screenshot shows a web application interface titled "My Profile" under "Account Control Center". On the left, there is a circular profile picture of a person labeled "piyush" and "customer". To the right, there are several input fields: "Username" (piyush), "Email Address" (piyushkarn76@gmail.com), "Phone Number" (9825914530), and a "Change Password" section with a note about leaving it blank to keep the current password. At the bottom are two buttons: "Save Changes" (in orange) and "Back to Home".

Figure 119: before updating user profile page

	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input checked="" type="checkbox"/> Delete	id	username	email	password	image_path	phone	role
				1	piyush	piyushkarn76@gmail.com	gQZT5ojQd2POJV+J EhI6U4iRM0izZ1lzOj5dpn4J785UW sYZ...	piyush.png	9825914530	customer

Figure 120: database before updating user

- After Updating User Profile

My Profile

Account Control Center



admin
customer

Username

Email Address

Phone Number

Change Password

Leave blank to keep current password

Please enter the current password if you don't want to update it before updating your profile.

Save Changes
Back to Home

Profile updated successfully. ×

Figure 121: After updating user profile

	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input checked="" type="checkbox"/> Delete	id	username	email	password	image_path	phone	role
				1	admin	admin@gmail.com	9hdjsiE5JwrbyyoT9DAK155EgMACGNvWISLBI7Zg2Mdy0jg6...	piyush.png	9825914530	customer

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.

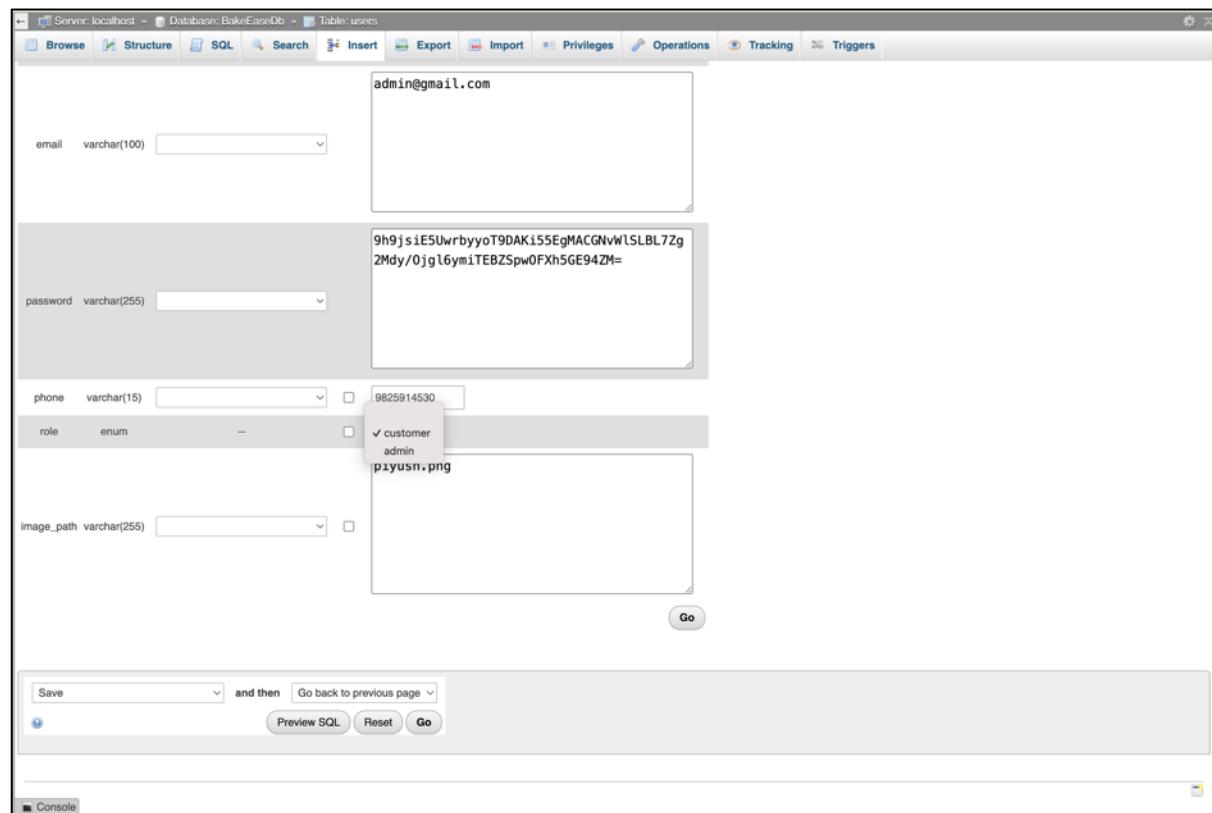
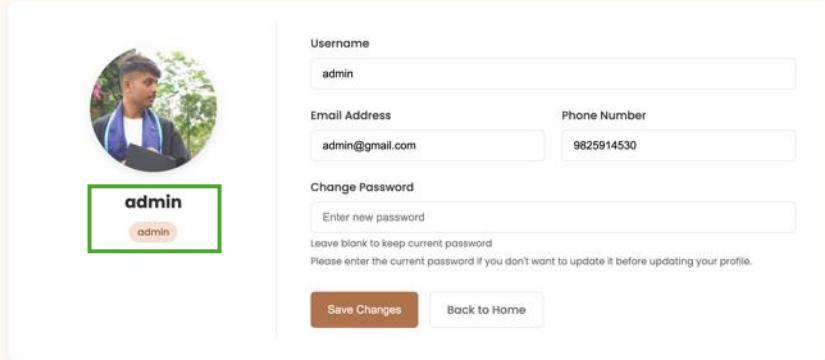


Figure 123: updating the user role from the database

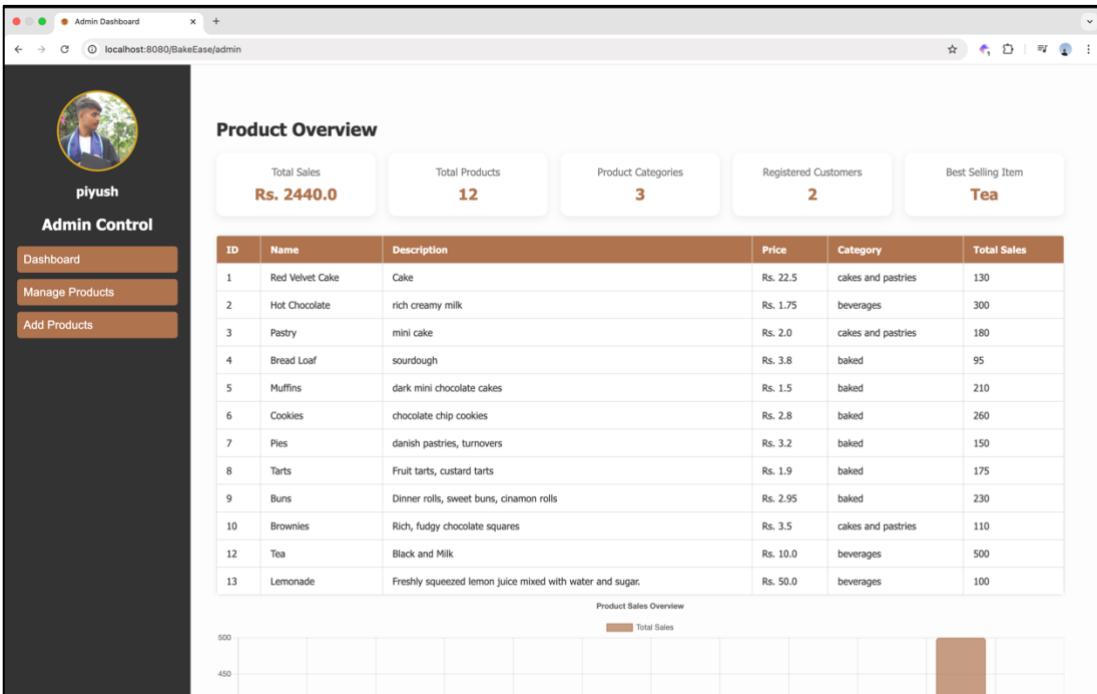
After the role is updated, the changes is also seen in the profile page.



The screenshot shows the 'My Profile' page from an account control center. On the left, there is a circular profile picture of a person and a name card with the text 'admin' and 'admin'. In the center, there are fields for 'Username' (admin), 'Email Address' (admin@gmail.com), and 'Phone Number' (9825914530). Below these fields is a 'Change Password' section with a note about current password requirements. At the bottom are 'Save Changes' and 'Back to Home' buttons.

Figure 124: changed role information in profile page

- Before adding product



The screenshot shows the Admin Dashboard with a sidebar titled 'Admin Control' containing 'Dashboard', 'Manage Products', and 'Add Products' options. The main area displays a 'Product Overview' with statistics: Total Sales (Rs. 2440.0), Total Products (12), Product Categories (3), Registered Customers (2), and Best Selling Item (Tea). Below this is a table of 13 products with columns for ID, Name, Description, Price, Category, and Total Sales. At the bottom is a chart titled 'Product Sales Overview' showing total sales.

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, cinnamon 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

Figure 125: Before adding product

- Adding Product

The screenshot shows the Admin Control interface. On the left, there's a sidebar with a user profile picture of Piyush and the name "piyush". Below the profile are three buttons: "Dashboard", "Manage Products", and "Add Products", with "Add Products" being the active tab. The main area has a green header bar with the title "Add Products". Below it, there are two input fields: one for "Name" containing "Cold Coffee" and another for "Description" containing "cold coffee with milk". There are also two input fields for "Price": "30" and "100". A dropdown menu labeled "Category" is set to "Beverages". At the bottom right of the "Add Products" section is a brown button labeled "Add Product". Below this section is a table titled "Product Overview" with columns: ID, Name, Description, Price, Category, and Total Sales. The table contains 12 rows of data.

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, cinnamon 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

The screenshot shows the Admin Control interface. The sidebar and "Add Products" tab are the same as in Figure 126. The main area now displays a green success message "Product added successfully!". Below it is a summary card titled "Product Overview" with five metrics: Total Sales (Rs. 2540.0), Total Products (13), Product Categories (3), Registered Customers (2), and Best Selling Item (Tea). Below the summary card is a table titled "Product Overview" with columns: ID, Name, Description, Price, Category, and Total Sales. The table contains 14 rows of data, including the new entries for "Cold Coffee" and "Lemonade". At the bottom of the table is a legend for "Product Sales Overview" with a brown square labeled "Total Sales".

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, cinnamon rolls	Rs. 2.95	baked	230
10	Brownies	Rich, fudgy chocolate squares	Rs. 3.5	cakes and pastries	110
11	Tea	Black and Milk	Rs. 10.0	beverages	500
12	Cold Coffee	cold coffee with milk	Rs. 30.0	beverages	100
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

Product added successfully!

Product Overview

Total Sales	Total Products	Product Categories	Registered Customers	Best Selling Item
Rs. 2540.0	13	3	2	Tea

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, cinnamon 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

Product Sales Overview

Legend: Total Sales

Figure 128: displaying the added product in the dashboard table

localhost:8080/BakeEase/admin

Product Sales Overview

Legend: Total Sales

ID	Name	Description	Price	Category	Total Sales
9	Buns	Dinner rolls, sweet buns, cinnamon 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

Bar Chart:

- Red Velvet Cake: 130
- Hot Chocolate: 300
- Pastry: 180
- Bread Loaf: 95
- Muffins: 210
- Cookies: 260
- Pies: 150
- Tarts: 175
- Buns: 230
- Brownies: 110
- Tea: 500
- Lemonade: 100
- Cold Coffee: 100

Go Back to Home

Figure 129: displaying the added product in the dashboard graph

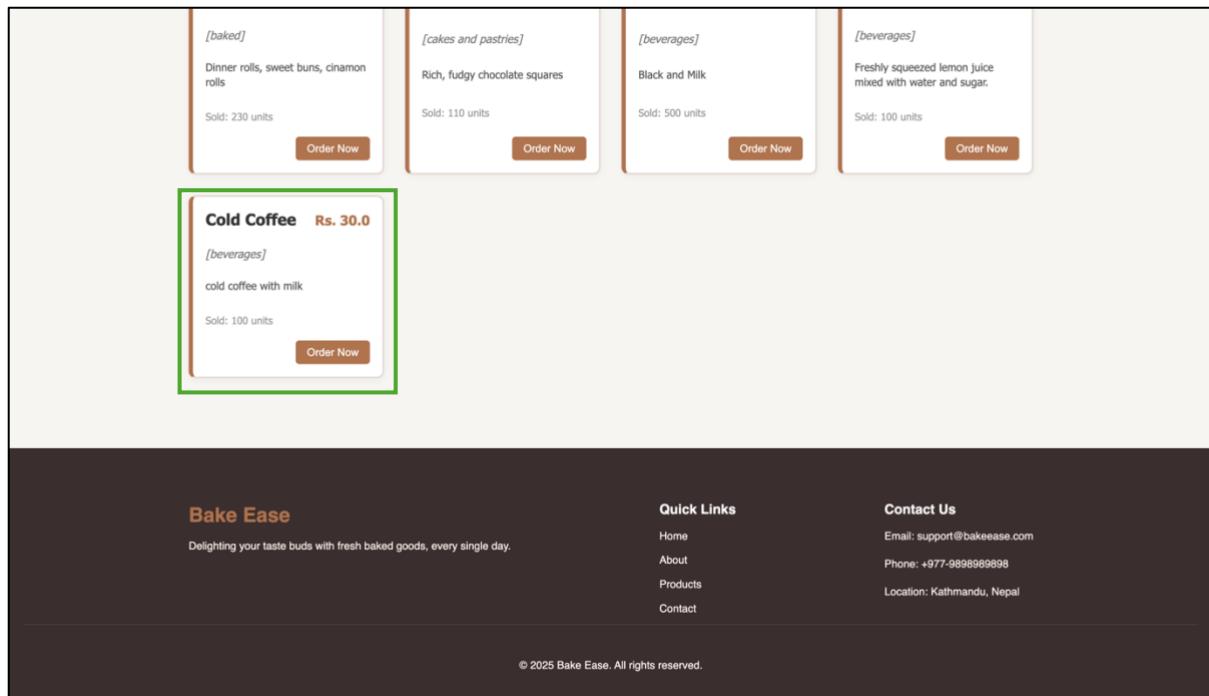


Figure 130: displaying the added product in the product page

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

Manage Products

Current Product Listings

ID	Name	Description	Price	Category	Total Sales	Actions
1	Red Velvet Cake	Cake	22.5	Beverages	130	<button>Update</button> <button>Delete</button>
2	Hot Chocolate	rich creamy milk	1.75	Beverages	300	<button>Update</button> <button>Delete</button>
3	Pastry	mini cake	2.0	Beverages	180	<button>Update</button> <button>Delete</button>
4	Bread Loaf	sourdough	3.8	Beverages	95	<button>Update</button> <button>Delete</button>
5	Muffins	dark mini chocolate cak	1.5	Beverages	210	<button>Update</button> <button>Delete</button>
6	Cookies	chocolate chip cookies	2.8	Beverages	260	<button>Update</button> <button>Delete</button>
7	Pies	danish pastries, turnover	3.2	Beverages	150	<button>Update</button> <button>Delete</button>
8	Tarts	Fruit tarts, custard tarts	1.9	Beverages	175	<button>Update</button> <button>Delete</button>
9	Buns	Dinner rolls, sweet bun	2.95	Beverages	230	<button>Update</button> <button>Delete</button>
10	Brownies	Rich, fudgy chocolate s	3.5	Beverages	110	<button>Update</button> <button>Delete</button>
12	Tea	Black and Milk	10.0	Beverages	500	<button>Update</button> <button>Delete</button>
13	Lemonade	Freshly squeezed lemo	50.0	Beverages	100	<button>Update</button> <button>Delete</button>
14	Cold Coffee	cold coffee with milk	30.0	Beverages	100	<button>Update</button> <button>Delete</button>

[Go Back to Home](#)

Figure 131: target product to update and delete

- **Updating Product**

Manage Products

Current Product Listings

ID	Name	Description	Price	Category	Total Sales	Actions
1	Red Velvet Cake	Cake	22.5	Beverages	130	<button>Update</button> <button>Delete</button>
2	Hot Chocolate	rich creamy milk	1.75	Beverages	300	<button>Update</button> <button>Delete</button>
3	Pastry	mini cake	2.0	Beverages	180	<button>Update</button> <button>Delete</button>
4	Bread Loaf	sourdough	3.8	Beverages	95	<button>Update</button> <button>Delete</button>
5	Muffins	dark mini chocolate cak	1.5	Beverages	210	<button>Update</button> <button>Delete</button>
6	Cookies	chocolate chip cookies	2.8	Beverages	260	<button>Update</button> <button>Delete</button>
7	Pies	danish pastries, turnover	3.2	Beverages	150	<button>Update</button> <button>Delete</button>
8	Tarts	Fruit tarts, custard tarts	1.9	Beverages	175	<button>Update</button> <button>Delete</button>
9	Buns	Dinner rolls, sweet bun	2.95	Beverages	230	<button>Update</button> <button>Delete</button>
10	Brownies	Rich, fudgy chocolate s	3.5	Beverages	110	<button>Update</button> <button>Delete</button>
12	Tea	Black and Milk	10.0	Beverages	500	<button>Update</button> <button>Delete</button>
13	Lemonade	Freshly squeezed lemo	50.0	Beverages	100	<button>Update</button> <button>Delete</button>
14	Butter croissant	french breakfast	40	Baked	120	<button>Update</button> <button>Delete</button>

[Go Back to Home](#)

Figure 132: updating target product

Product updated successfully!

