Chapter - 1

1. Introduction

1.1. Introduction of the System.

1.1.1. Project Title

"OnShop" (E-commerce web application)

1.1.2. Category

Web Application using RDBMS & REST APIs.

1.1.3. Overview

E-commerce is fast gaining ground as an accepted and used business paradigm. More and more business houses are implementing websites providing functionality for performing commercial transactions over the web. It is reasonable to say that the process of shopping on the web is becoming commonplace. This project aims to develop an e-commerce store for a hypermarket store where products can be bought from the comfort of home through the Internet. However, for implementation purposes, this paper will deal with online shopping. An online store is an online store where customers can browse the catalogue and select products of interest. The selected items may be collected in a shopping cart. At checkout time, the items in the shopping cart will be presented as an order. At that time, more information will be needed to complete the transaction. Usually, the customer will be asked to fill or select a billing address, a shipping address, a shipping option, and payment information such as a credit card number.

1.2. Background

1.2.1. Introduction of the Company

~~This Project is built for ……. For selling their products online.OnShop is to help customers by providing all kinds of stationary and related products related information on the website. It enables customers to purchase products online.~~

1.3. Objectives of the System

- To develop an optimised and fast website using which customers can easily browse products and place an order.
- To provide a solution to reduce and optimize the expenses of customer order management.
- To create an avenue where people can shop for products online.
- To sell the products using online payment.
- To increase revenue through e-commerce.
- To develop a database to store information about products and customers.

1.4. Scope of the System

Every project is done to achieve a set of goals with some conditions keeping in mind that it should be easy to use, feasible and user friendly. As the goal of this project is to develop an online system to sell products, this system will be designed keeping in mind the conditions (easy to use, feasibility and user friendly) stated above. It may help in effective and efficient order management. In every short time, the collection will be obvious, simple and sensible. It is very possible to observe the customer potential and purchase patterns because all the ordering history is stored in the database. It is efficiently managing all the operations of an online store within a single platform. The project aims to automate the business process of the Family Hypermarket store.

1.5. Structure of the System

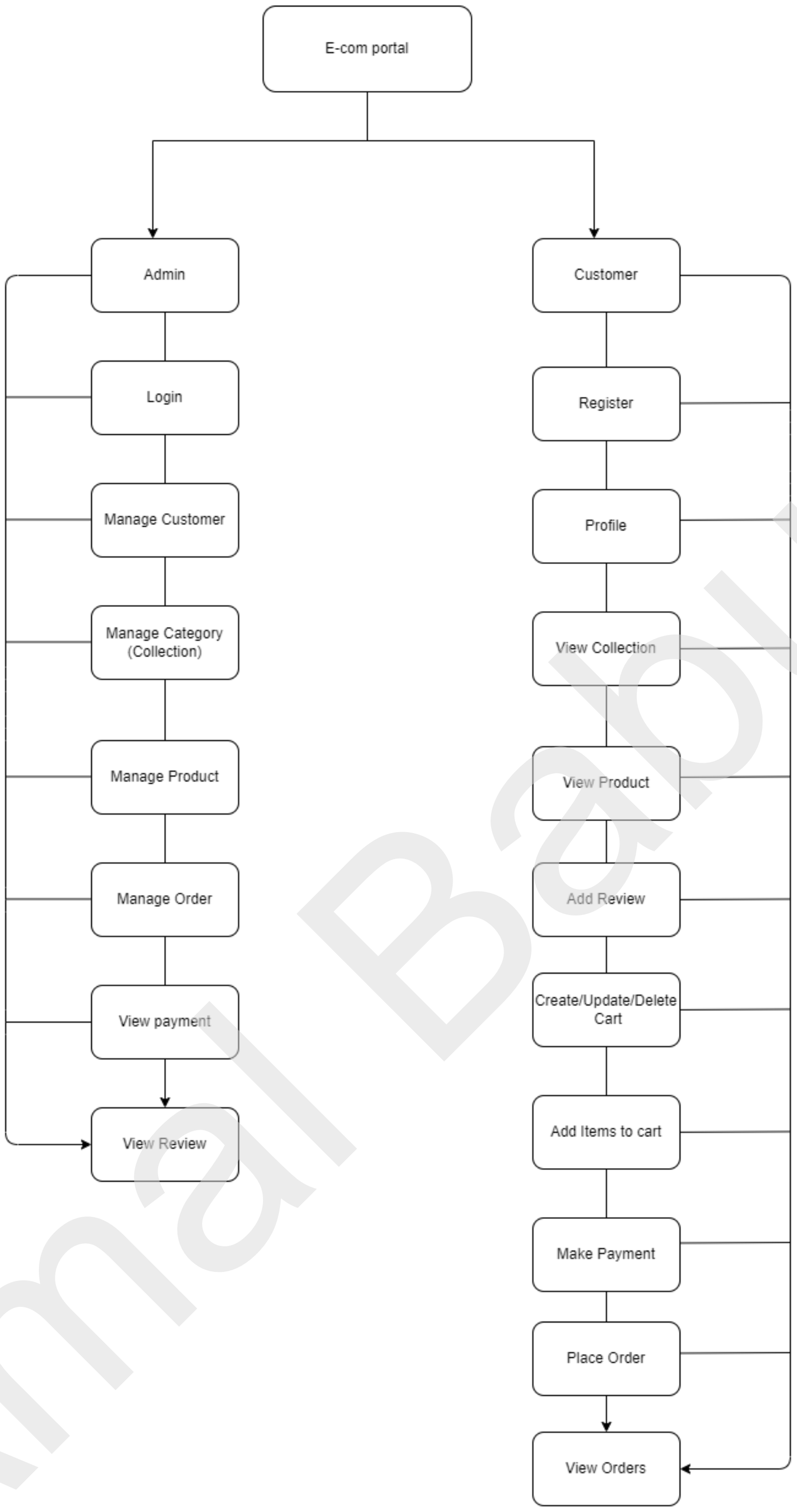


Fig. 1.1: Structure Chart

1.6. System Architecture

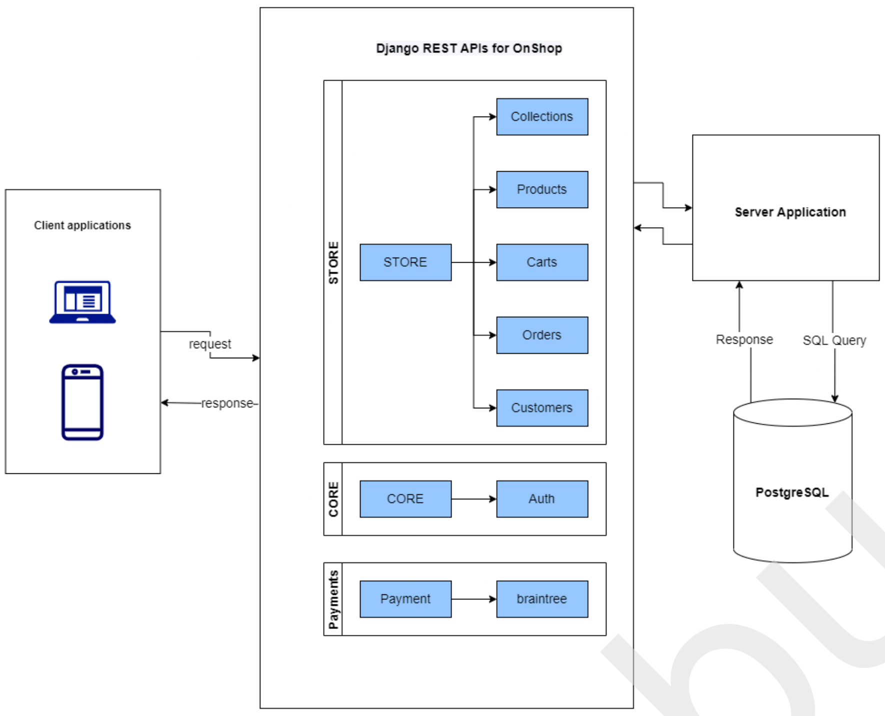


Fig. 1.2: System Architecture

1.7. End Users

- **Admin**
- **Customer**

1.8. Software/Hardware used for the development

Software requirement (for Development)

- **Frameworks**: Django 4.0 & Django REST framework
- **Front-end libraries/frameworks** :ReactJS, ReduxJS, MaterialDesignBootstrap.
- **Backend language** : Python, javaScript
- **Front-end languages**: JS, JSX, HTML, CSS.
- **Database**: PostgreSQL
- **Code editor**: VS code.

Hardware requirement (for Development)

- **Processor** – Intel Dual Core
- **RAM** – Minimum 4GB
- **Hard Disk** - Minimum 40GB (SSD)
- **Keyboard, Monitor, Mouse**

1.9. Software/Hardware used for the implementation

Software requirement (for implementation)

- **Database**– PostgreSQL
- **Language** – Python, JS, JSX
- **Browser** : Chrome, Opera, Firefox, Microsoft Edge

Hardware requirement (for implementation)

- **Processor** – Intel Dual Core
- **RAM** – Minimum 2GB
- **Hard Disk** - Minimum 40GB

****

Chapter - 2

2. System Requirement Specification

2.1. Introduction

System Requirements Specification is a structured collection of information that embodies the requirements of a system. A business analyst, sometimes titled system analyst, is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions.

System specification describes the operational and performance requirements of a system, such as a computer. It is considered a high-level document that dictates global functions. System specifications help to define the operational and performance guidelines for a system. An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations.

2.2. Overall Description

2.2.1. Product perspective

With the ever-increasing popularity and accessibility of the internet, it is only natural that the educational community should want to make use of this tremendous resource. The administrative user interface concentrates on the consistent information that is practically part of the organizational activities and which needs proper authentication for the data collection. The Interface helps the administration with all the transactional states like data insertion, data deletion, and data updating along with executive data search capabilities.