Product Overview

Total Sales	Total Products	Product Categories	Registered Customers	Best Selling Item
Rs. 2560.0	13	3	2	Tea

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, cinnamon 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	Butter croissant	french breakfast	Rs. 40.0	baked	120

Product Sales Overview
Total Sales

Figure 133: after updating target product

Product updated successfully!

Product Overview

Total Sales	Total Products	Product Categories	Registered Customers	Best Selling Item
Rs. 2560.0	13	3	2	Tea

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, cinnamon 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	Butter croissant	french breakfast	Rs. 40.0	baked	120

Product Sales Overview
Total Sales

Figure 134: updated result in dashboard table

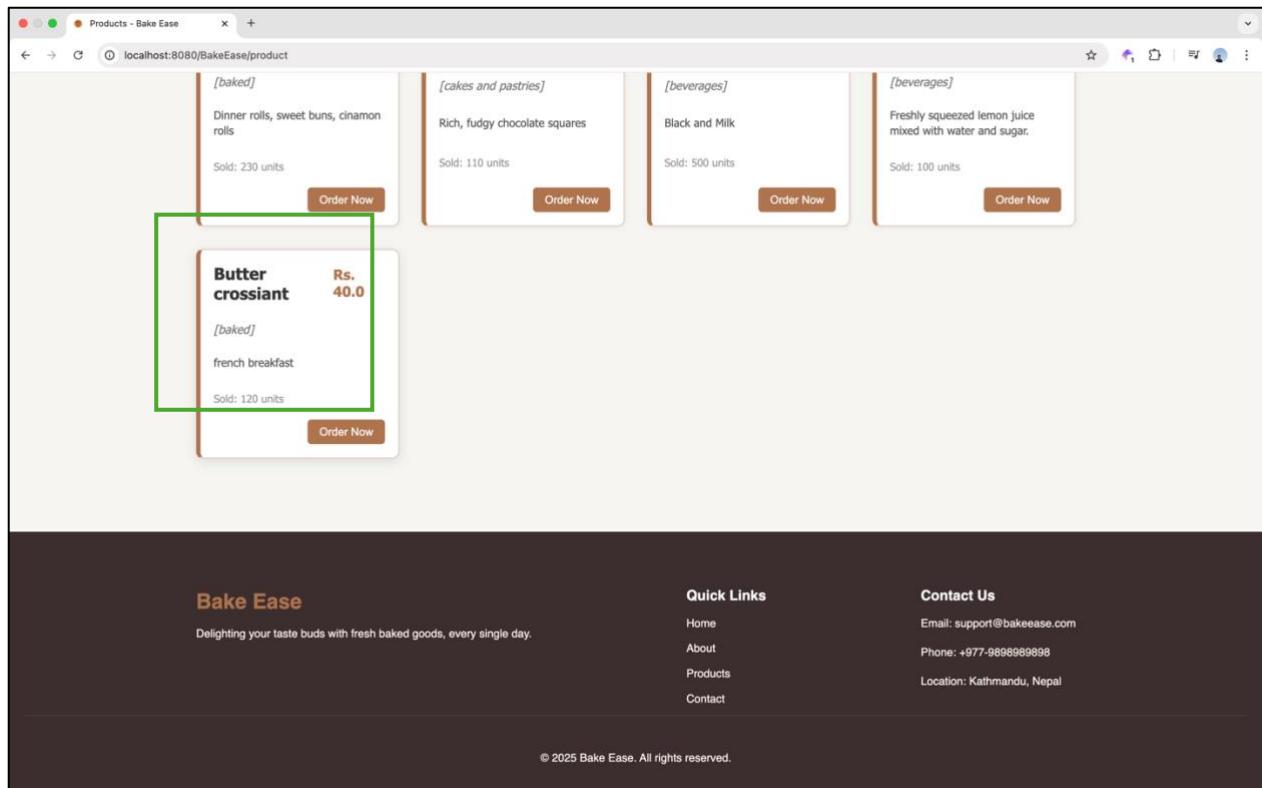


Figure 135: update result in products page

Server: localhost - Database: BakeEaseDb - Table: products									
Browse Structure SQL Search Insert Export Import Privileges Operations Tracking Triggers									
✓ Showing rows 0 - 12 (13 total). Query took 0.0004 seconds.									
SELECT * FROM `products`									
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]									
<input type="checkbox"/> Show all	Number of rows:	25	Filter rows:	Search this table	Sort by key:	None			
Extra options									
← →	id	name	description	price	category	total_sales			
<input type="checkbox"/>	Edit	Copy	Delete	1	Red Velvet Cake	Cake	22.50	cakes and pastries	130
<input type="checkbox"/>	Edit	Copy	Delete	2	Hot Chocolate	rich creamy milk	1.75	beverages	300
<input type="checkbox"/>	Edit	Copy	Delete	3	Pastry	mini cake	2.00	cakes and pastries	180
<input type="checkbox"/>	Edit	Copy	Delete	4	Bread Loaf	sourdough	3.80	baked	95
<input type="checkbox"/>	Edit	Copy	Delete	5	Muffins	dark mini chocolate cakes	1.50	baked	210
<input type="checkbox"/>	Edit	Copy	Delete	6	Cookies	chocolate chip cookies	2.80	baked	260
<input type="checkbox"/>	Edit	Copy	Delete	7	Pies	danish pastries, turnovers	3.20	baked	150
<input type="checkbox"/>	Edit	Copy	Delete	8	Tarts	Fruit tarts, custard tarts	1.90	baked	175
<input type="checkbox"/>	Edit	Copy	Delete	9	Buns	Dinner rolls, sweet buns, cinnamon rolls	2.95	baked	230
<input type="checkbox"/>	Edit	Copy	Delete	10	Brownies	Rich, fudgy chocolate squares	3.50	cakes and pastries	110
<input type="checkbox"/>	Edit	Copy	Delete	12	Tea	Black and Milk	10.00	beverages	500
<input type="checkbox"/>	Edit	Copy	Delete	13	Lemonade	Freshly squeezed lemon juice mixed with water and ...	50.00	beverages	100
<input type="checkbox"/>	Edit	Copy	Delete	14	Butter crossiant	french breakfast	40.00	baked	120

Figure 136: update result in database

- **Deleting Target Product**

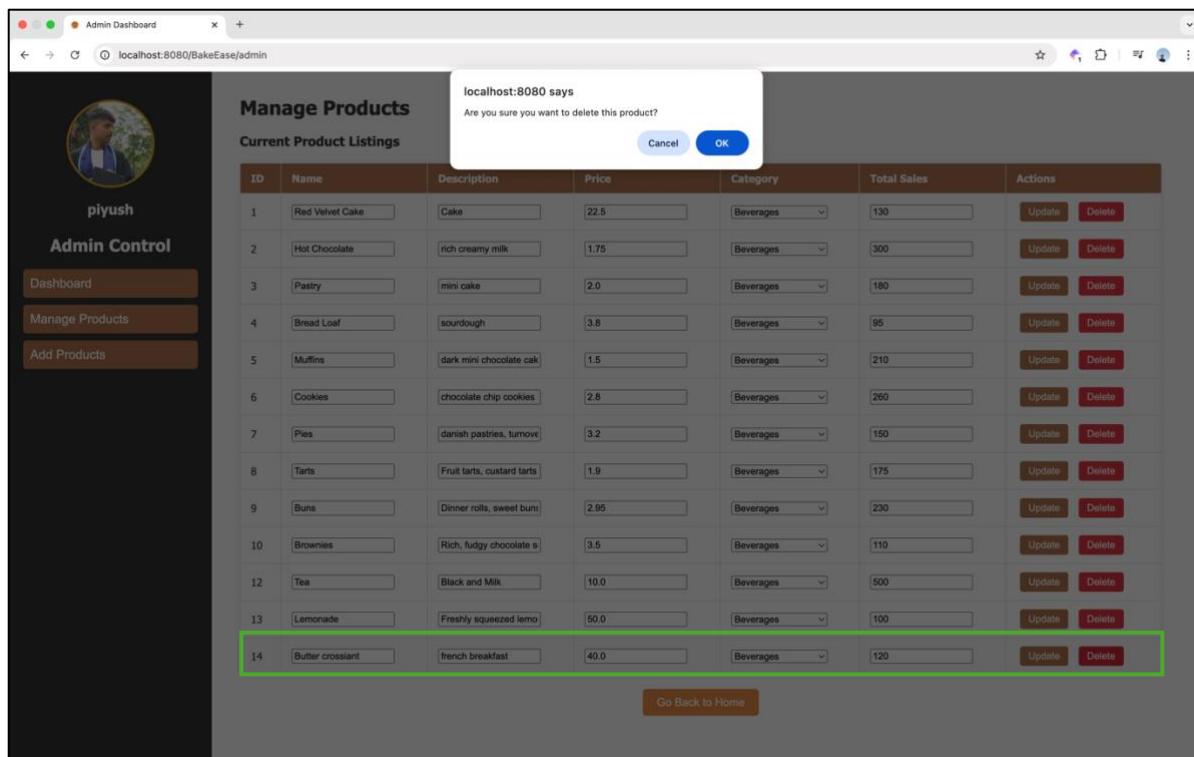


Figure 137: before deleting the target product

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, cinnamon 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

Figure 138: After deleting target product

The screenshot shows the MySQL Workbench interface with the 'products' table selected. A message at the top indicates 12 rows were found. The table data shows 11 rows remaining, with the last row (id 13) being a deleted entry.

	<input type="checkbox"/>	<input type="checkbox"/> Edit	<input type="checkbox"/> Copy	<input type="checkbox"/> Delete	id	name	description	price	category	total_sales
					1	Red Velvet Cake	Cake	22.50	cakes and pastries	130
					2	Hot Chocolate	rich creamy milk	1.75	beverages	300
					3	Pastry	mini cake	2.00	cakes and pastries	180
					4	Bread Loaf	sourdough	3.80	baked	95
					5	Muffins	dark mini chocolate cakes	1.50	baked	210
					6	Cookies	chocolate chip cookies	2.80	baked	260
					7	Pies	danish pastries, turnovers	3.20	baked	150
					8	Tarts	Fruit tarts, custard tarts	1.90	baked	175
					9	Buns	Dinner rolls, sweet buns, cinnamon rolls	2.95	baked	230
					10	Brownies	Rich, fudgy chocolate squares	3.50	cakes and pastries	110
					12	Tea	Black and Milk	10.00	beverages	500
					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**

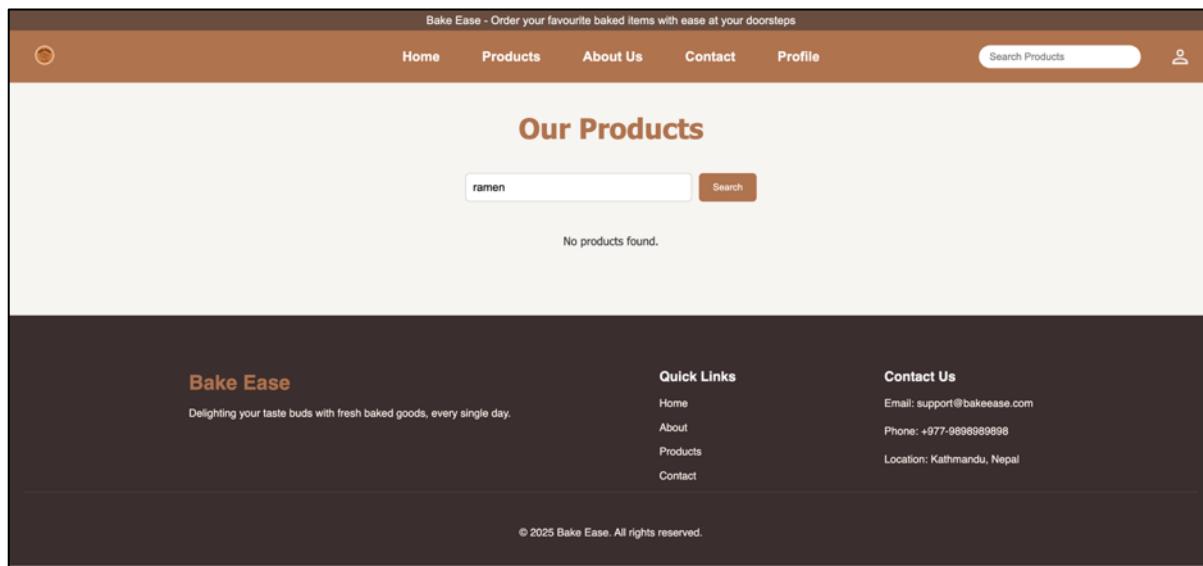


Figure 140: product not found

The screenshot shows a web application for ordering baked items. At the top, there's a navigation bar with links for Home, Products, About Us, Contact, and Profile, along with a search bar and a user icon. Below the navigation is a section titled "Our Products" with a search input field containing "cappuccino" and a "Search" button. A single product card is displayed for a "Cappuccino", categorized under "beverages". The card includes a description: "Espresso-based drink with steamed milk and foam.", a sold quantity of "260 units", and a price of "Rs. 2.8". A "Order Now" button is at the bottom right of the card. The footer contains sections for "Bake Ease" (tagline: "Delighting your taste buds with fresh baked goods, every single day."), "Quick Links" (links to Home, About, Products, and Contact), and "Contact Us" (with email, phone number, and location information). A copyright notice at the bottom states "© 2025 Bake Ease. All rights reserved."

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

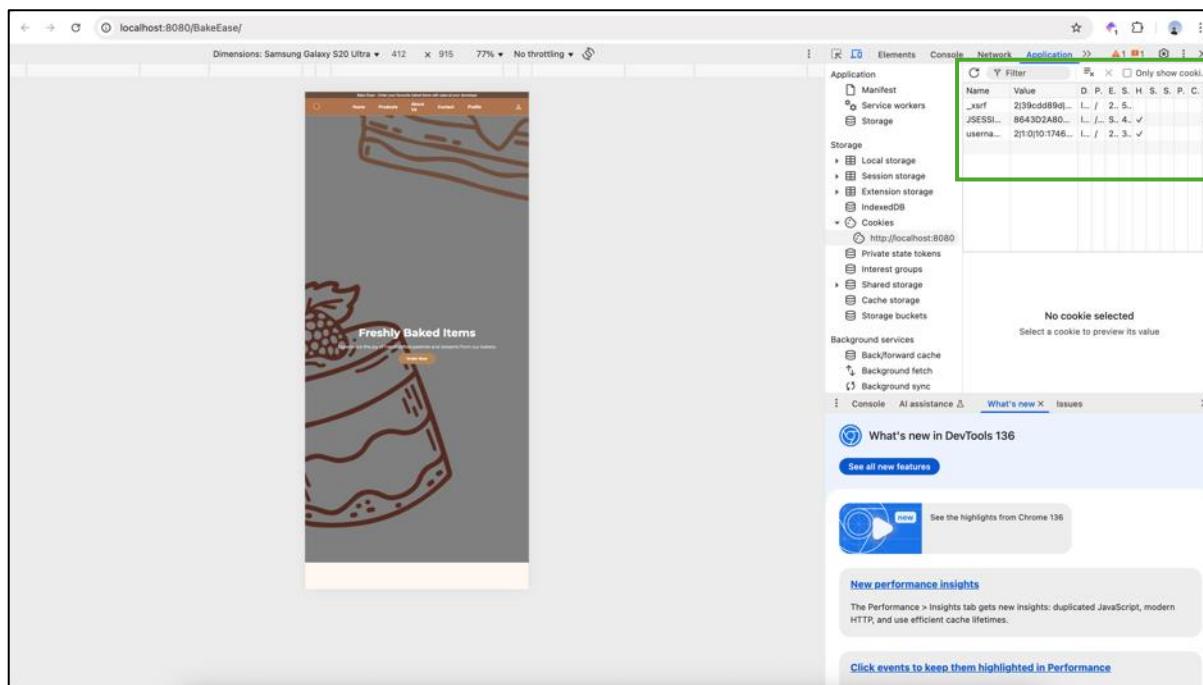


Figure 142: cookie testing before logging in

After Logging in

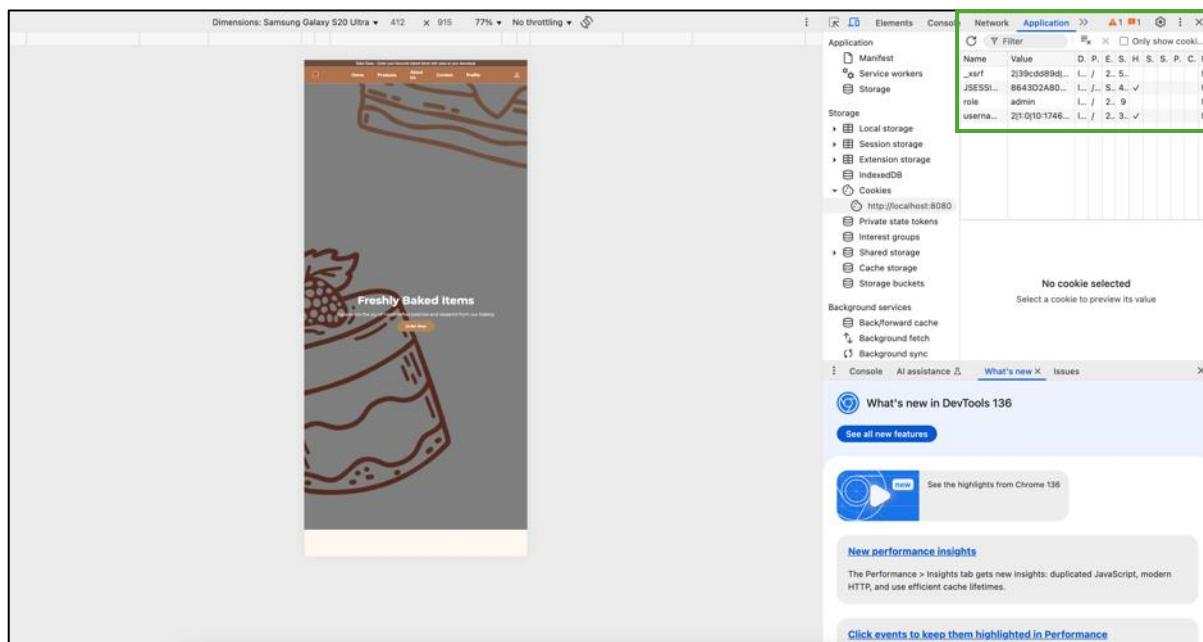


Figure 143: testing cookie after logging in

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

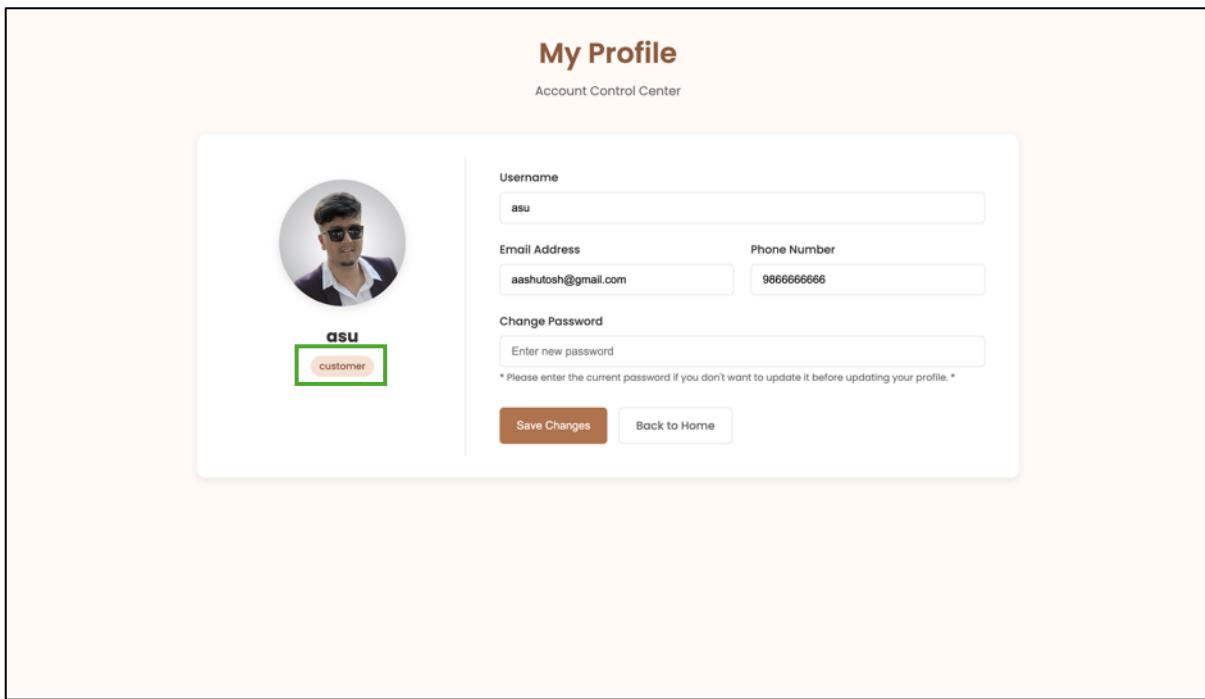


Figure 144: logging in as a customer

Trying to access admin dashboard as a customer



Figure 145: trying to access admin dashboard as a customer

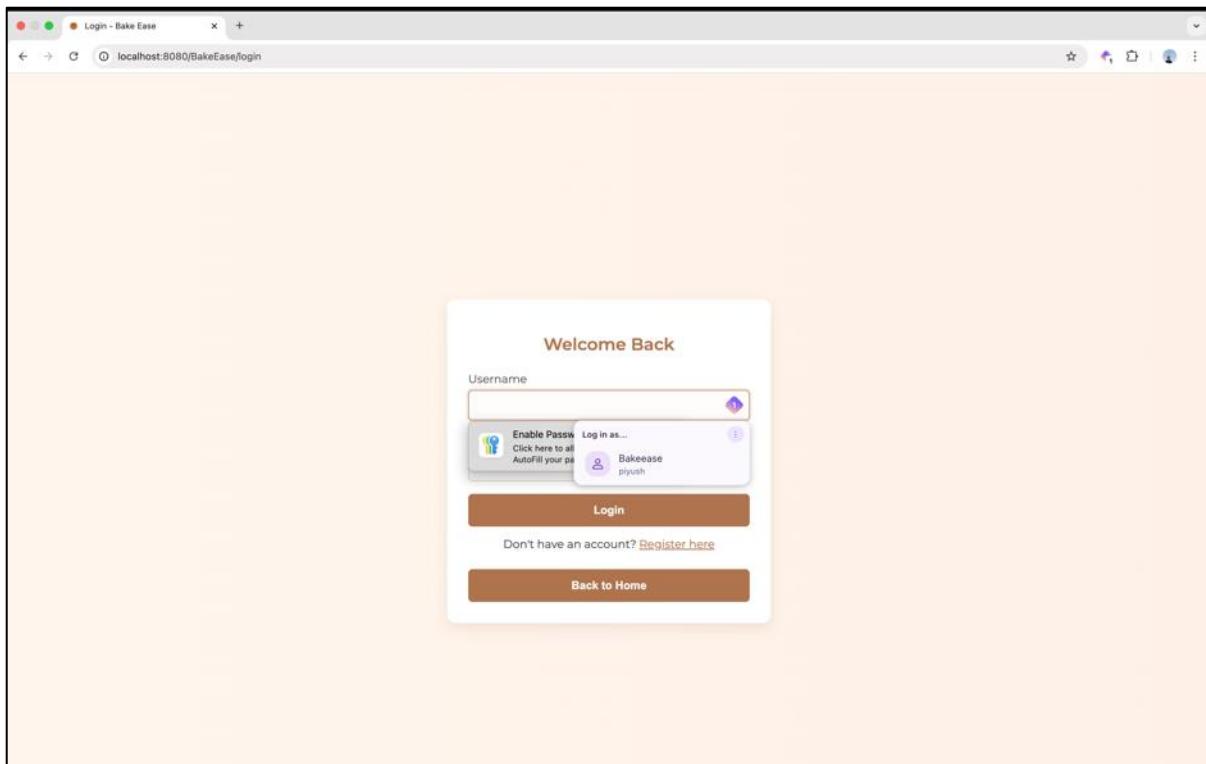


Figure 146: login redirection

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

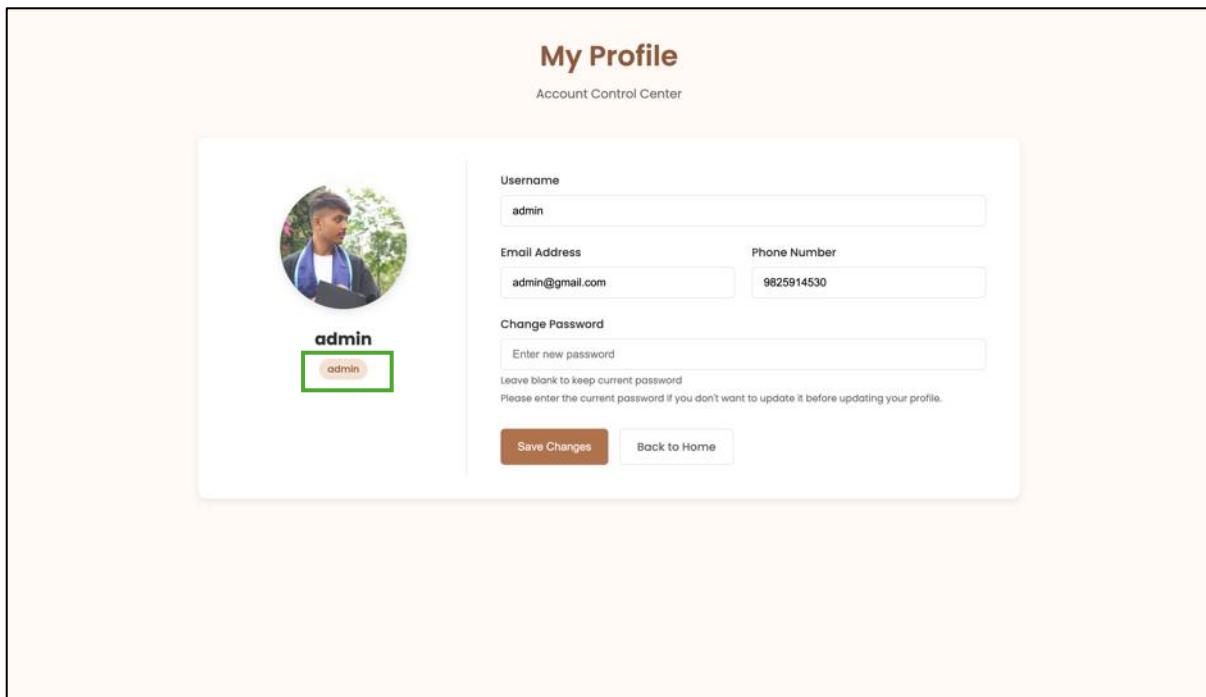


Figure 147: logging in as admin

Trying to access admin dashboard as admin

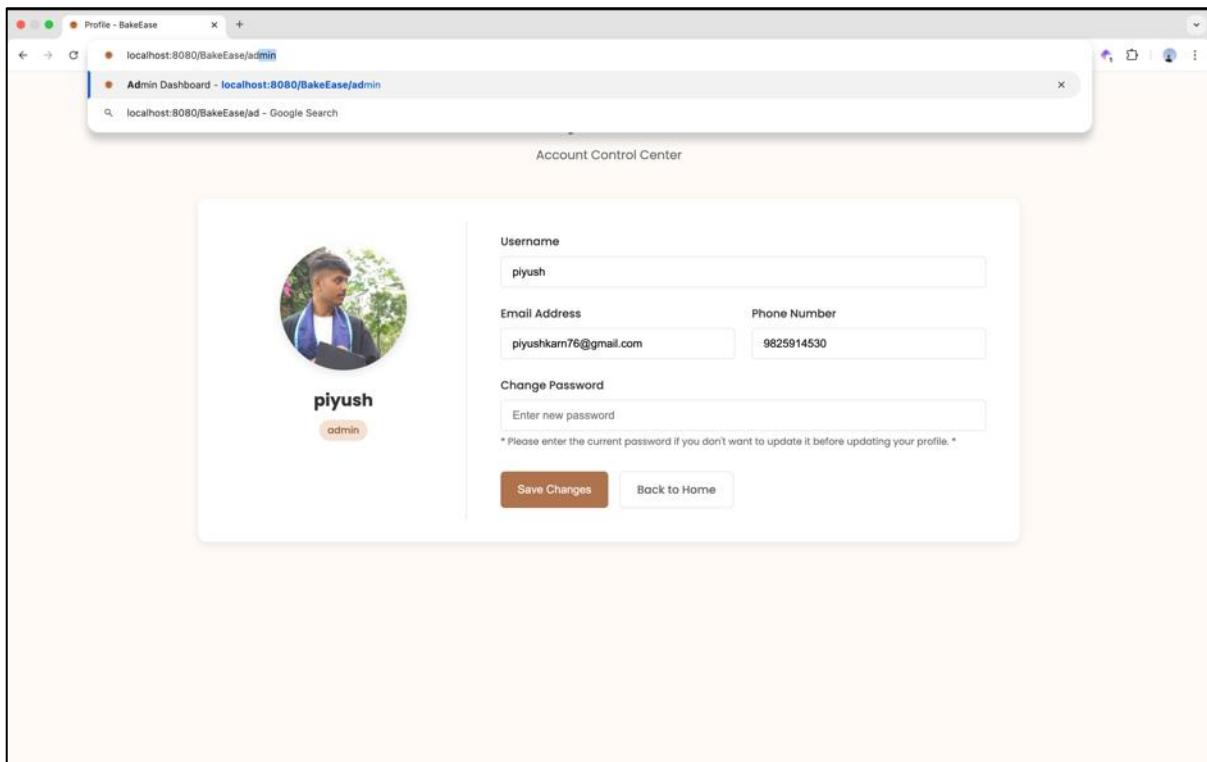


Figure 148: accessing admin dashboard as admin

Successfully redirected to Admin Dashboard

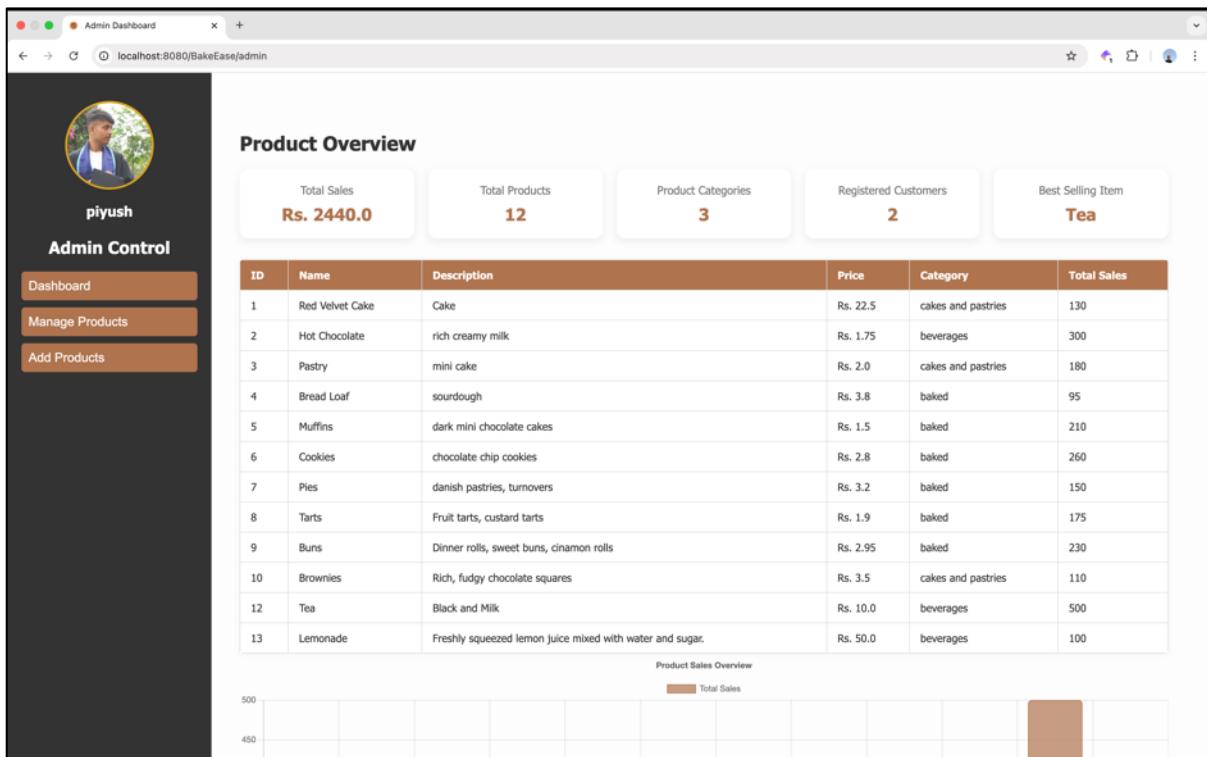


Figure 149: admin page redirection

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

Server: localhost - Database: BokEasyDb - Table: users								
Browse		Structure	SQL	Search	Insert	Export	Import	Privileges
✓ Showing rows 0 - 2 (3 total, Query took 0.0003 seconds.)								
SELECT * FROM `users`								
<input type="checkbox"/> Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]								
<input type="checkbox"/> Show all Restore column order Number of rows: 25 Filter rows: Search this table Sort by key: None								
<input type="checkbox"/> Extra options								
	id	username	email	password	image_path	phone	role	