The project titled "on Shop" is a self-contained product developed as per the client's definition and requirements. The system contains all the necessary components required for selling and buying products in the form of online and offline payments. The system is a platform-independent and secured one.

2.2.2. Product Functions:

Project OnShop has following product functions:

- **Admin dashboard:** It views all the entities in the system. Admin performs overall observation of the system using the admin dashboard, admin can add new products, and customer manage orders.
- **Register:** Admin, customers and staff need to register with the system by providing appropriate information.
- **Login:** Once the User is registered then they can login using an email and password.
- **Get Products:** This function allows the customer products page and shop the products.
- **Search Products**: filter products by title and description.
- **Sort Product:** Sort products by price range
- **Filter Products:** This function allows the customer to filter products based on product category and price.
- **Customer account:** Customer can view update name and password
- **Add to cart** : This function allows users to add items to cart.
- **Payment:** Customers can make payment online.
- **Place order:** Customers can place orders with cart id and payment payload.

2.2.3. User characteristics

Table 2.1 : User characteristics

User	**Description**
Admin (Super user)	Admin will login to the system. He/She has a dashboard to trace all the activities of the project. The main function of the admin is to view and manage all the entities of the project.
Customer	Customers can register and need to login and purchase the available products through an online or offline payment.

2.2.4. General constraints

- Database is password protected.
- Should use less RAM and processing power.
- Each user should have individual username and password.
- Only superuser can access the whole system.

2.2.5. Assumptions

- Each User must have a Username and password.
- There are multiple admin users.
- Proper browsers should be installed.
- Data should be properly connected to the browser.

2.3. Functional Requirements

2.3.1. Admin Dashboard

- **Create Group**

  Function: Admin can create group of users with permissions.

  Input: Group name and choose permissions from list.

  Output: Group added message and navigate to groups page.

- **Add collection**

  Function: Add collection (category) of products.

  Input: collection title.

  Output: Collections page with success message.

- **Add Products**

  Function: Admin or staff can add new products with product details as input.

- **Add User.**

  Function: Admin can add new user directly from admin dashboard with user details.

- **View payment details.**

  Function: Admin can view payment details.

Stimulus & Response

Stimulus: Admin request for the login page.

Response: The login page is displayed.

Stimulus: Admin enters username, Password and clicks on

login button.

Response: Admin page is displayed if Username and Password are correct else error message is displayed.

Stimulus: Admin clicks the addCategory button.

Response: Add category form displayed.

Stimulus: Admin enters required fields and clicks the submit button.

Response: added categories are displayed.

Stimulus: Admin clicks Add new products Button.

Response: Add Product form is displayed.

Stimulus: Admin manages product details.

Response: requested product details are displayed.

Stimulus: The admin clicks the edit product button.

Response: Edit product form is displayed.

Stimulus: Admin enters required fields and clicks the submit button.

Response: Edited products displayed.

Stimulus: Admin manages customer orders.

Response: The requested orders are displayed.

Stimulus: Admin manages customer details.

Response: requested details are displayed.

2.3.2. Customer

- **Register(signup)**

  Function: Customer can register with by providing valid details.

- **Login**

  Function : Customer should login with username and password as input.

- **Add to cart**

  Function: add products to the cart by passsing cart_id and product id as input.

- **Make Payment**

  Function: make payment, by giving amount and payment method as input.

- **Place Order**

  Function: Place the order. Inputs are cart_id, payment_details.

Stimulus & Response

Stimulus: Customer clicks login Button.

Response: Login page id displayed.

Stimulus: Customer enters username and Password and clicks LoginButton

Response: Products home page is displayed if Username and Password are correct else display the error message.

Stimulus: Customer chooses a product and clicks view

Response: Selected products displayed with name, price and rating.

Stimulus: Customer click cart button

Response: Cart is displayed

Stimulus: Customer chooses payment method and clicks buy button.

Response: The product will be selected and billed to the customer.

Stimulus: The customer clicks the view profile button.

Response: Customer profile displayed with edit option.

Stimulus: Customer clicks the edit profile button

Response: The profile edit form is displayed.

Stimulus: Customer enters required fields and clicks submit button

Response: profile details evaluated. if error, an error message is displayed. otherwise, a profile updated notification is displayed

2.4. Design Constraints

- All the inputs should be checked for validation and messages should be given for the improper data. The invalid data are to be ignored and error messages should be given.
- While adding the product details to the system, mandatory fields must be checked for validation whether the admin has filled appropriate data in these mandatory fields. If not, proper error message should be displayed or else the data is to be stored in database for later retrieval.
- All mandatory fields should be filled by admin, while adding the customer detail into the database.

2.5. System Attributes

The Quality of the website is maintained in such a way so that it can be very user friendly to all the users of the website.

- **Reliability:** Good validation of user inputs will be done to avoid entering incorrect username and password.
- **Availability:** The system shall be available all the time.
- **Security:** Each time there is a security violation, System restricts the user from accessing that function.
- **Maintainability:** The ability to maintain, modify information and update fix problems of the system.
- **Portability:** This system can be run in any operating system and browser.
- **Accessibility:** Administrator and many other users can access the system but the access level is controlled for each user according to their work scope.

2.6. Other Requirements

2.6.1. Performance Requirements

- **Response time:** The system will give responses within 1 second after the checking of the customer information and other-information.
- **Capacity:** The system must support 100 people at a time.
- **User Interface:** Frontend application built with a single page application framework(ReactJs).

2.6.2. Safety and Requirements

- The database may get crashed at any certain time due to virus or operating system failure or mishandling the system, therefore, it is required to take the database backup.

2.6.3. Security Requirements

- The backend build with most secure framework Django and Django REST, Django have inbuilt mechanism to prevent security vulnerabilities like CSRF.
- This application using JWT authentication, the expired access tokens are blacklisted.

****

Chapter – 3

3. System Design

3.1. Introduction

Systems design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. Systems design implies a systematic approach to the design of a system. It may take a bottom-up or top-down approach, but either way the process is systematic wherein it takes into account all related variables of the system that needs to be created from the architecture, to the required hardware and software, right down to the data and how it travels and transforms throughout its travel through the system.

This phase will take the project to one step ahead as SRS includes only the project explanation theoretically. Once the requirements of the system design are analysed, the system will go from a pictorial representation stage to a theoretical stage. Hence the features and modules of the system are pictured in the form of diagrams for better understanding.

3.2. Assumption and Constraints.

3.2.1. Assumptions

- Each User must have a Username and password.
- Proper browsers should be installed.
- Data should be properly connected to the browser.

3.2.2. Constraints

- Database is password protected.
- Should use less RAM and processing power.
- Each user should have an individual username and password.

3.3. Functional decomposition.

Functional decomposition refers broadly to the process of resolving a functional relationship into its constituent parts in such a way that the original can be reconstructed from those parts by function components. In general, this process of decomposition is undertaken either for the purpose of gaining insight into the identity of the constituent components (which may reflect individual physical process of interest) or for the purpose of obtaining a compressed representation of the global function, a task which is feasible only when the constituent processes possess a certain level of modularity (i.e., independence or non-interaction).

3.3.1. System software architecture

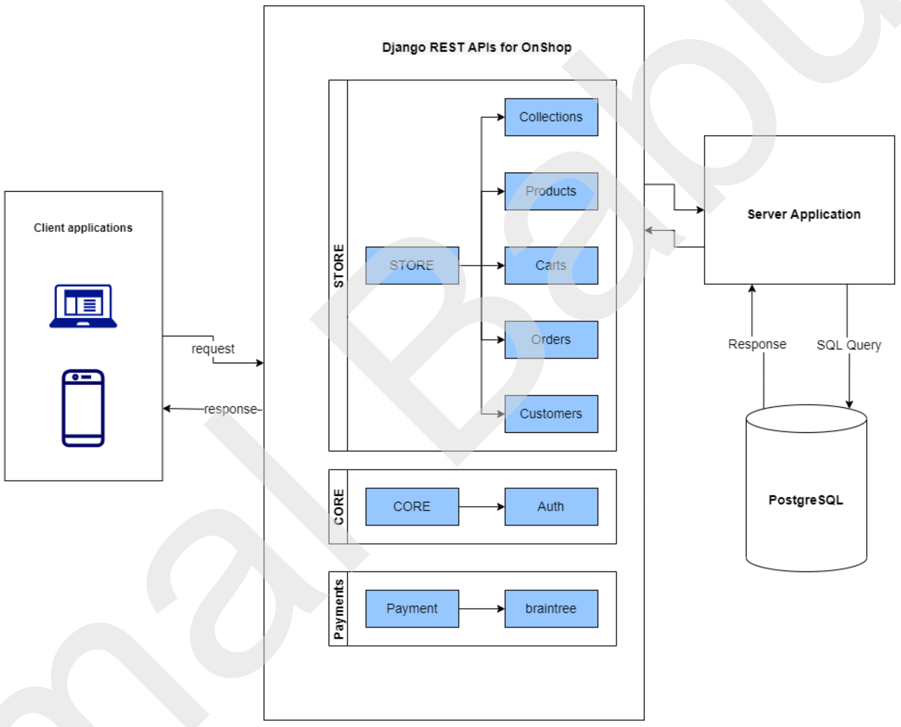


Fig. 3.1: System Software Architecture

3.3.2. System technical architecture

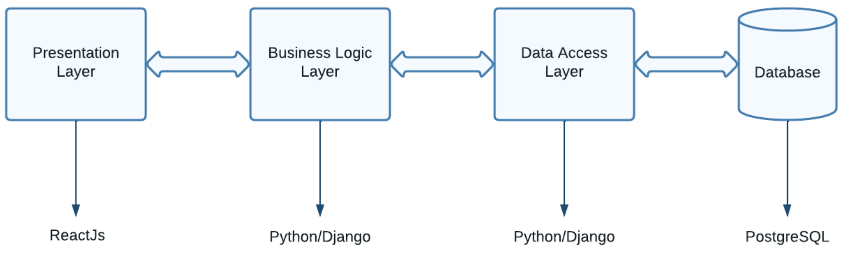


Fig. 3.2. System Technical Architecture

3.3.3. System hardware architecture

In software engineering, hardware architecture refers to the identification of a system's physical components and their interrelationships. This description, often called a hardware design model, allows hardware designers to understand how their components fit into a system architecture and provides to software component designers important information needed for software development and integration.