<input type="checkbox"/>	1	piyush	piyushkarn76@gmail.com	khXg+84yIckMrCGuB4EWAz15Tthh8dVWhTlIrqEBPBKamAWWTy...	piyush.png	9825914530	admin	<input type="button" value="Edit"/>
<input checked="" type="checkbox"/>	2	asu	aashutosh@gmail.com	6ucJzRZQWOyy6hBdWnDy79+eQf8PA/Yzi/Q5bZxsIX1MVp0mSO...	aashutosh.png	9866666666	customer	<input type="button" value="Edit"/>
<input type="checkbox"/>	3	arpan	arpan@gmail.com	UU/EIB/dUMz9x/BFh8RvrirE6/3lz4FOPCFYERdNC1lIRaklb...	arpan.png	9800000000	customer	<input type="button" value="Edit"/>

Figure 150: target user for the profile page

Logging in with the target user

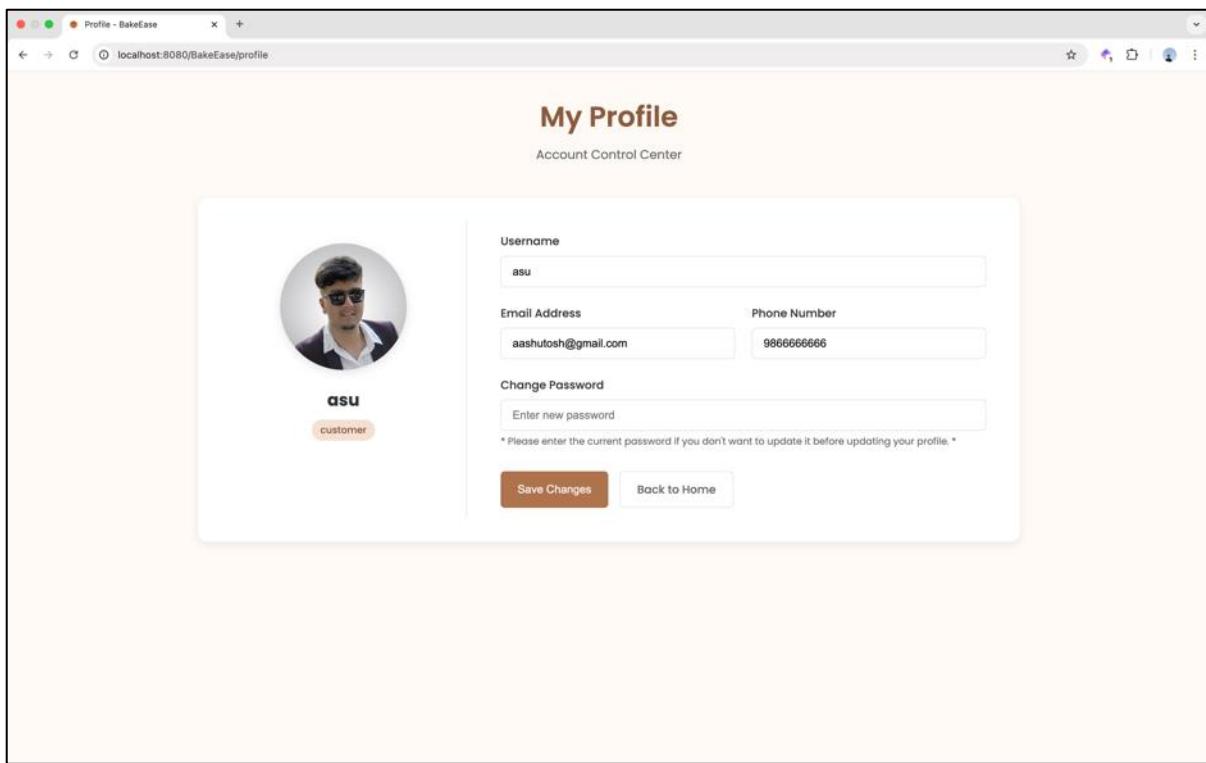


Figure 151: logging in with the target user

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

Showing rows 0 - 2 (3 total, Query took 0.0000 seconds.)								
SELECT * FROM `users` <input type="checkbox"/> Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]								
Show all Restore column order Number of rows: 25 Filter rows: Search this table Sort by key: None								
Extra options								
	#	id	username	email	password	image_path	phone	role
<input type="checkbox"/>	Edit	1	piyush	piyushkarn76@gmail.com	khXq+84ylckMrCGuB4EWAZt5Thth8dVWhTlrlqEBPBKamAWWTy...	piyush.png	9825914530	admin
<input checked="" type="checkbox"/>	Edit	2	aasu	aashutosh@gmail.com	6ucJzRZQWOyy6hBdWnDy79+eQFSPAYzaQ5bZxsIX1MVp0mSO...	aashutosh.png	9866666666	customer
<input type="checkbox"/>	Edit	3	arpan	arpan@gmail.com	UU/EIB/dUMz9xBFh6RvirE63Iz4FOPCFYERdNC1!fRlaklb...	arpan.png	9800000000	customer

Figure 152: target user for profile update

Current Profile credentials of the target user

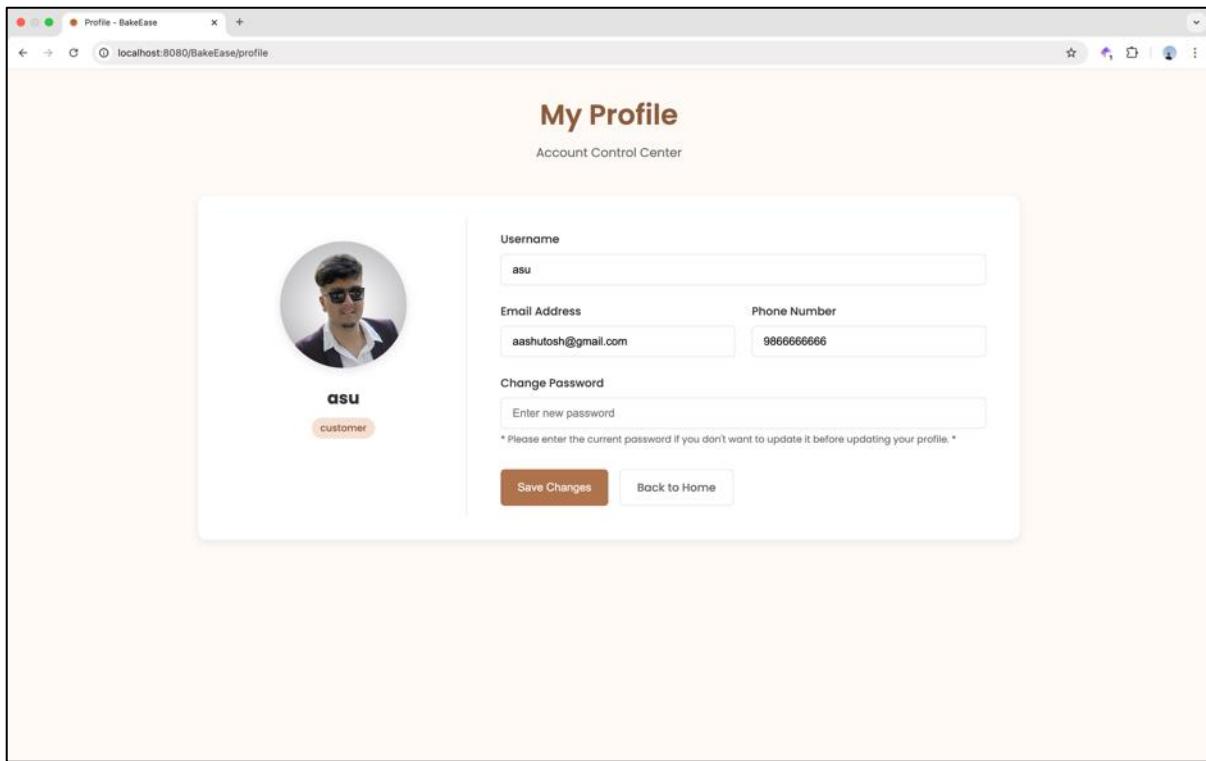


Figure 153: Current user credentials of the target user

Updating credentials of the target user

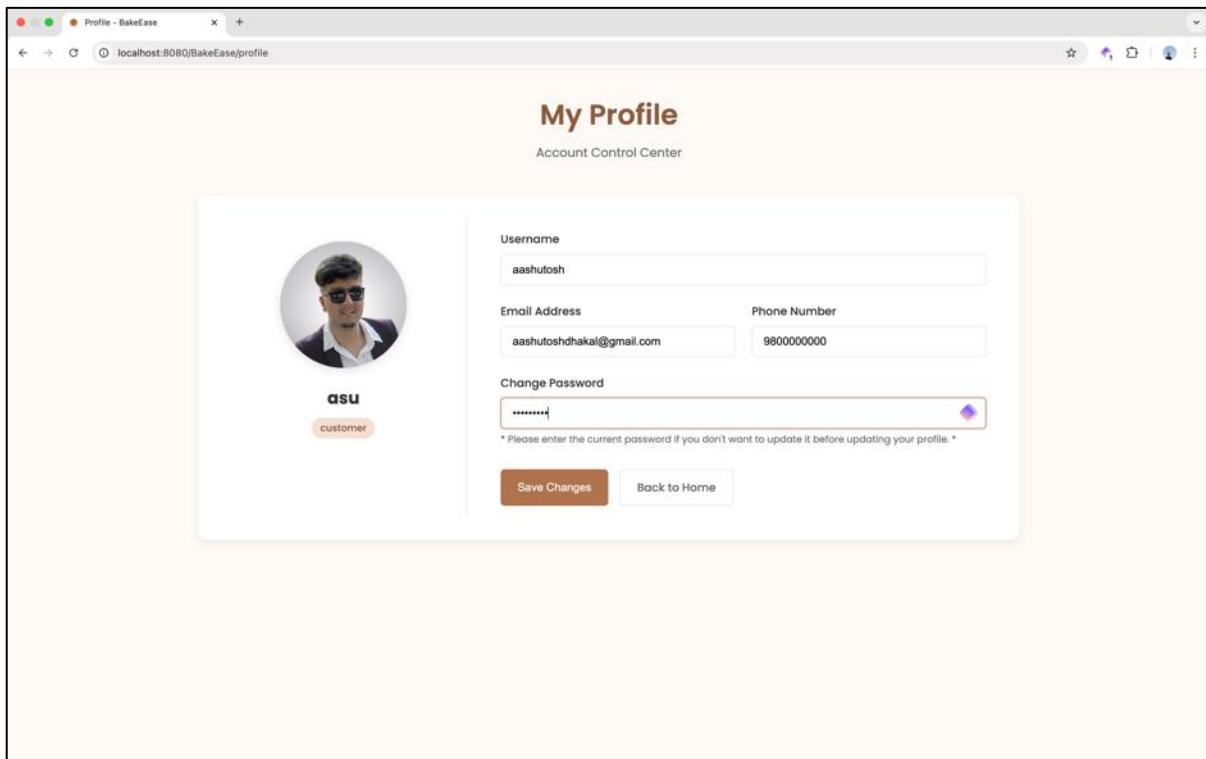


Figure 154: updating credentials of the target user

Updated user credentials of the target user

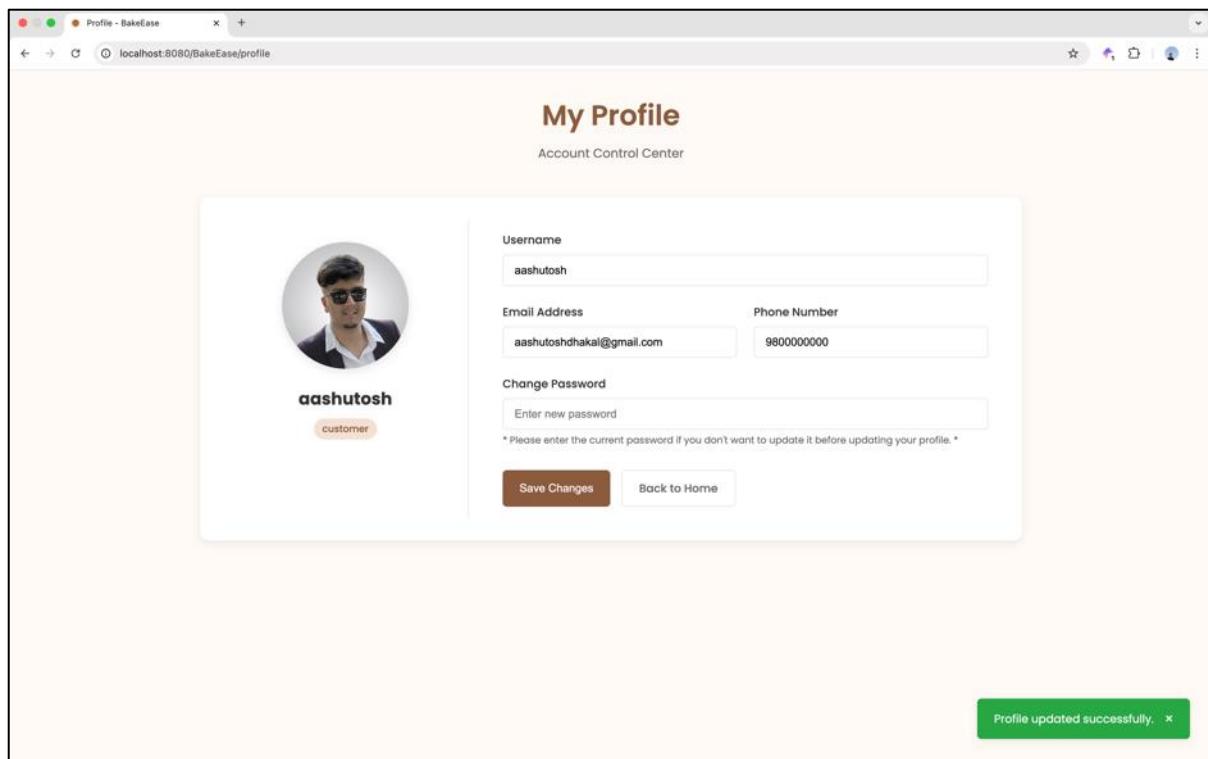


Figure 155: updated user credentials of the target user

After Updating User credentials of the target user

Table: users								
	#	id	username	email	password	image_path	phone	role
✓ Showing rows 0 - 2 (3 total, Query took 0.0001 seconds.)								
<code>SELECT * FROM `users`</code>								
<input type="checkbox"/> Profiling	Edit inline	Edit	Explain SQL	Create PHP code	Refresh			
<input type="checkbox"/> Show all			Restore column order	Number of rows:	25	Filter rows:	Search this table	Sort by key: None
Extra options								
<input type="checkbox"/>	Edit	Copy	Delete	1	piyushkam76@gmail.com	khXq+84yfckMrCGuB4EWAZt5Thth8dVWhTlrqEBPBKamAWWTy...	piyush.png	9825914530 admin
<input checked="" type="checkbox"/>	Edit	Copy	Delete	2	aashutosh	eun9lgMZVEA9XjVE0meFbilxYE1EDDf+tuuvV2+4myWbRNy7...	aashutosh.png	9800000000 customer
<input type="checkbox"/>	Edit	Copy	Delete	3	arpan	UU/EIBdUMz9XrBFh6RvrirE6/3lz4FOPCFYERdNC1flRakib...	arpan.png	9800000000 customer
Up	<input type="checkbox"/> Check all	With selected:		Edit	Copy	Delete	Export	

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)

```

1 package com.college.controller;
2
3 import java.io.IOException;
4 import java.time.LocalDate;
5
6 import com.college.model.ProgramModel;
7 import com.college.model.StudentModel;
8 import com.college.service.RegisterService;
9 import com.college.util.ImageUtil;
10 import com.college.util.PasswordUtil;
11 import com.college.util.ValidationUtil;
12
13 import jakarta.servlet.ServletException;
14 import jakarta.servlet.annotation.MultipartConfig;
15 import jakarta.servlet.annotation.WebServlet;
16 import jakarta.servlet.http.HttpServlet;
17 import jakarta.servlet.http.HttpServletRequest;
18 import jakarta.servlet.http.HttpServletResponse;
19 import jakarta.servlet.http.Part;
20
21 /**
22 * RegisterController handles user registration requests and processes form
23 * submissions. It also manages file uploads and account creation.
24 */
25 @WebServlet(asyncSupported = true, urlPatterns = { "/register" })
26 @MultipartConfig(fileSizeThreshold = 1024 * 1024 * 2, // 2MB
27                   maxFileSize = 1024 * 1024 * 10, // 10MB
28                   maxRequestSize = 1024 * 1024 * 50) // 50MB
29 public class RegisterController extends HttpServlet {
30     private static final long serialVersionUID = 1L;
31
32     private final ImageUtil imageUtil = new ImageUtil();
33     private final RegisterService registerService = new RegisterService();
34 }

```

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.

A screenshot of a web-based registration form titled "Create Account". The form includes fields for Username (filled with "piyush"), Email (filled with "piyushkarn76@gmail.com"), Password (filled with a series of dots), Confirm Password (filled with a series of dots), Phone Number (filled with "9812121212"), and Profile Image (with a "Choose File" button and a thumbnail labeled "me.jpg"). Below the form are two buttons: "Register" (in brown) and "Back to Home" (in brown). A link "Already have an account? [Login here](#)" is also present.

Figure 158: problem faced (before)

A screenshot of the same "Create Account" form, but now all input fields are empty. The Email field still shows the placeholder "piyushkarn76@gmail.com" and the Profile Image field still shows the thumbnail "me.jpg", but both are empty. The "Choose File" button is visible with the text "no file selected". The rest of the form and buttons are identical to Figure 158.

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.

```

110
111     if (isValid && registerService.isDuplicateUsernameOrEmail(username, email)) {
112         request.setAttribute("duplicateError", "Duplicate entries found. Please choose another username and email.");
113         isValid = false;
114     }
115
116     if (!isValid) {
117         request.setAttribute("username", username);
118         request.setAttribute("email", email);
119         request.setAttribute("phone", phone);
120         request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
121         return;
122     }
123
124     String encryptedPassword = PasswordUtil.encrypt(username, password);
125     if (encryptedPassword == null) {
126         request.setAttribute("error", "Error encrypting password. Try again.");
127         request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
128         return;
129     }
130
131     String role = "customer";
132     UserModel user = new UserModel(username, email, phone, encryptedPassword, role, fileName);
133
134     boolean isRegistered = registerService.addUser(user);
135     if (isRegistered) {
136         response.sendRedirect(request.getContextPath() + "/login");
137     } else {
138         request.setAttribute("error", "Registration failed. Try again.");
139         request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(request, response);
140     }
141 }
142

```

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.

The screenshot shows the MySQL Workbench interface with the 'users' table selected. The table has columns: id, username, email, password, phone, role, and image_path. The data is as follows:

	id	username	email	password	phone	role	image_path
<input type="checkbox"/>	1	test1	test@gmail.com	V5OVke/3sKvp1ydZvkNIQvgLfiUoiBB05Md!Eo+Bw3hOARqRre...	9801001000	admin	me.jpg
<input type="checkbox"/>	2	admin	admin@gmail.com	NRT2l2CYjzJr5+hekfICXkm2u+S8TIKOS85SmIIPJlVtjjpuB...	9833333333	customer	me.jpg
<input type="checkbox"/>	4	piyush	piyushkarn76@gmail.com	g10RkMm4h2pF3QBt1Iz12gYysG5Ovpk4dBoE+/BQnPQtw6a...	9825914530	customer	me.jpg

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.

```

CollegeSystem / main / java / com / college / controller / LoginController.java
Code Blame 99 Lines (87 loc) + 3.29 KB
public class LoginController extends HttpServlet {
    /**
     * Constructor initializes the LoginService.
     */
    public LoginController() {
        this.loginService = new LoginService();
    }
    /**
     * Handles GET requests to the login page.
     *
     * @param request HttpServletRequest object
     * @param response HttpServletResponse object
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException      if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(request, response);
    }
    /**
     * Handles POST requests for user login.
     *
     * @param request HttpServletRequest object
     * @param response HttpServletResponse object
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException      if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        StudentModel studentModel = new StudentModel(username, password);
        Boolean loginStatus = loginService.loginUser(studentModel);
    }
}

```

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.

The screenshot shows a "Welcome Back" login form. The "Username" field contains "piyush" and the "Password" field contains a masked password. A brown "Login" button is below them. Below the fields is a link "Don't have an account? [Register here](#)". At the bottom is a brown "Back to Home" button.

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

The screenshot shows the same "Welcome Back" login form. Now, both the "Username" and "Password" fields are empty. A red error message box at the top says "Invalid username or password." Below the fields is a link "Don't have an account? [Register here](#)". At the bottom is a brown "Back to Home" button.

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.

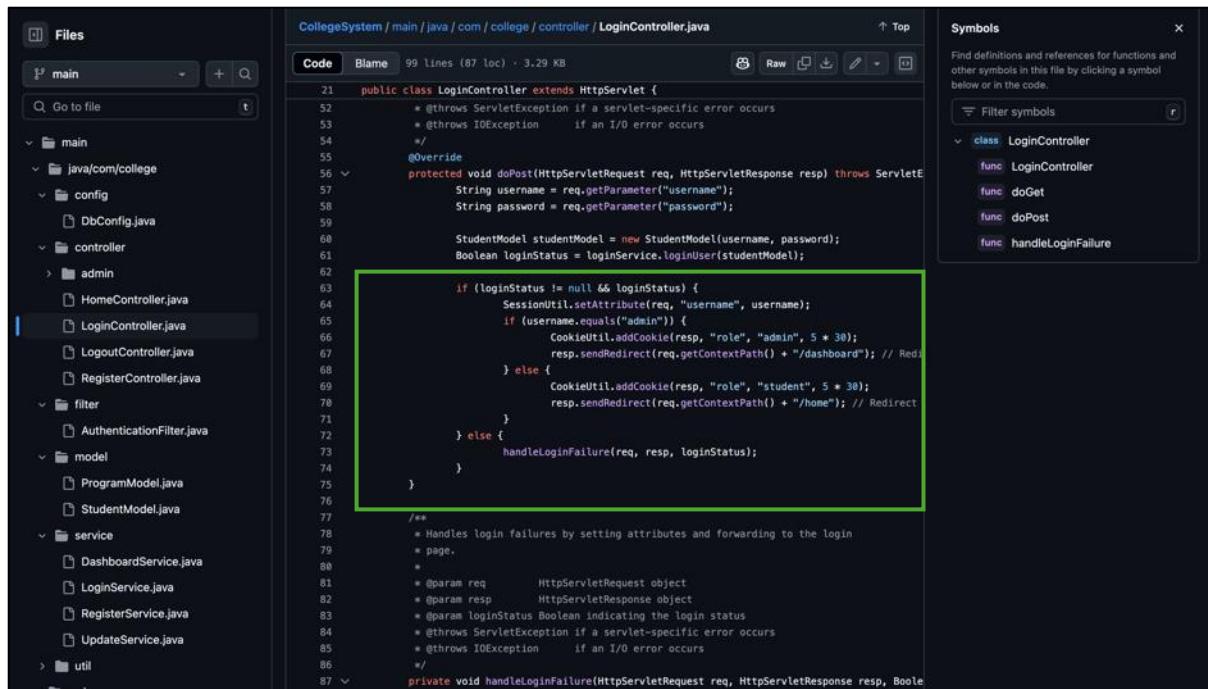
```

1 package com.bakeease.controller;
2
3 import com.bakeease.service.LoginService;
4
5 @WebServlet("/login")
6 public class Login extends HttpServlet {
7     private static final long serialVersionUID = 1L;
8
9     private final LoginService loginService = new LoginService();
10
11    @Override
12    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
13        req.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(req, resp);
14    }
15
16    @Override
17    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
18        request.setCharacterEncoding("UTF-8");
19
20        String username = request.getParameter("username");
21        String inputPassword = request.getParameter("password");
22
23        String role = null;
24
25        if (role != null) {
26            HttpSession session = request.getSession();
27            session.setAttribute("username", username);
28            CookieUtil.addCookie(response, "role", role, 60 * 60);
29
30            if ("admin".equalsIgnoreCase(role)) {
31                response.sendRedirect(request.getContextPath() + "/admin");
32            } else {
33                response.sendRedirect(request.getContextPath() + "/home");
34            }
35        }
36
37        return;
38    }
39
40    // Always executed, regardless of correct credentials
41    request.setAttribute("error", "Invalid username or password.");
42    request.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(request, response);
43
44 }
45
46
47
48 }
49

```

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.