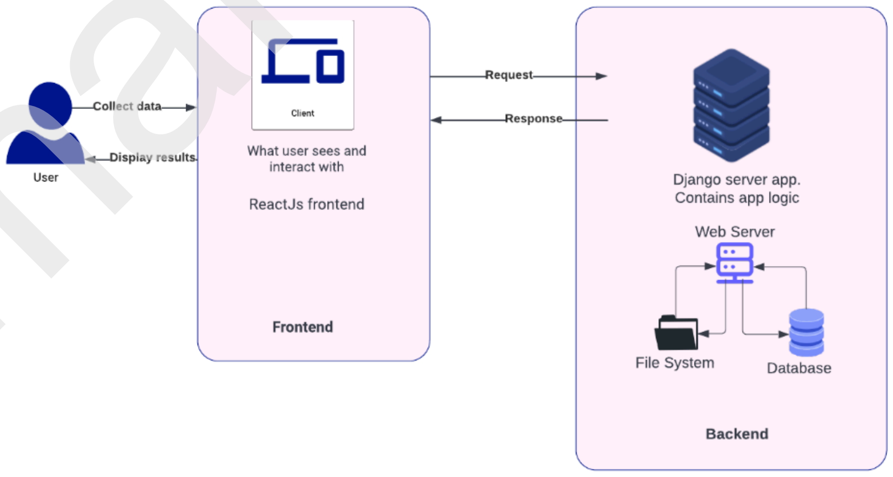


Fig. 3.3: System Hardware Architecture

3.3.4. External interface

Braintree payment gateway interface

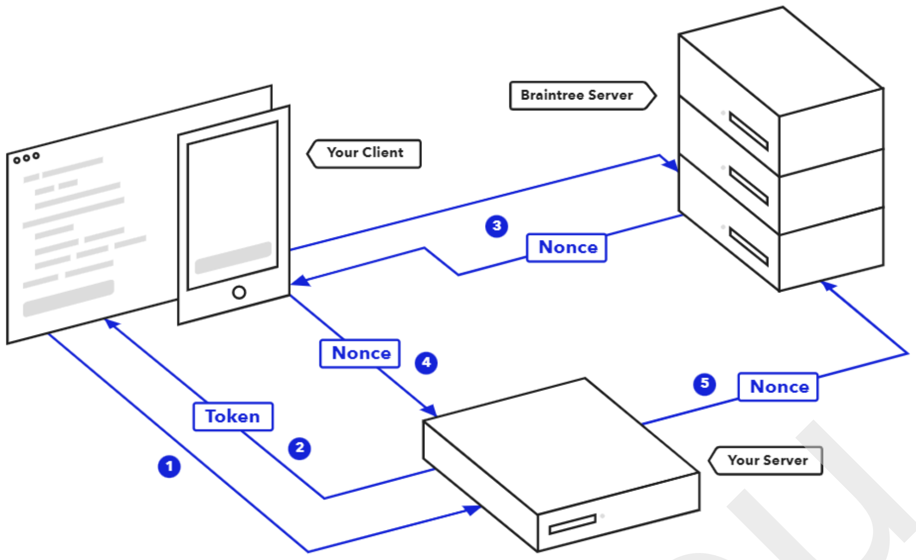
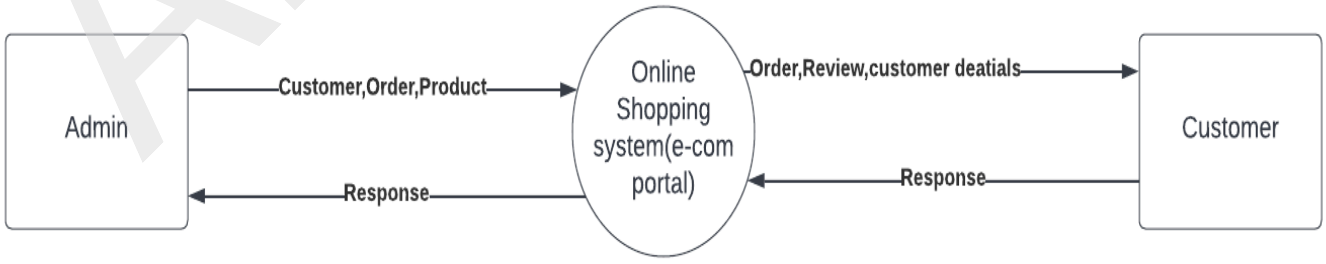


Figure 3.4 : braintree payment gateway interface

3.4. Description of Programs

3.4.1. Context Flow Diagram (CFD)

Context flow diagrams must be drawn, that is because this gives a brief description of the working of the system. The DFD illustrates the working of each module, whereas the context flow diagram below illustrates the communication between the different actors of the system with each other as well as with the database. The main function of each actor of the system is highlighted here and followed by which the DFDs are designed. Context flow diagram is also a DFF, but since it gives the overall description it is known as CFD.



Admin
Customer,Order,Product
Response
Online Shopping system(e-com portal)
Order,Review,customer deatials
Response
Customer

Fig. 3.5: Content Flow Diagram

3.4.2. Data Flow Diagrams (DFDs – Level 0, Level 1, Level 2)

Data Flow Diagram is a graphical representation of a system or a portion of the system. It consists of data flow, process, sources and sink and stores all the description through the use of easily understandable symbols. DFD is one of the most important modelling tools. It is used to model the system, components that interact with the system, and uses the data and information flows in the system. DFD shows the information moves through , and how it is modified by a series of transformations. It is a graphical representation that depicts information moves from input to output. DFD is also known as bubble charts or Data Flows Graphs. DFD may be used to represent the system at any level of abstraction.

Table 3.1: Symbols and Characteristics

Symbol	Description
◯	An oval or a circle symbol is used which will represent the process. The process includes the main factors or the working of the system.
▭	A rectangular box which is used to represent the source or sink. The source or sink describes the users of the system and this is connected to the process to represent the working of the system with the users.
	A straight line represents the flow of control. This is very important as it shows the connection between the process and the source or sink.
═	Open box or the parallel line symbol is used to represent the tables of the database. It is used to show the connection between each of the modules and the tables of the database in the system.

DFD Level – 1 Admin

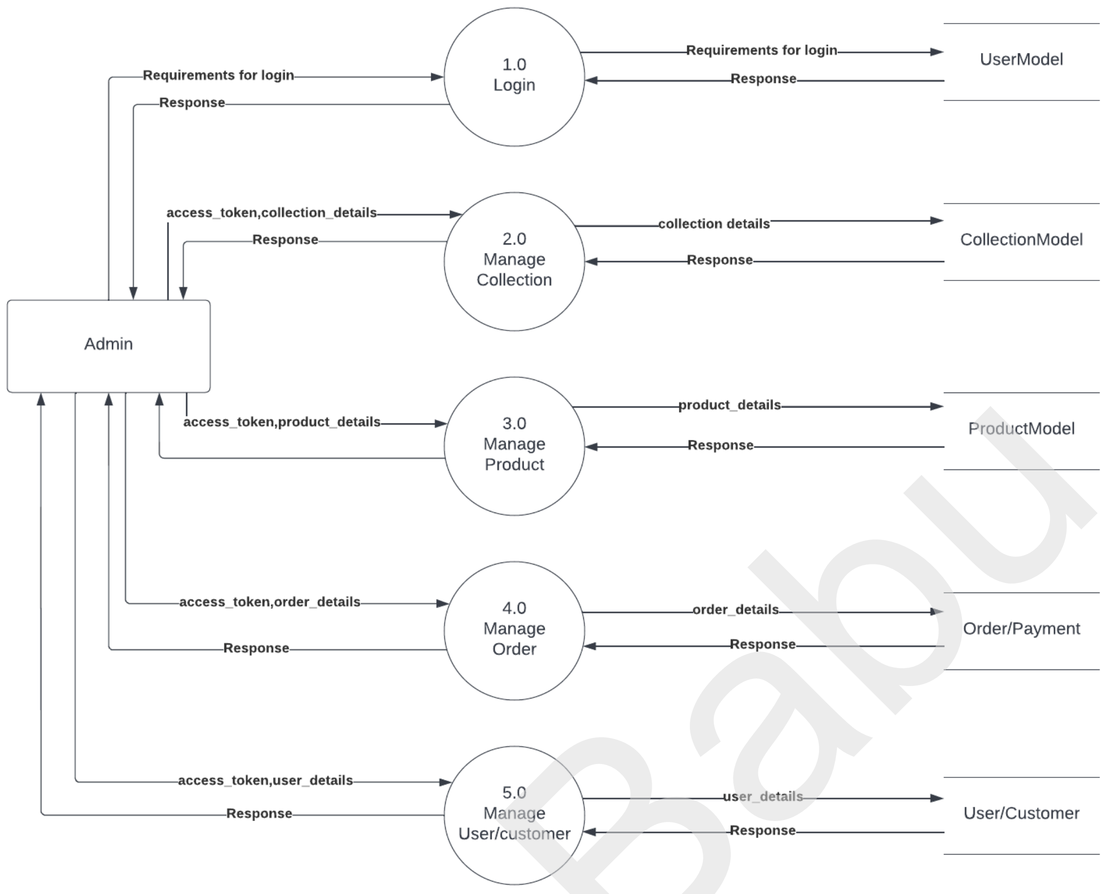


Fig.3.6: level-1 DFD admin

DFD Level – 2 Admin(3.0)