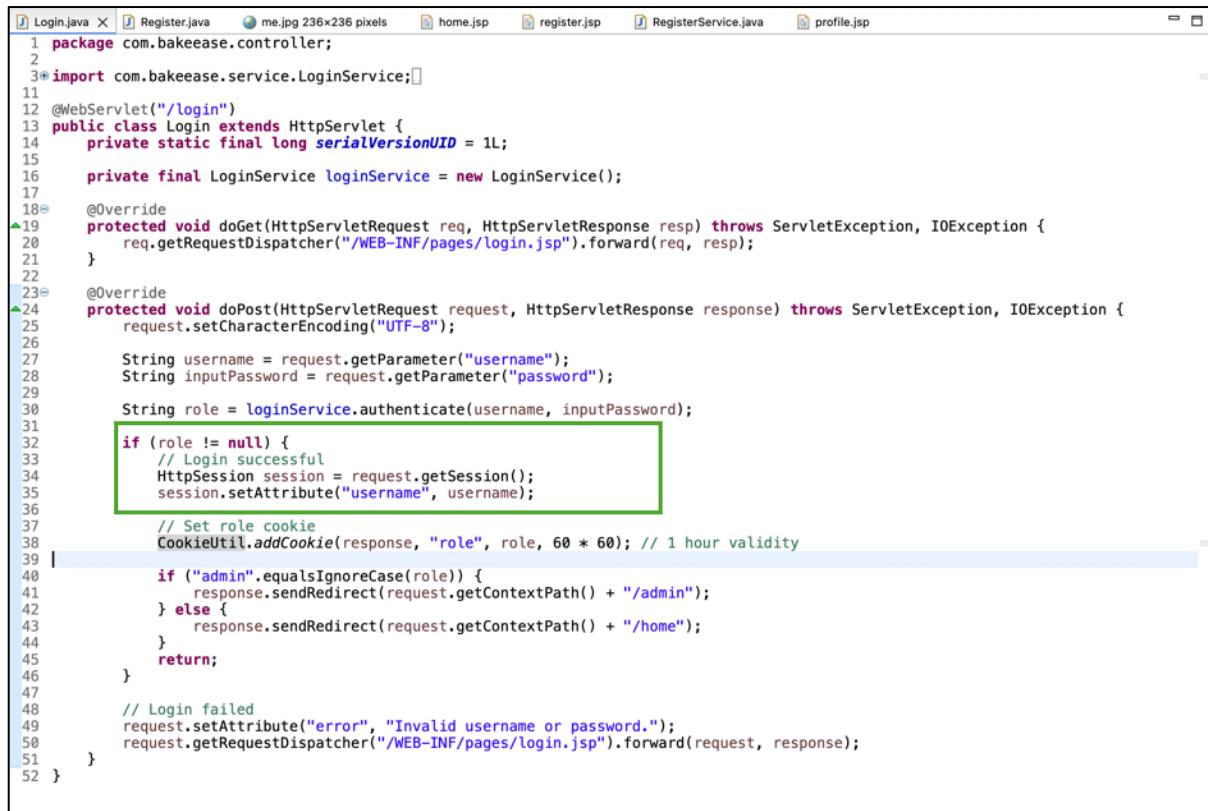
```

CollegeSystem / main / java / com / college / controller / LoginController.java
Code Blame 99 Lines (87 loc) · 3.29 KB
21 public class LoginController extends HttpServlet {
22     * @throws ServletException if a servlet-specific error occurs
23     * @throws IOException      if an I/O error occurs
24     */
25     @Override
26     protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
27         String username = req.getParameter("username");
28         String password = req.getParameter("password");
29
30         StudentModel studentModel = new StudentModel(username, password);
31         Boolean loginStatus = loginService.loginUser(studentModel);
32
33         if (loginStatus != null && loginStatus) {
34             SessionUtil.setAttribute(req, "username", username);
35             if (username.equals("admin")) {
36                 CookieUtil.addCookie(resp, "role", "admin", 5 * 30);
37                 resp.sendRedirect(req.getContextPath() + "/dashboard"); // Redirect
38             } else {
39                 CookieUtil.addCookie(resp, "role", "student", 5 * 30);
40                 resp.sendRedirect(req.getContextPath() + "/home"); // Redirect
41             }
42         } else {
43             handleLoginFailure(req, resp, loginStatus);
44         }
45     }
46
47     /**
48      * Handles login failures by setting attributes and forwarding to the login
49      * page.
50      *
51      * @param req      HttpServletRequest object
52      * @param resp     HttpServletResponse object
53      * @param loginStatus Boolean indicating the login status
54      * @throws ServletException if a servlet-specific error occurs
55      * @throws IOException      if an I/O error occurs
56      */
57     private void handleLoginFailure(HttpServletRequest req, HttpServletResponse resp, Boolean
58                                     loginStatus) {
59         req.setAttribute("error", "Invalid username or password.");
60         req.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(req, response);
61     }
62 }

```

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.



```

Login.java X Register.java me.jpg 236x236 pixels home.jsp register.jsp RegisterService.java profile.jsp
1 package com.bakeease.controller;
2
3 import com.bakeease.service.LoginService;
4
5 @WebServlet("/login")
6 public class Login extends HttpServlet {
7     private static final long serialVersionUID = 1L;
8
9     private final LoginService loginService = new LoginService();
10
11     @Override
12     protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
13         req.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(req, resp);
14     }
15
16     @Override
17     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
18         request.setCharacterEncoding("UTF-8");
19
20         String username = request.getParameter("username");
21         String inputPassword = request.getParameter("password");
22
23         String role = loginService.authenticate(username, inputPassword);
24
25         if (role != null) {
26             // Login successful
27             HttpSession session = request.getSession();
28             session.setAttribute("username", username);
29
30             // Set role cookie
31             CookieUtil.addCookie(response, "role", role, 60 * 60); // 1 hour validity
32
33             if ("admin".equalsIgnoreCase(role)) {
34                 response.sendRedirect(request.getContextPath() + "/admin");
35             } else {
36                 response.sendRedirect(request.getContextPath() + "/home");
37             }
38         }
39
40         // Login failed
41         request.setAttribute("error", "Invalid username or password.");
42         request.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(request, response);
43     }
44 }

```

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

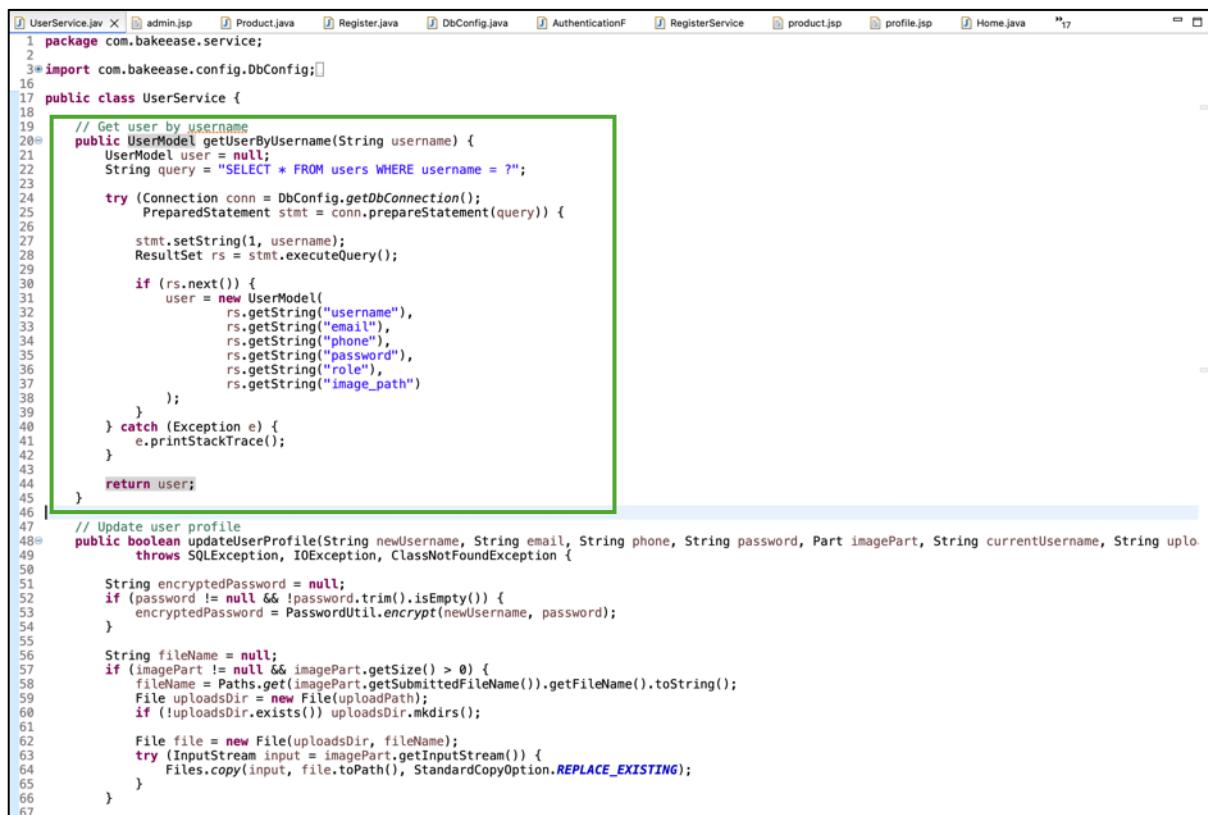
The profile feature is used to show the user their credentials and also allow user to update their user credentials from the account control center that is profile page.

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.



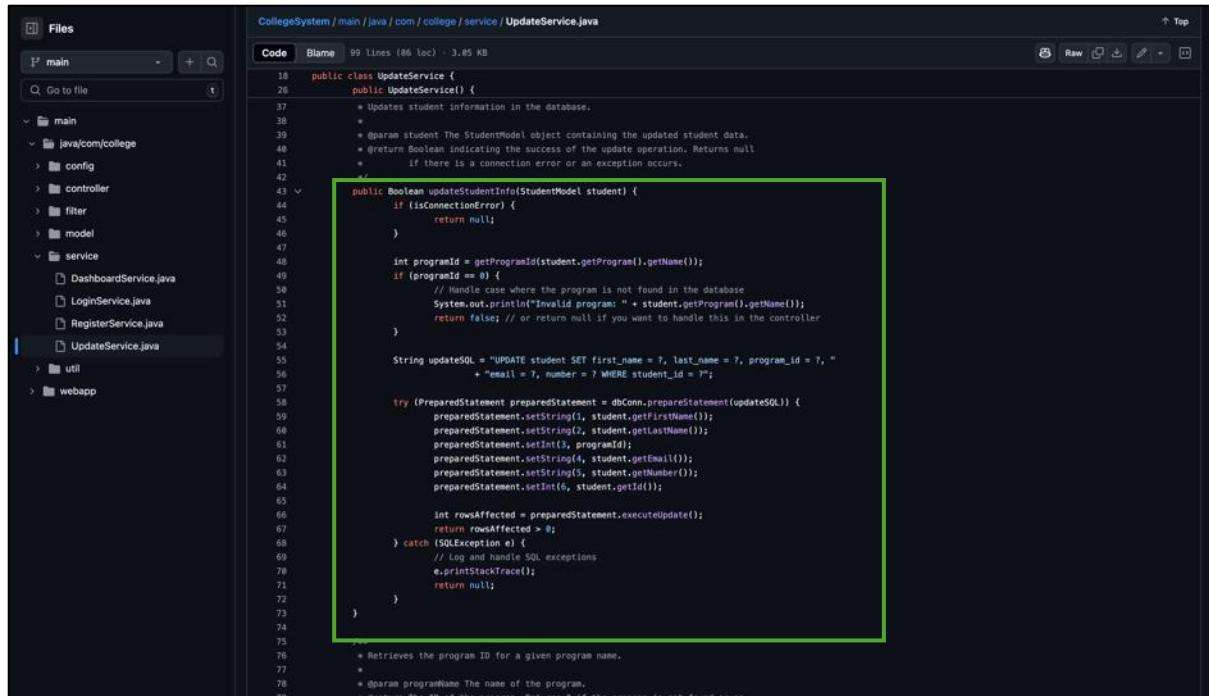
```

1 package com.bakeease.service;
2
3 import com.bakeease.config.DbConfig;
4
5 public class UserService {
6
7     // Get user by username
8     public UserModel getUserByUsername(String username) {
9         UserModel user = null;
10        String query = "SELECT * FROM users WHERE username = ?";
11
12        try (Connection conn = DbConfig.getDbConnection();
13             PreparedStatement stmt = conn.prepareStatement(query)) {
14
15            stmt.setString(1, username);
16            ResultSet rs = stmt.executeQuery();
17
18            if (rs.next()) {
19                user = new UserModel(
20                    rs.getString("username"),
21                    rs.getString("email"),
22                    rs.getString("phone"),
23                    rs.getString("password"),
24                    rs.getString("role"),
25                    rs.getString("image_path")
26                );
27            }
28        } catch (Exception e) {
29            e.printStackTrace();
30        }
31
32        return user;
33    }
34
35    // Update user profile
36    public boolean updateUserProfile(String newUsername, String email, String phone, String password, Part imagePart, String currentUsername, String uploadPath)
37        throws SQLException, IOException, ClassNotFoundException {
38
39        String encryptedPassword = null;
40        if (password != null && !password.trim().isEmpty()) {
41            encryptedPassword = PasswordUtil.encrypt(newUsername, password);
42        }
43
44        String fileName = null;
45        if (imagePart != null && imagePart.getSize() > 0) {
46            fileName = Paths.get(imagePart.getSubmittedFileName()).getFileName().toString();
47            File uploadsDir = new File(uploadPath);
48            if (!uploadsDir.exists()) uploadsDir.mkdirs();
49
50            File file = new File(uploadsDir, fileName);
51            try (InputStream input = imagePart.getInputStream()) {
52                Files.copy(input, file.toPath(), StandardCopyOption.REPLACE_EXISTING);
53            }
54        }
55    }
56}

```

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.



```

CollegeSystem / main / java / com / college / service / UpdateService.java
Code Blame 99 Lines (86 loc) - 3.05 KB
Top Raw Edit ▾
18 public class UpdateService {
19     public UpdateService() {
20
21         /*
22          * Updates student information in the database.
23          *
24          * @param student The StudentModel object containing the updated student data.
25          * @return Boolean indicating the success of the update operation. Returns null
26          *         if there is a connection error or an exception occurs.
27         */
28
29     public Boolean updateStudentInfo(StudentModel student) {
30         if (isConnectionError()) {
31             return null;
32         }
33
34         int programId = getProgramId(student.getProgram().getName());
35         if (programId == 0) {
36             // Handle case where the program is not found in the database
37             System.out.println("Invalid program: " + student.getProgram().getName());
38             return false; // or return null if you want to handle this in the controller
39         }
40
41         String updateSQL = "UPDATE student SET first_name = ?, last_name = ?, program_id = ?, "
42             + "email = ?, number = ? WHERE student_id = ?";
43
44         try (PreparedStatement preparedStatement = dbConn.prepareStatement(updateSQL)) {
45             preparedStatement.setString(1, student.getFirstName());
46             preparedStatement.setString(2, student.getLastName());
47             preparedStatement.setInt(3, programId);
48             preparedStatement.setString(4, student.getEmail());
49             preparedStatement.setString(5, student.getNumber());
50             preparedStatement.setInt(6, student.getId());
51
52             int rowsAffected = preparedStatement.executeUpdate();
53             return rowsAffected > 0;
54         } catch (SQLException e) {
55             // Log and handle SQL exceptions
56             e.printStackTrace();
57             return null;
58         }
59     }
60
61     /*
62      * Retrieves the program ID for a given program name.
63      *
64      * @param programName The name of the program.
65     */
66
67 }
68
69
70
71
72
73
74
75
76
77
78

```

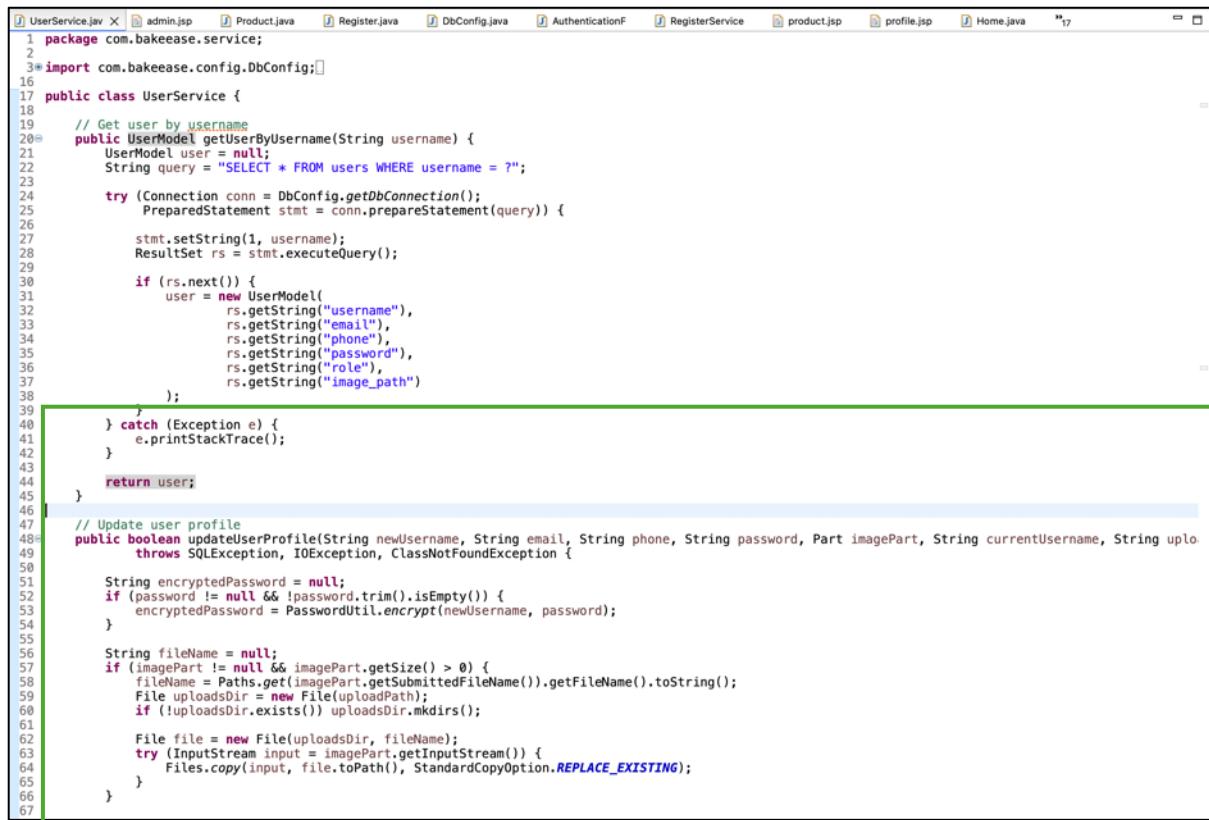
Figure 169: Internet support for update user profile