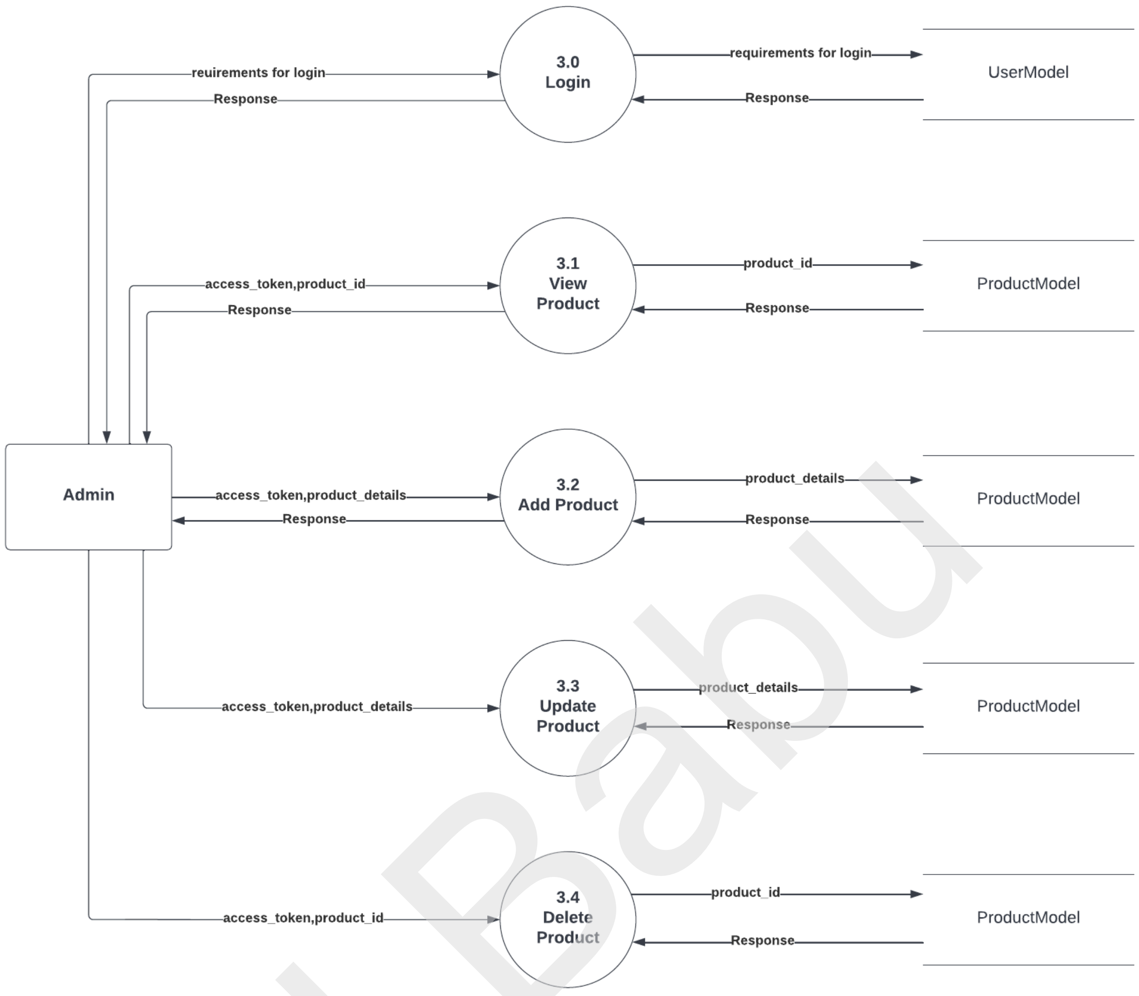


Fig.3.7: level-2 DFD admin(3.0) manage product

DFD Level – 2 Admin(4.0)

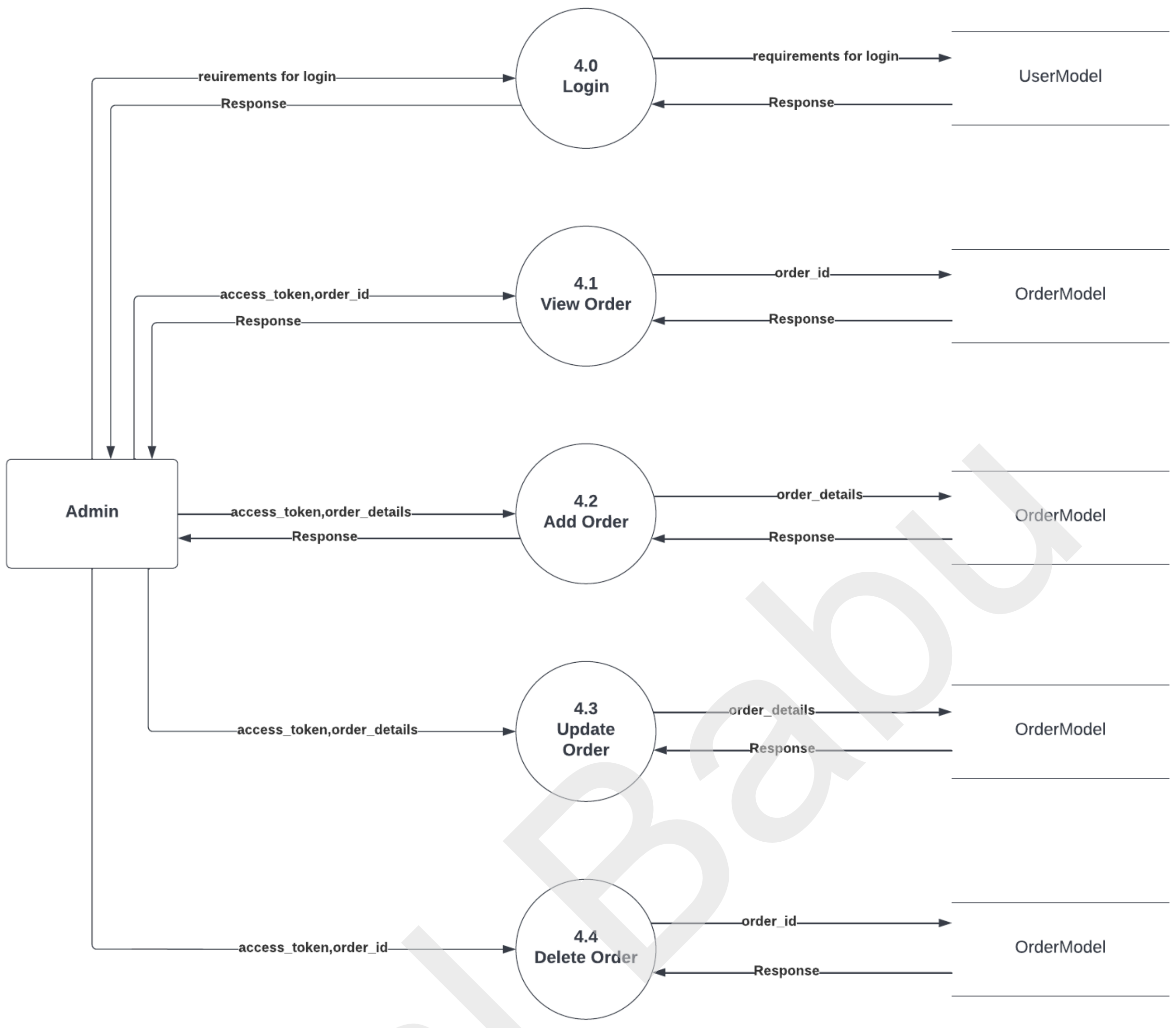


Fig. 3.8: Level 2 DFD admin(4.0) manage order

DFD Level – 1 Customer

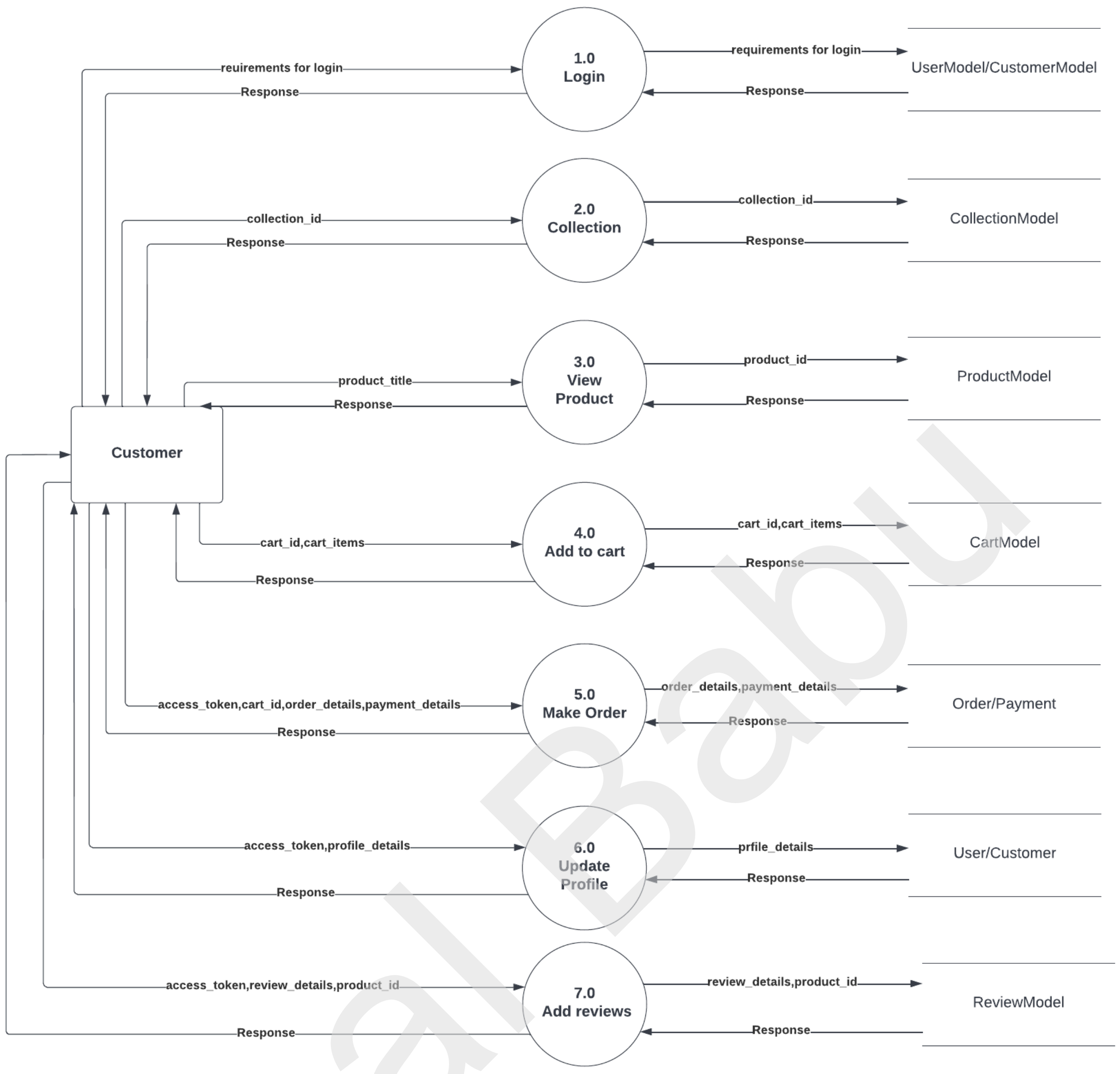


Figure 3.9 : Level 1 customer

DFD Level – 2 Customer(5.0)

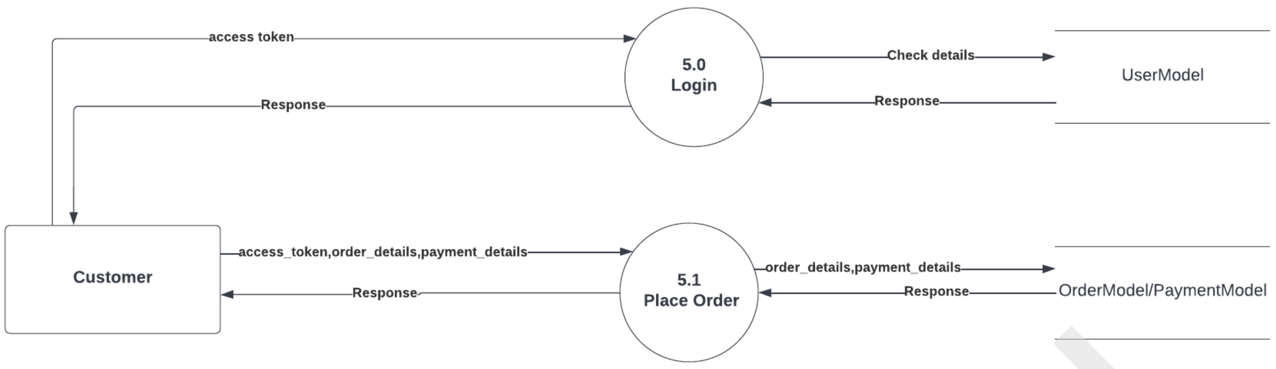


Figure 3.10: Level 2 customer(5.0)

DFD Level – 2 Customer(6.0)

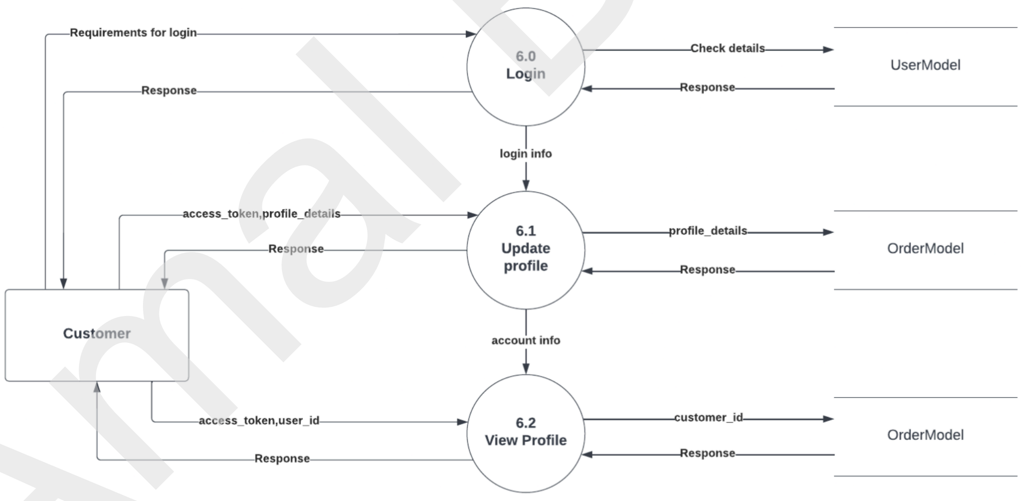


Figure 3.11 Level 2 customer

3.5. Description of components.

3.5.1. View Customer

- **Input:** Customer uploads his id and name
- **Process:** Fetching customer details from the Customer database
- **Output:** Details are displayed.

3.5.2. Add Products

- **Input**: Enter product name,category, quantity and price
- **Process**: Products are stored in a database.
- **Output**: Products are visible for customers to purchase.

3.5.3. Get Products

- **Input** : Send a get request for products with product title or category.
- **Process:** fetching the data from the database.
- **Output:**Fetched data displayed.

3.5.4. Add to cart

- **Input** :Customer selects products and requests for add to cart.
- **Process:** The cart details are stored in the database.
- **Output:** cart details are displayed.

3.5.5. Make order

- **Input**: Products are booked by customers with cart id.
- **Process**: Orders stored in database .
- **Output**:Order details are displayed with payment status.

****

Chapter - 4

4. Database Design

4.1. Introduction

Database: A Database is collection of related data, which can be of any size and complexity. By using the concept of Database, we can easily store and retrieve the data. The major purpose of a database is to provide the information, which utilises it with the information that the system needs according to its own requirements.

Database Design: Database design is done before building it to meet needs of end-users within a given information-system that the database is intended to support. The database design defines the needed data and data structures that such a database comprises. The database is physically implemented using SQL(PostgreSQL).

4.2. Purpose and scope

The main purpose of developing an OnShop application is helping the customers to purchase the products online. After the login, customers can place the order in the cart and make the online payment. The main agenda of the Online SuperMarket is to set up a portal where the customer can choose their vegetables, fruits, bakery products,stationary, branded products etc online without having to visit the shop physically. The current system is an offline system. More physical interaction takes place in this current system between Customer and administrator. Before ordering a product , customer preview cannot be done in the existing system. New online system will solve all the issues because online designers are implemented in the project and companies can expand their business all over the world.

4.3. Database Identification

- Database table name and column names are defined without leaving space.
- Lowercase used to create database tables and columns.

4.4. Schema information

A schema is the structure behind data organisation. It is a visual representation of how different table relationships enable the schema's underlying mission business rules for which the database is created. In a schema diagram, all database tables are designated with unique columns and special features, e.g., primary/foreign keys or not null, etc. Formats and symbols for expression are universally understood, eliminating the possibility of confusion. The table relationships also are expressed via a parent table's primary key lines when joined with the child table's corresponding foreign keys. Schema diagrams have an important function because they force database developers to transpose ideas to paper. This provides an overview of the entire database, while facilitating future database administrator work.

<u>**These are:**</u>

- Physical Schema
- Logical Schema
- View Schema

A physical schema can be defined as the design of a database at its physical level. In this level, it is expressed how data is stored in blocks of storage. A logical schema can be defined as the design of the database at its logical level. In this level, the programmers as well as the database administrator work. At this Level, data can be described as certain types of data records which can be stored in the form of data structures. However, the internal details will be remaining hidden at this level. View schema can be defined as the design of the database at view level which Generally describes end-user interaction with database systems.

4.5. Table Definition

Table Name : core_user

Description : This table stores all user information including admin,staff customers

Table 4.1 core_user table

Column	Data Type	Length/Precision	Constraints	Description
id	bigint		Primary Key, Not NULL	User id
password	character varying	128	Not NULL	User password
last_login	timestamp			User last login date and time
is_superuser	boolean		Not NULL	is the user superuser or not.
username	character varying	20	Not NULL	User unique name
first_name	character varying	20	Not NULL	First name of user
last_name	character varying	25	Not NULL	Last name of user
is_staff	boolean		Not NULL	True if user is a staff otherwise false
is_active	boolean		Not NULL	True then the user is active
date_joined	timestamp			User joined date

email	character varying	50		User email address

Table Name : store_customer

Description : This table stores customer information.

Table 4.2 store_customer table

Column	Data Type	Length/Precision	Constraints	Description
id	uuid		Primary Key	Customer id
phone	character varying	14		Customer contact number
birth_date	date			Date of birth of customer
membership	character varying	1	Not NULL	User membership value
user_id	bigint		Foreign Key	

Table Name : store_collection

Description : This table stores all product category details.