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.



```

1 package com.bakeease.service;
2
3 import com.bakeease.config.DbConfig;
4
5 public class UserService {
6
7     // Get user by username
8     public UserModel getUserByUsername(String username) {
9         UserModel user = null;
10        String query = "SELECT * FROM users WHERE username = ?";
11
12        try (Connection conn = DbConfig.getDbConnection();
13             PreparedStatement stmt = conn.prepareStatement(query)) {
14
15            stmt.setString(1, username);
16            ResultSet rs = stmt.executeQuery();
17
18            if (rs.next()) {
19                user = new UserModel(
20                    rs.getString("username"),
21                    rs.getString("email"),
22                    rs.getString("phone"),
23                    rs.getString("password"),
24                    rs.getString("role"),
25                    rs.getString("image_path")
26                );
27            }
28        } catch (Exception e) {
29            e.printStackTrace();
30        }
31
32        return user;
33    }
34
35    // Update user profile
36    public boolean updateUserProfile(String newUsername, String email, String phone, String password, Part imagePart, String currentUsername, String uploadPath)
37        throws SQLException, IOException, ClassNotFoundException {
38
39        String encryptedPassword = null;
40        if (password != null && !password.trim().isEmpty()) {
41            encryptedPassword = PasswordUtil.encrypt(newUsername, password);
42        }
43
44        String fileName = null;
45        if (imagePart != null && imagePart.getSize() > 0) {
46            fileName = Paths.get(imagePart.getSubmittedFileName()).getFileName().toString();
47            File uploadsDir = new File(uploadPath);
48            if (!uploadsDir.exists()) uploadsDir.mkdirs();
49
50            File file = new File(uploadsDir, fileName);
51            try (InputStream input = imagePart.getInputStream()) {
52                Files.copy(input, file.toPath(), StandardCopyOption.REPLACE_EXISTING);
53            }
54        }
55    }
56}

```

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

Output

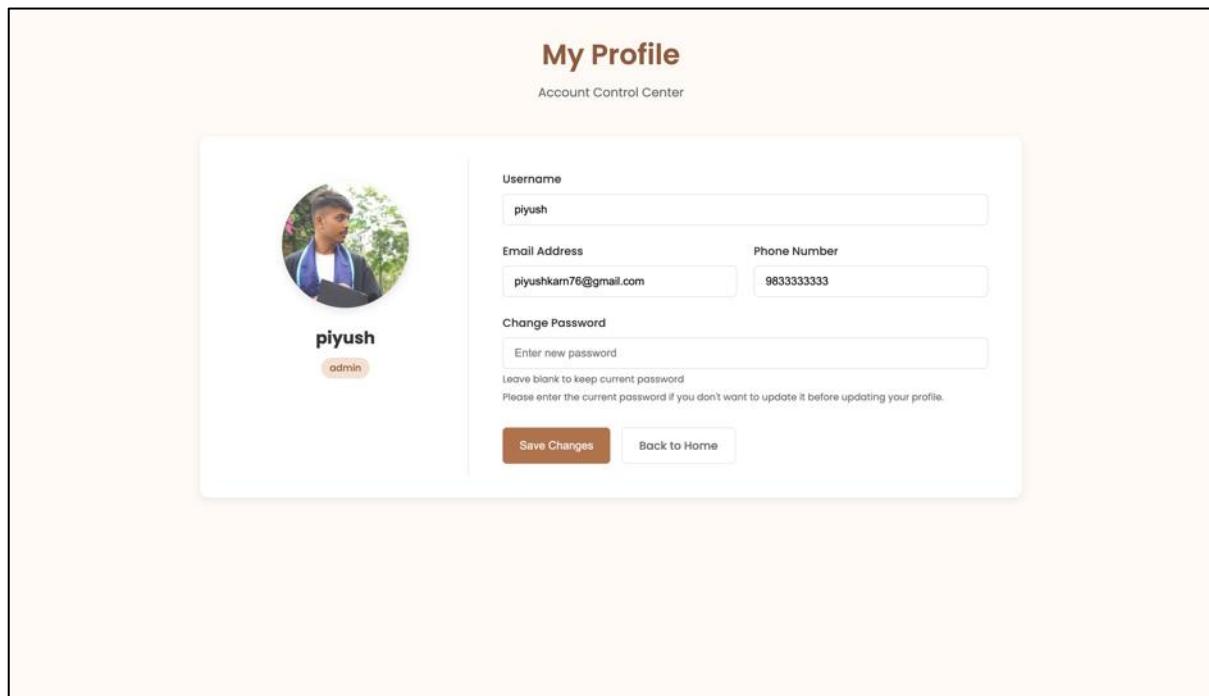


Figure 171: before updating user profile in profile page

User Management									
	Id	username	email	password	image_path	phone	role	Actions	
<input type="checkbox"/>	1	piyush	piyushkarn76@gmail.com	BRB2a2NAXdGsz0YdNzucOAaxeKRJdp2YLwHcmcZA+QWSUJyv9o...	piyush.png	9833333333	admin	 Edit	 Copy
<input type="checkbox"/>								 Delete	 Check all

Figure 172: before updating user credentials in database

My Profile

Account Control Center



admin
admin

Username

Email Address

Phone Number

Change Password

Leave blank to keep current password

Please enter the current password if you don't want to update it before updating your profile.

Save Changes
Back to Home

Profile updated successfully. 

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

User Management									
	Id	username	email	password	image_path	phone	role	Actions	
<input type="checkbox"/>	1	admin	admin@gmail.com	hqs0vWIReLM94hQ27wMzN73Hi7k3/TxiteMQ0bFnmY5zcLZpik...	piyush.png	9877777777	admin	 Edit	 Copy
<input type="checkbox"/>								 Delete	 Check all

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

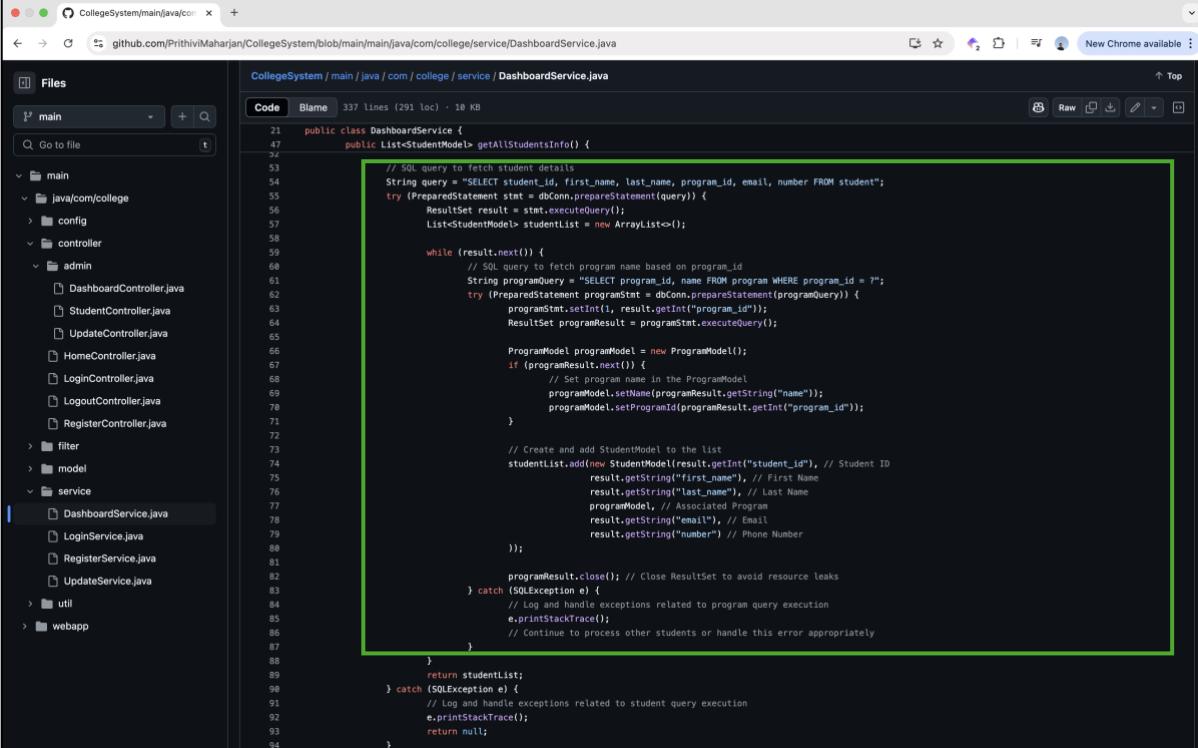
The feature required didn't require much effort as most features only required to run the sql queries for example to fetch the data from the database, to update the data to the database and delete data from the database. Just like other features, here also we used the GitHub repository provided to us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

23048660 Piyush Karn

124

6.5.2. Internet Support

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

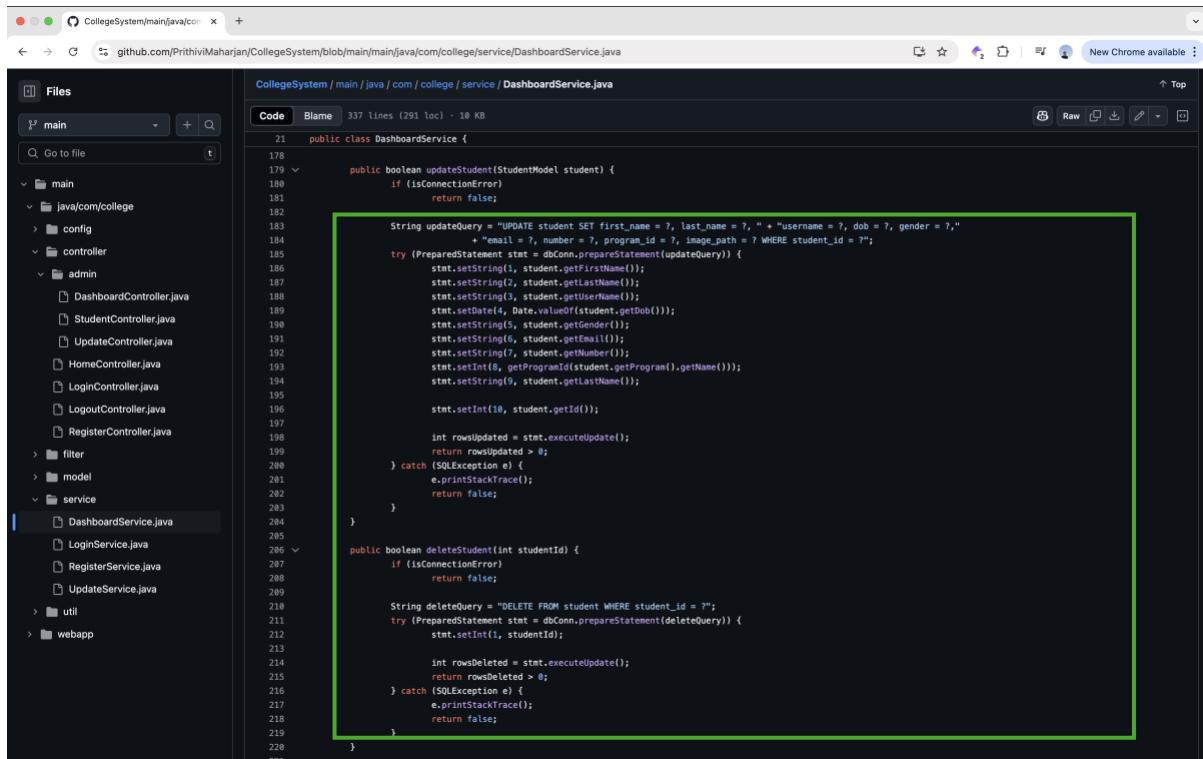


```

21  public class DashboardService {
22      public List<StudentModel> getAllStudentsInfo() {
23          // SQL query to fetch student details
24          String query = "SELECT student_id, first_name, last_name, program_id, email, number FROM student";
25          try (PreparedStatement stmt = dbConn.prepareStatement(query)) {
26              ResultSet result = stmt.executeQuery();
27              List<StudentModel> studentList = new ArrayList<>();
28
29              while (result.next()) {
30                  // SQL query to fetch program name based on program_id
31                  String programQuery = "SELECT program_id, name FROM program WHERE program_id = ?";
32                  try (PreparedStatement programStmt = dbConn.prepareStatement(programQuery)) {
33                      programStmt.setInt(1, result.getInt("program_id"));
34                      ResultSet programResult = programStmt.executeQuery();
35
36                      ProgramModel programModel = new ProgramModel();
37                      if (programResult.next()) {
38                          // Set program name in the ProgramModel
39                          programModel.setName(programResult.getString("name"));
40                          programModel.setProgramId(programResult.getInt("program_id"));
41
42                          // Create and add StudentModel to the list
43                          studentList.add(new StudentModel(result.getInt("student_id"), // Student ID
44                                              result.getString("first_name"), // First Name
45                                              result.getString("last_name"), // Last Name
46                                              programModel, // Associated Program
47                                              result.getString("email"), // Email
48                                              result.getString("number") // Phone Number
49
50                          ));
51
52                          programResult.close(); // Close ResultSet to avoid resource leaks
53                      } catch (SQLException e) {
54                          // Log and handle exceptions related to program query execution
55                          e.printStackTrace();
56
57                          // Continue to process other students or handle this error appropriately
58                      }
59                  }
60
61                  return studentList;
62              } catch (SQLException e) {
63                  // Log and handle exceptions related to student query execution
64                  e.printStackTrace();
65                  return null;
66              }
67          }
68      }
69  }