Table 4.3 store_collection table

Column	Data Type	Length/Precision	Constraints	Description
id	bigint		Primary Key	Collection id
title	character varying	50	Not NULL	Collection title name

Table Name : store_product

Description : This table stores all product details.

Table 4.4 store_product table

Column	Data Type	Length/Precision	Constraints	Description
id	bigint		Primary Key	Product id
title	character varying	50	Not NULL	Product name
description	text			Product description
unit_price	numeric	6	Not NULL	Price of the product
inventory	integer	5	Not NULL	Product inventory
last_update	timestamp		Not NULL	Last updated date
collection_id	bigint		Foreign Key	Parent table relation

Table Name : store_cart

Description : This table stores cart details.

Table 4.5 store_cart

Column	Data Type	Length/Precision	Constraints	Description
id	uuid		Primary Key	Cart id
created_at	timestamp		Not NULL	Created date

Table Name : store_cartitem

Description : This table stores cart items.

Table 4.6 store_cartitem

Column	Data Type	Length/Precision	Constraints	Description
id	uuid		Primary Key	Cart item id
quantity	smallint		Not NULL	Product quantity
cat_id			Foreign Key	
product_id	bigint		Foreign Key	

Table Name : store_order

Description : This table stores orders.

Table 4.7 store_order

Column	Data Type	Length/Precision	Constraints	Description
id	uuid		Primary Key	order id
placed_at	smallint		Not NULL	Order placed date
is_shipped			Not NULL	
is_delivered	bigint		Not NULL	
is_cancelled			Not NULL	
customer_id			Foreign Key	
total_price	numeric	10	Not NULL	

Table Name : store_orderitem

Description : This table stores order items.

Table 4.8 store_orderitem

Column	Data Type	Length/Precision	Constraints	Description
id	uuid		Primary Key	Order item id
quantity	smallint		Not NULL	Item quantity
unit_price	numeric	6	Not NULL	
order_id	bigint		Foreign Key	
product_id	bigint		Foreign Key	

Table Name : payment_payment

Description : This table stores payment details.

Table 4.9 payment_payment

Column	Data Type	Length/Precision	Constraints	Description
id	uuid		Primary Key	
total_amount	numeric	6	Not NULL	Total paid amount
payment_method	character varying	3	Not NULL	
payment_status	character varying	1	Not NULL	
transaction_id	character varying	50	Not NULL	
username	character varying	30	Not NULL	
created_at	timestamp		Not NULL	Created date and time

order_id	bigint		Foreign Key	

Table Name : store_productimage

Description : This table stores product image details.

Table 4.10 store_productimage

Column	Data Type	Length/Precision	Constraints	Description
id	uuid		Primary Key	
image	character varying	100	Not NULL	Link of product image
product_id	bigint		Foreign Key	

Table Name : store_review

Description : This table stores product review details.

Table 4.11 store_review

Column	Data Type	Length/Precision	Constraints	Description
id	bigint		Primary Key	
name	character varying	30	Not NULL	Reviewer name
description	text		Not NULL	Link of product image
date	date		Not NULL	Created date
product_id	bigint		Foreign Key	

Table Name : store_address

Description : This table customers address details.

Table 4.12 store_address

Column	Data Type	Length/Precision	Constraints	Description
id	bigint		Primary Key	
street	character varying	50	Not NULL	Reviewer name
house_no	smallint		Not NULL	house/building number of customer
city	character varying	50	Not NULL	Link of product image
phone_no	character varying	12	Not NULL	
postal	smallint		Not NULL	Created date
customer_id	uuid		Foreign Key	

o

4.6. Physical Design

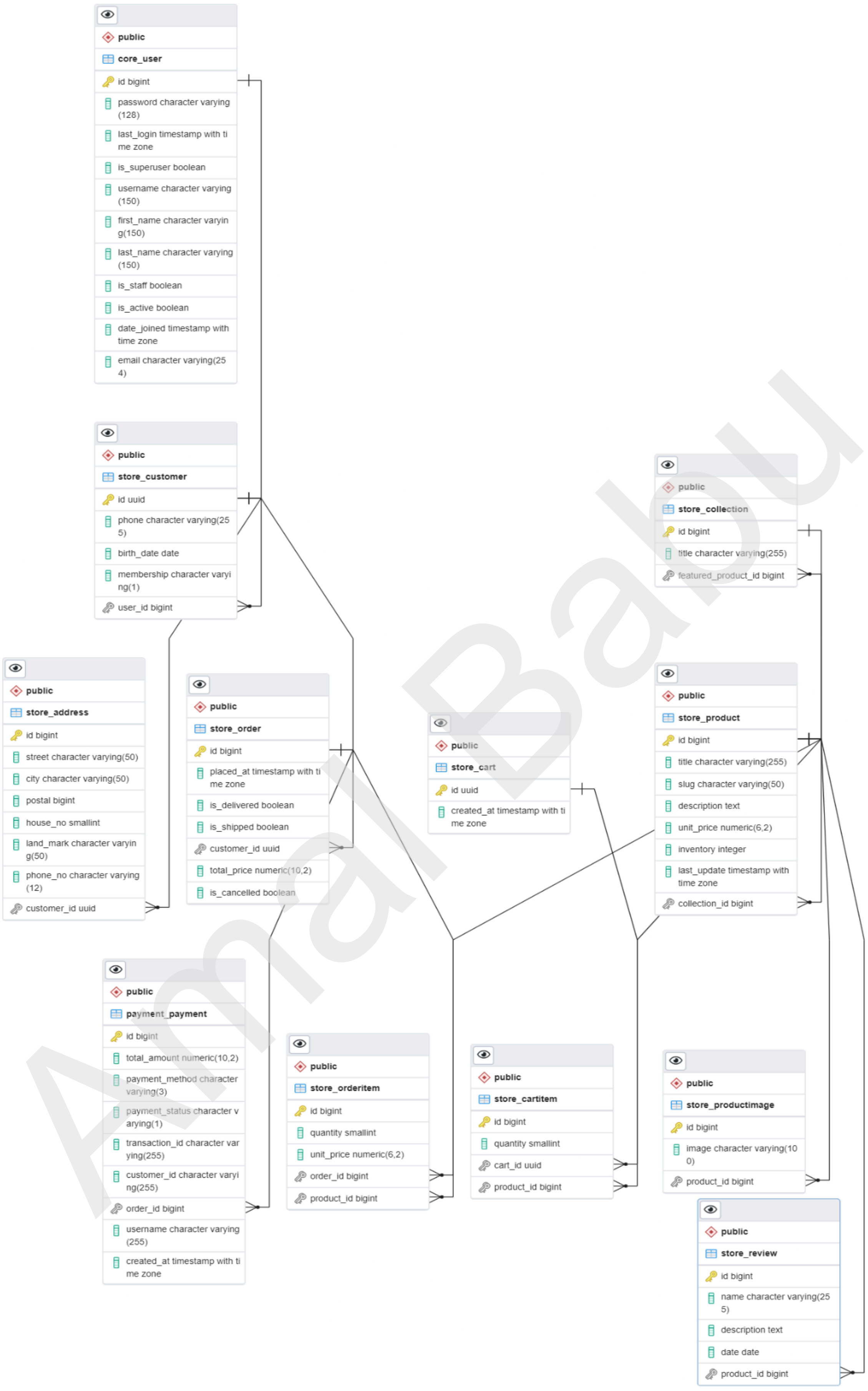

public
core_user
id bigint
password character varying (128)
last_login timestamp with time zone
is_superuser boolean
username character varying (150)
first_name character varying(150)
last_name character varying (150)
is_staff boolean
is_active boolean
date_joined timestamp with time zone
email character varying(254)
public
store_customer
id uuid
phone character varying(255)
birth_date date
membership character varying(1)
user_id bigint
public
store_collection
id bigint
title character varying(255)
featured_product_id bigint
public
store_address
id bigint
street character varying(50)
city character varying(50)
postal bigint
house_no smallint
land_mark character varying(50)
phone_no character varying (12)
customer_id uuid
public
store_order
id bigint
placed_at timestamp with time zone
is_delivered boolean
is_shipped boolean
customer_id uuid
total_price numeric(10,2)
is_cancelled boolean
public
store_cart
id uuid
created_at timestamp with time zone
public
store_product
id bigint
title character varying(255)
slug character varying(50)
description text
unit_price numeric(6,2)
inventory integer
last_update timestamp with time zone
collection_id bigint
public
payment_payment
id bigint
total_amount numeric(10,2)
payment_method character varying(3)
payment_status character varying(1)
transaction_id character varying(255)
customer_id character varying(255)
order_id bigint
username character varying (255)
created_at timestamp with time zone
public
store_orderitem
id bigint
quantity smallint
unit_price numeric(6,2)
order_id bigint
product_id bigint
public
store_cartitem
id bigint
quantity smallint
cart_id uuid
product_id bigint
public
store_productimage
id bigint
image character varying(100)
product_id bigint
public
store_review
id bigint
name character varying(255)
description text
date date
product_id bigint

Figure 4.1 : physical design diagram

4.7. Data Dictionary

A data dictionary can be seen as a repository for information about a database. There are no industry standards that go into a data dictionary. It may be as simple as a list of tables with basic descriptions. Alternatively, it can be an extensive list of properties outlining precisely how data is structured, maintained, and used.

Table 4.13 :core_user table

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	core_user	PK	id	bigint	NOT NULL	user id
public	core_user		password	character varying(128)	NOT NULL	user password
public	core_user		last_login	timestamp with time zone(6)	NULL	last login date
public	core_user		is_superuser	boolean	NOT NULL	true if the user is superuser
public	core_user		username	character varying(50)	NOT NULL	user unique name
public	core_user		first_name	character varying(50)	NOT NULL	first name of user
public	core_user		last_name	character varying(50)	NOT NULL	last name of user
public	core_user		is_staff	boolean	NOT NULL	true if the user is a staff
public	core_user		is_active	boolean	NOT NULL	user is active or not

public	core_user		date_joined	timestamp with time zone(6)	NOT NULL	joined data
public	core_user		email	character varying(50)	NOT NULL	email of user

Table 4.14: payment_payment table

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	payment_payment	PK	id	bigint(64)	NOT NULL	payment id
public	payment_payment		total_amount	numeric(6,2)	NOT NULL	total amount paid
public	payment_payment		payment_method	character varying(3)	NOT NULL	payment method
public	payment_payment		payment_status	character varying(1)	NOT NULL	payment status, payment pending or complete
public	payment_payment		transaction_id	character varying(50)	NULL	payment transaction id
public	payment_payment		customer_id	uuid	NOT NULL	customer id
public	payment_payment	FK	order_id	bigint(64)	NOT NULL	order id
public	payment_payment		username	character varying(255)	NULL	user name

Table 4:15 : store_address

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_address	PK	id	bigint(64)	NOT NULL	id
public	store_address		street	character varying(50)	NOT NULL	street of customer
public	store_address		city	character varying(50)	NOT NULL	customer city name
public	store_address		postal	smallint(16)	NOT NULL	post code
public	store_address		house_no	smallint(16)	NOT NULL	house number
public	store_address		land_mark	character varying(50)	NOT NULL	landmark of customer
public	store_address		phone_no	character varying(12)	NULL	contact number of customer
public	store_address	FK	customer_id	uuid	NOT NULL	customer id

Tabel 4.16 store_cart

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_cart	PK	id	uuid	NOT NULL	cart id
public	store_cart		created_at	timestamp with time zone(6)	NOT NULL	cart created date

Table 4.17 store_cartitem

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_cartitem	PK	id	bigint(64)	NOT NULL	cartitem id
public	store_cartitem		quantity	smallint(16)	NOT NULL	items quantity
public	store_cartitem	FK	cart_id	uuid	NOT NULL	parant table id
public	store_cartitem	FK	product_id	bigint(64)	NOT NULL	product id

Table 4.18 store_collection

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_collection	PK	id	bigint(64)	NOT NULL	collection id
public	store_collection		title	character varying(50)	NOT NULL	collection name

Table 4.19 store_customer table

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_customer	PK	id	uuid	NOT NULL	customer id
public	store_customer		phone	character varying(12)	NULL	customer contact number
public	store_customer		birth_date	date(3)	NULL	date of birth of customer
public	store_customer		membership	character varying(1)	NOT NULL	membership of customer
public	store_customer	FK	user_id	bigint(64)	NOT NULL	user id

Table 4.20 : store_order

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_order	PK	id	bigint(64)	NOT NULL	order id
public	store_order		placed_at	timestamp with time zone(6)	NOT NULL	order placed date and time
public	store_order		is_delivered	boolean	NOT NULL	order is delivered or not
public	store_order		is_shipped	boolean	NOT NULL	true if the order is shipped

public	store_order	FK	customer_id	uuid	NOT NULL	customer id
public	store_order		total_price	numeric(6,2)	NOT NULL	total amount to pay
public	store_order		is_cancelled	boolean	NOT NULL	is the order cancelled

Table 4.21 store_orderitem

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_orderitem	PK	id	bigint(64)	NOT NULL	order item id
public	store_orderitem		quantity	smallint(16)	NOT NULL	order item quantity
public	store_orderitem		unit_price	numeric(6,2)	NOT NULL	unit price of product
public	store_orderitem	FK	order_id	bigint(64)	NOT NULL	order id
public	store_orderitem	FK	product_id	bigint(64)	NOT NULL	product id

Table 4.22 store_product

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_product	PK	id	bigint(64)	NOT NULL	product id
public	store_product		title	character varying(50)	NOT NULL	product name
public	store_product		slug	character varying(50)	NOT NULL	auto generated name
public	store_product		descripti	text	NULL	description

			on			
public	store_product		unit_price	numeric(6,2)	NOT NULL	unit price of product
public	store_product		inventory	integer(32)	NOT NULL	inventory
public	store_product		last_update	timestamp with time zone(6)	NOT NULL	last updated date and time
public	store_product	FK	c	bigint(64)	NOT	collection id

			o l l e c t i o n _ i d		NULL	

Table 4.23 store_productimage

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_productimage	PK	id	bigint(64)	NOT NULL	Product image id
public	store_productimage		image	character varying(100)	NOT NULL	Product image link
public	store_productimage	FK	product_id	bigint(64)	NOT NULL	product id

Table 4.24 store_review table

schema_name	table_name	is_key	column_name	data_type	nullable	column_description
public	store_review	PK	id	bigint(64)	NOT NULL	review id
public	store_review		name	character varying(20)	NOT NULL	reviewer name
public	store_review		description	text	NOT NULL	review text
public	store_review		date	date(3)	NOT NULL	date review created
public	store_review	FK	product_id	bigint(64)	NOT NULL	date review created

4.8. ER diagram

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

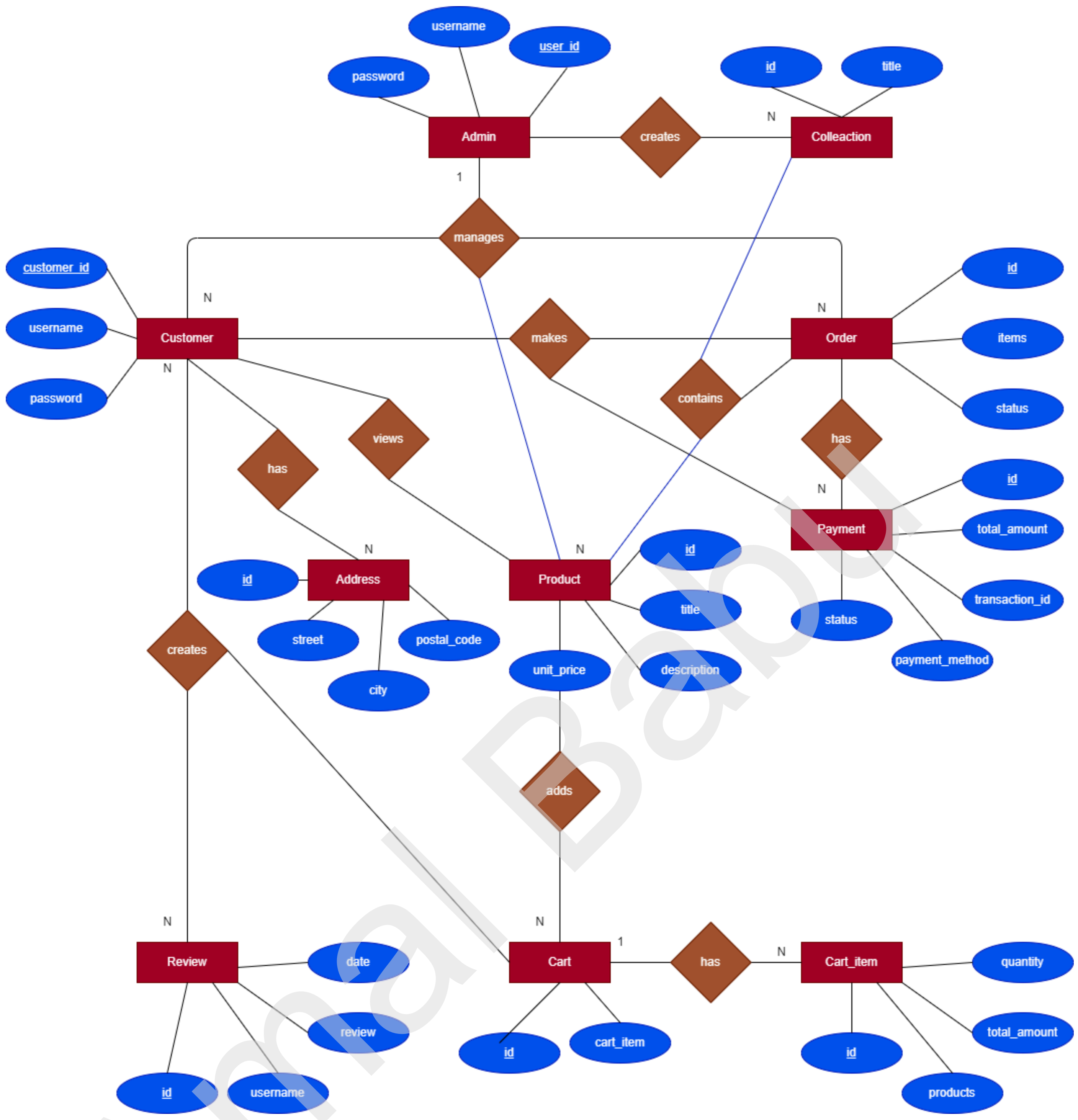


Fig. 4.2: ER Diagram

An entity-relationship (ER) diagram is a specialized graphic that illustrates the relationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities. Diamonds are normally used to represent relationships and ovals are used to represent attributes.

4.8.1. Entity:

Entity is represented by a box within the ER Diagram. Entities are abstract concepts, each representing one or more instances of the concept in question. An entity might be considered a container that holds all of the instances of a particular thing in a system. Entities are equivalent to database tables in a relational database, with each row of the table representing an instance of that entity

4.8.2. Relationship:

Relationships are represented by Diamonds. A relationship is a named collection or association between entities or used to relate to two or more entities with some common attributes or meaningful interaction between the objects.