```

Figure 175: fetching data query



```

21  public class DashboardService {
178
179      public boolean updateStudent(StudentModel student) {
180          if (isConnectionError)
181              return false;
182
183          String updateQuery = "UPDATE student SET first_name = ?, last_name = ?, " + "username = ?, dob = ?, gender = ?," +
184                  + "email = ?, number = ?, program_id = ?, image_path = ? WHERE student_id = ?";
185          try (PreparedStatement stmt = dbConn.prepareStatement(updateQuery)) {
186              stmt.setString(1, student.getFirstName());
187              stmt.setString(2, student.getLastName());
188              stmt.setString(3, student.getUsername());
189              stmt.setDate(4, Date.valueOf(student.getDob()));
190              stmt.setString(5, student.getGender());
191              stmt.setString(6, student.getEmail());
192              stmt.setString(7, student.getNumber());
193              stmt.setInt(8, getProgramId(student.getProgram().getName()));
194              stmt.setString(9, student.getLastName());
195
196              stmt.setInt(10, student.getId());
197
198              int rowsUpdated = stmt.executeUpdate();
199              return rowsUpdated > 0;
200          } catch (SQLException e) {
201              e.printStackTrace();
202              return false;
203          }
204      }
205
206      public boolean deleteStudent(int studentId) {
207          if (isConnectionError)
208              return false;
209
210          String deleteQuery = "DELETE FROM student WHERE student_id = ?";
211          try (PreparedStatement stmt = dbConn.prepareStatement(deleteQuery)) {
212              stmt.setInt(1, studentId);
213
214              int rowsDeleted = stmt.executeUpdate();
215              return rowsDeleted > 0;
216          } catch (SQLException e) {
217              e.printStackTrace();
218              return false;
219          }
220      }
}

```

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.

```

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

```

Figure 177: code to search code

```

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

```

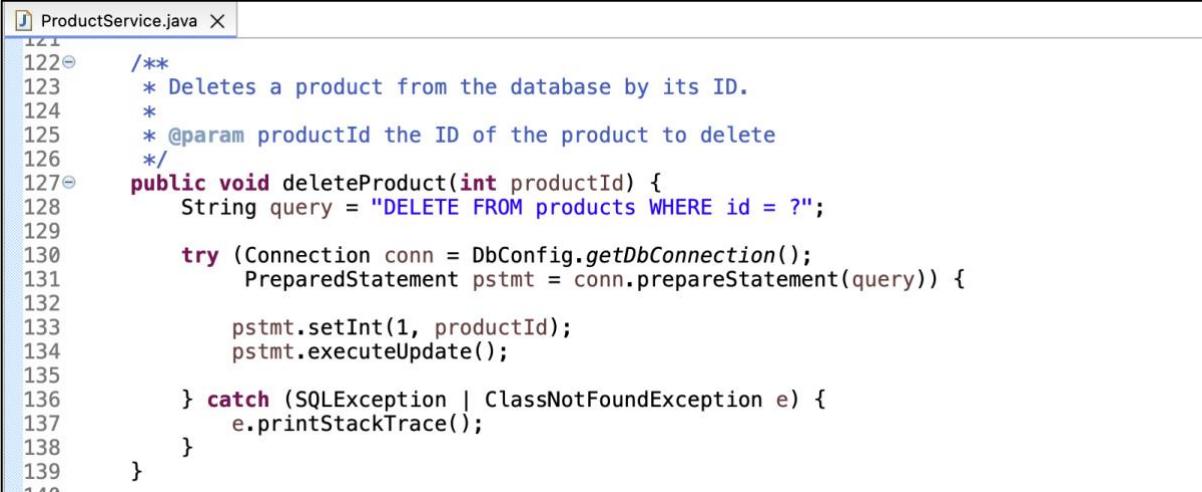
Figure 178: code to add product

```

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

```

Figure 179: code to update product



```

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

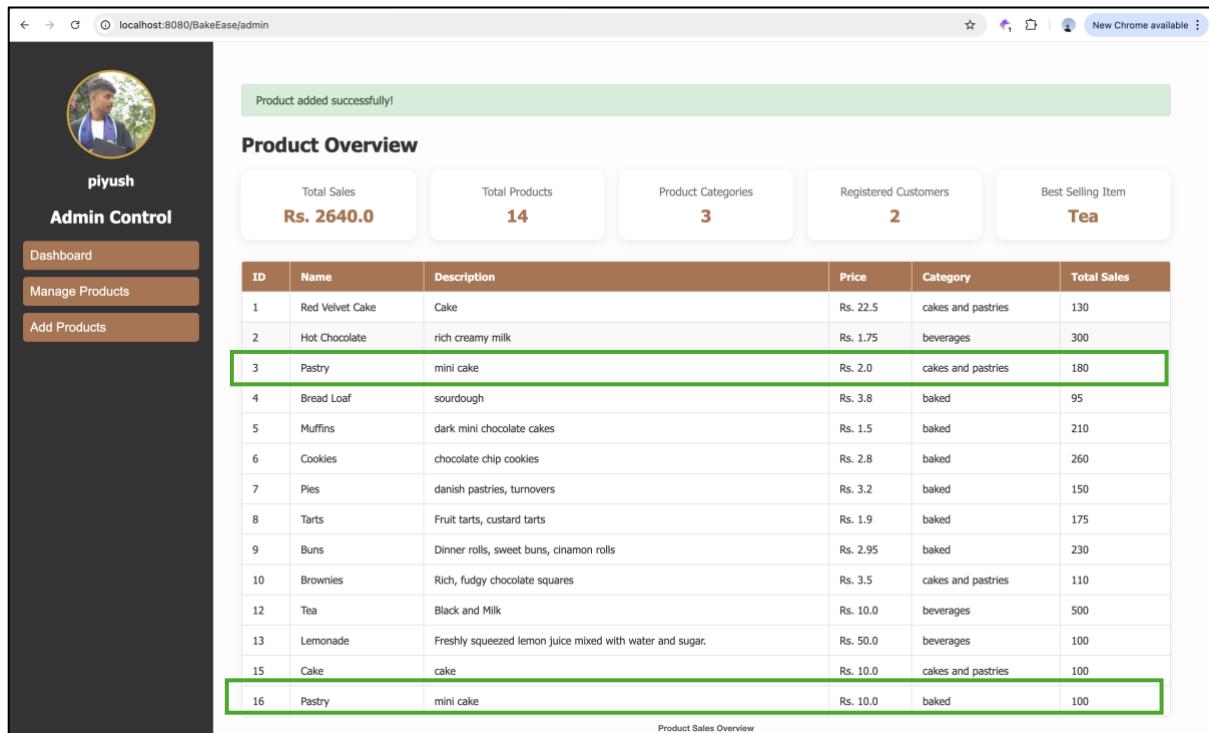
To solve this problem, we first took a glace at the database. To solve the problem it required the name column of the products table to be UNIQUE identifier. First we changed the column type of the products column.



#	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.



The screenshot shows a web application interface for 'BakeEase/admin'. On the left, there's a dark sidebar with a user profile picture of 'piyush' and a menu with 'Admin Control' at the top, followed by 'Dashboard', 'Manage Products', and 'Add Products'. The main content area has a green header bar stating 'Product added successfully!'. Below it, a section titled 'Product Overview' displays summary statistics: Total Sales (Rs. 2640.0), Total Products (14), Product Categories (3), Registered Customers (2), and Best Selling Item (Tea). A large table follows, showing a list of 16 products with columns for ID, Name, Description, Price, Category, and Total Sales. Two rows for 'Pastry' are highlighted with a green border, indicating a duplicate entry.

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, cinnamon 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
15	Cake	cake	Rs. 10.0	cakes and pastries	100
16	Pastry	mini cake	Rs. 10.0	baked	100

Figure 182: screenshot of the problem

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

The screenshot shows the MySQL Workbench interface. At the top, there are tabs for Structure, SQL, Search, Query, Export, Import, and Options. Below the tabs, there are two buttons: 'Multi-table query' and 'Query by example'. The 'Query by example' button is highlighted. A 'Query window' is open, displaying the following SQL code:

```

1 ALTER TABLE products
2 MODIFY COLUMN name VARCHAR(255) NOT NULL,
3 ADD UNIQUE (name);

```

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.

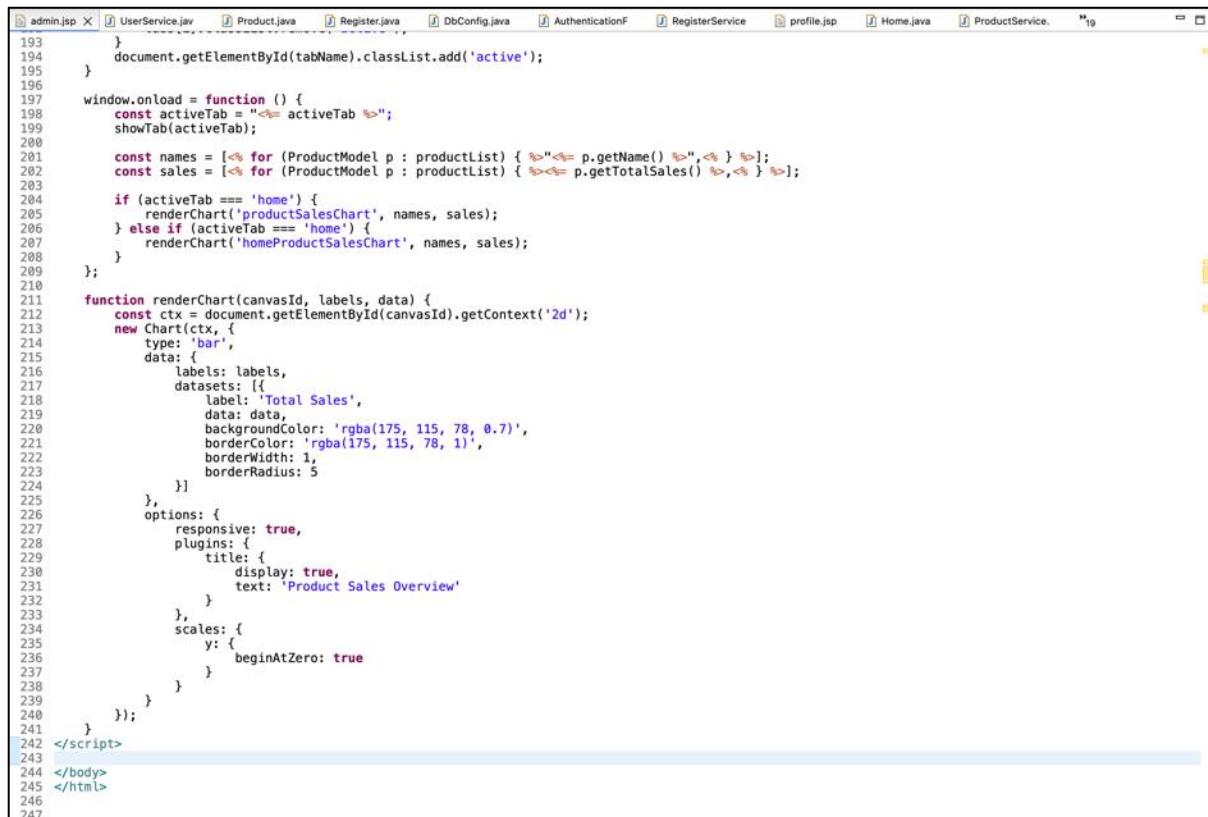
```

1 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2 <%@ page import="com.bakeease.model.ProductModel" %>
3 <%@ page import="java.util.List" %>
4
5<%>
6 // Ensure productList is accessible for entire page
7 List<ProductModel> productList = (List<ProductModel>) request.getAttribute("products");
8 String activeTab = (String) request.getAttribute("activeTab");
9 if (activeTab == null) activeTab = "home";
10 String successMessage = (String) request.getAttribute("successMessage");
11 <%>
12
13 <!DOCTYPE html>
14<html lang="en">
15<head>
16 <meta charset="UTF-8">
17 <title>Admin Dashboard</title>
18 <link rel="icon" type="image/png" href="${pageContext.request.contextPath}/resources/assets/favicon.png" />
19 <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}/css/admin.css" />
20 <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
21 </head>
22<body>
23
24<div class="sidebar">
25 <div class="sidebar-title">Admin</div>
26 <button class="tab-btn" onclick="showTab('home')">Home</button>
27 <button class="tab-btn" onclick="showTab('dashboard')">Dashboard</button>
28 <button class="tab-btn" onclick="showTab('products')">Products</button>
29 </div>
30
31<div class="main-content">
32
33 <% if (successMessage != null) { %>
34 <div class="alert success"><%= successMessage %></div>
35 <% } %>
36
37
38 <!-- Dashboard Tab -->
39<div id="dashboard" class="tab-content <%= "dashboard".equals(activeTab) ? "active" : "" %>">
40 <h2 class="dashboard-title">Manage Products</h2>
41
42 <div class="dashboard-table">
43 <div class="dashboard-cell">
44 <div class="dashboard-label">Total Sales</div>
45 <div class="dashboard-value">Rs. <%= request.getAttribute("totalSales") != null ? request.getAttribute("totalSales") : 0 %></div>
46 </div>
47 <div class="dashboard-cell">
48 <div class="dashboard-label">Items</div>
49 <div class="dashboard-value"><%= request.getAttribute("items") != null ? request.getAttribute("items") : 0 %></div>
50 </div>
51 <div class="dashboard-cell">
52 <div class="dashboard-label">Categories</div>
53 <div class="dashboard-value"><%= request.getAttribute("categories") != null ? request.getAttribute("categories") : 0 %></div>
54 </div>
55 <div class="dashboard-cell">

```

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.



```

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 </body>
244 </html>
245
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.

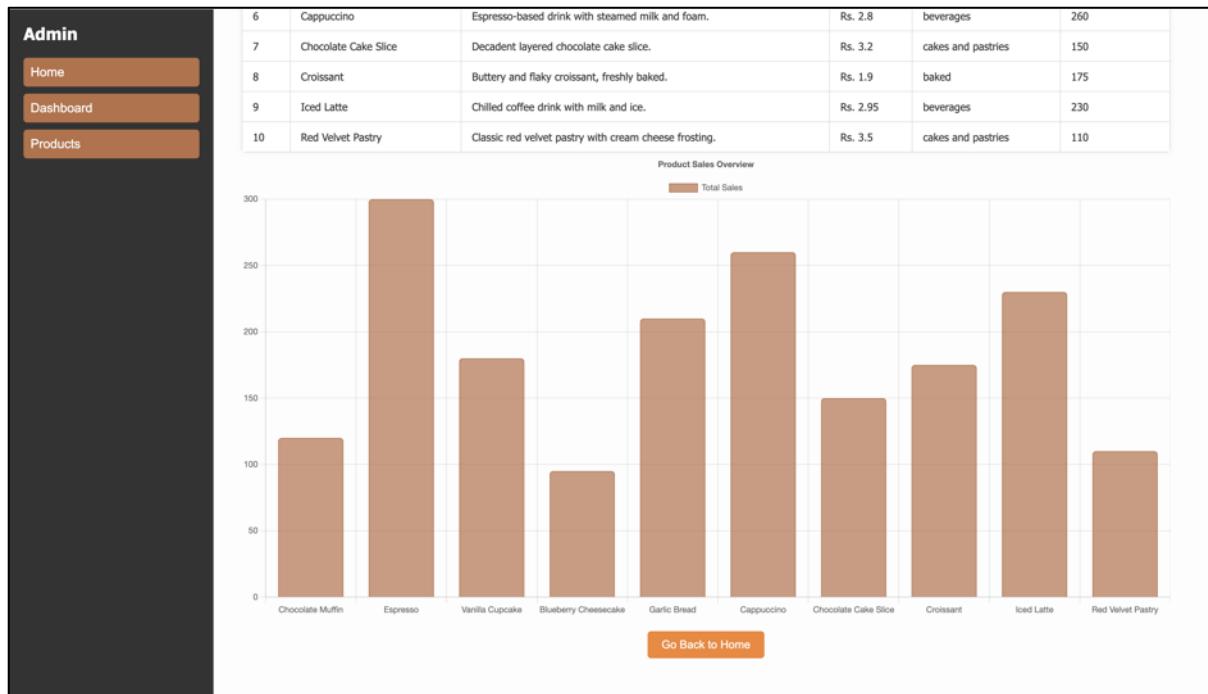


Figure 186: bar graph in admin dashboard

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/>