4.8.3. Attributes:

Attributes are represented by Oval. An attribute is a single data item related to a database object. The database schema associates one or more attributes with each database entity

4.9. Database Administration

4.9.1. System information

- **Server:** localhost via TCP/IP
- **Server type:** Postgresql
- **Server version:** PostgreSQL 14.4
- **User**: localhost/admin
- **Server charset:** UTF-8 Unicode (utf8)

4.9.2. DBMS configuration

- **Version:** Postgresql 14.4
- **pgAdmin version :** 14
- **Postgresql connector:** psycopg2 2.9.3
- **Supported Operating System:** Windows 10,11

4.9.3. Support software required

- **Postgresql:** Postgresql is an open source relational database management system.emphasizing extensibility and SQL compliance.

- **pgAdmin:** pgAdmin4 is a popular application to manage Postgres databases. All types of PostgreSQL features are supported by this application.
- **Postgresql connector:** psycopg2 is required to connect postgresql and python.

4.9.4. Storage requirement

- The storage engine represents the heart of a Postgresql Server.
- Recovering the database from system failure
- Management of files and database pages used to store data
- Manage data buffers and system IO to the physical data pages
- Manage locking and concurrency issues

4.9.5. Backup and recovery

Database recovery is the process of restoring the databases to a correct state following a failure. The failure may be the result of a system crash due to hardware of software errors, a media failure, such as a head crash, or a software error in the application, such as a logical error in the program that is accessing the database. It may also be the result of unintentional or intentional corruption or destruction of data. Whatever the underlying cause of the failure, the DBMS must be able to recover from the failure and restore the database to a consistent state.

It is the responsibility of DBMS to ensure that the database is reliable and remains in a consistent state in the presence of failure. In general, backup and recovery refers to the various strategies and procedures involved in protecting the database against data loss and reconstructing the data such as that no data is lost after failure.

****

Chapter - 5

5. Detailed Design

5.1. Introduction

Detailed design is sometimes referred to as ‘developer design’. Detailed design is the second level of the design process. During detailed design, we specify how the module in the system interacts with each other and the internal logic of each of the modules specified during system design is decided; hence it is also called as logic design.

Detailed design essentially expands the system design and database design to contain a more detailed description of the processing logic and data structures so that the design is sufficiently complete of coding. The purpose of preparing this document is to explain complete design details of our Electronic Shop Management System. This design document is developer blueprint. During this phase design team uses both the requirement specification and the architecture specification provided by the previous phase to develop detailed design of the system.

5.2. Structure of the software package

Admin

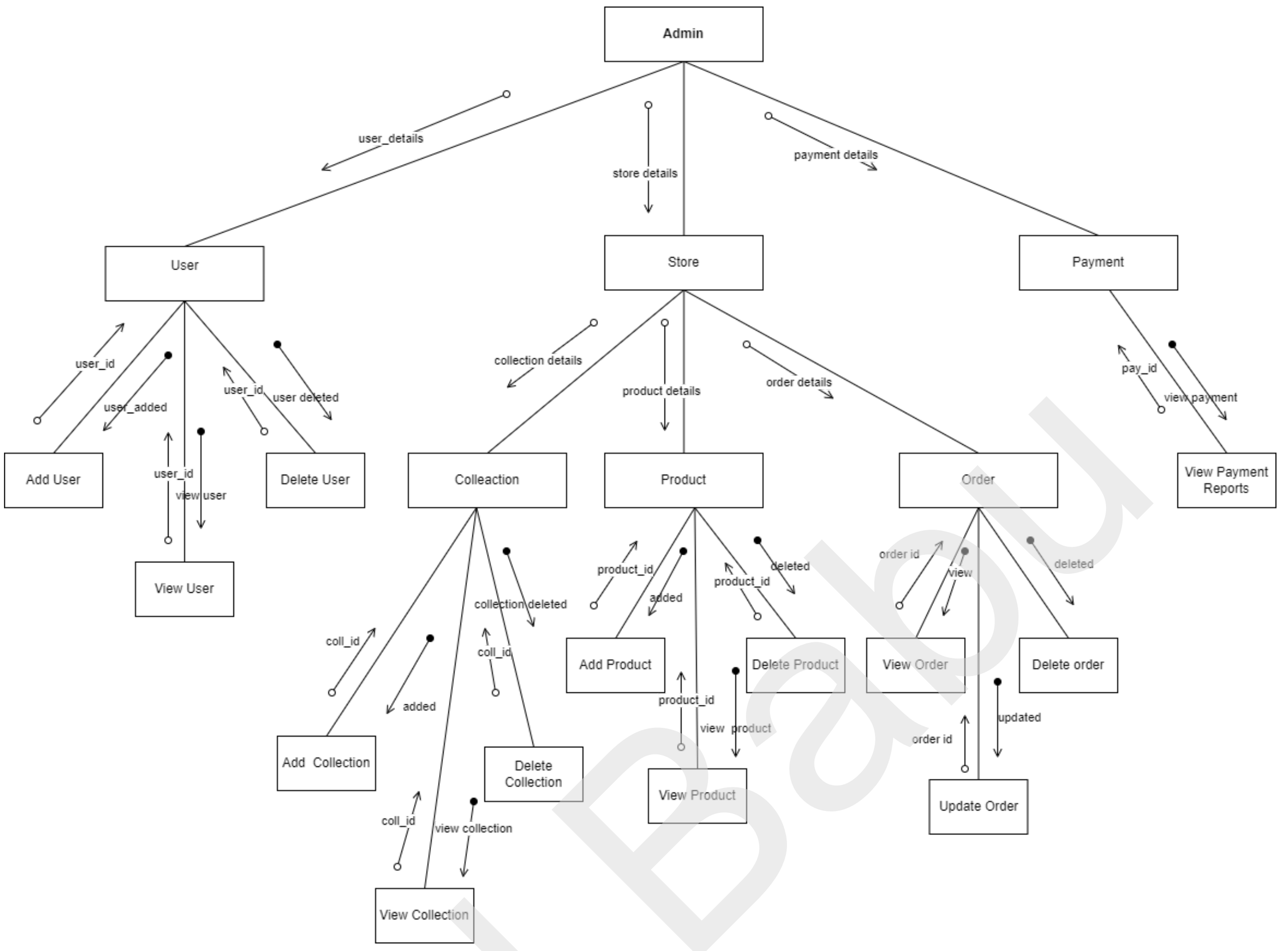


Figure 5.1 : structure chart admin

Customer

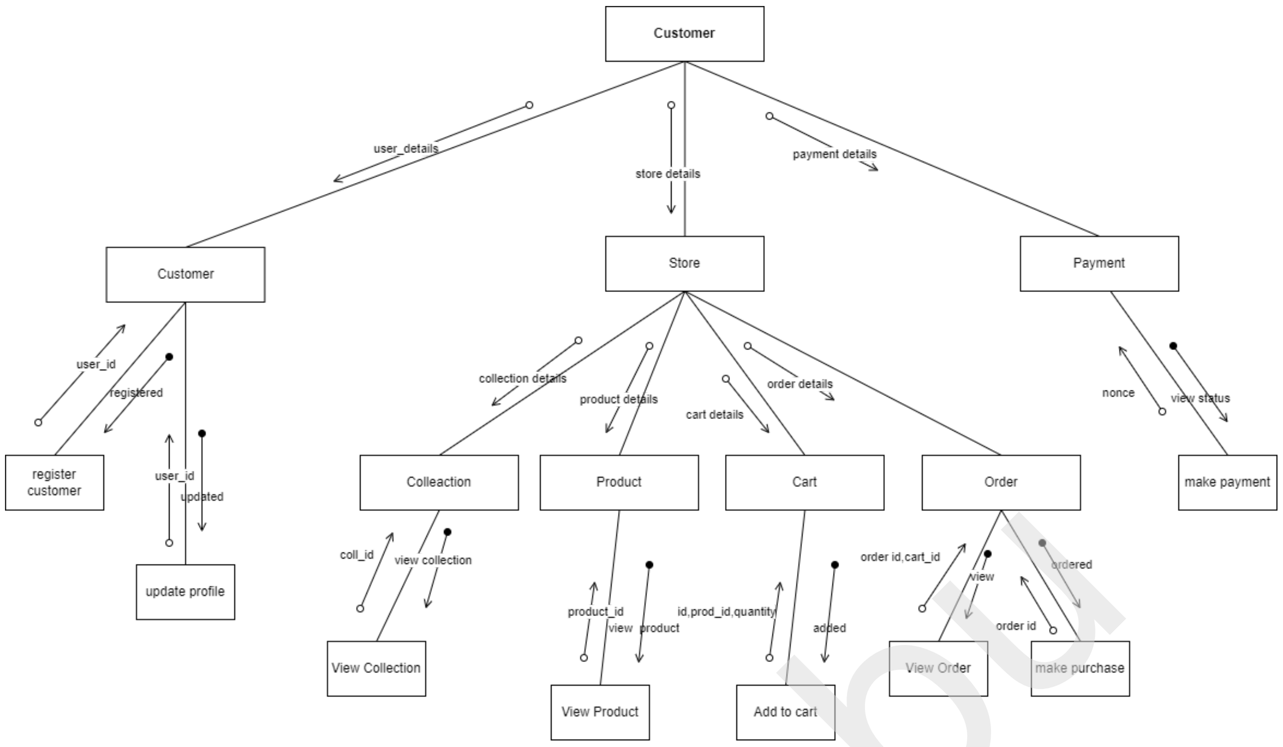


Figure 5.2 : structure chart customer

5.3. Modular decomposition of the System

5.3.1. Admin

5.3.1.1. Admin login

a) **Inputs:** Username,password

b) **Procedural Details (Flow Chart):**

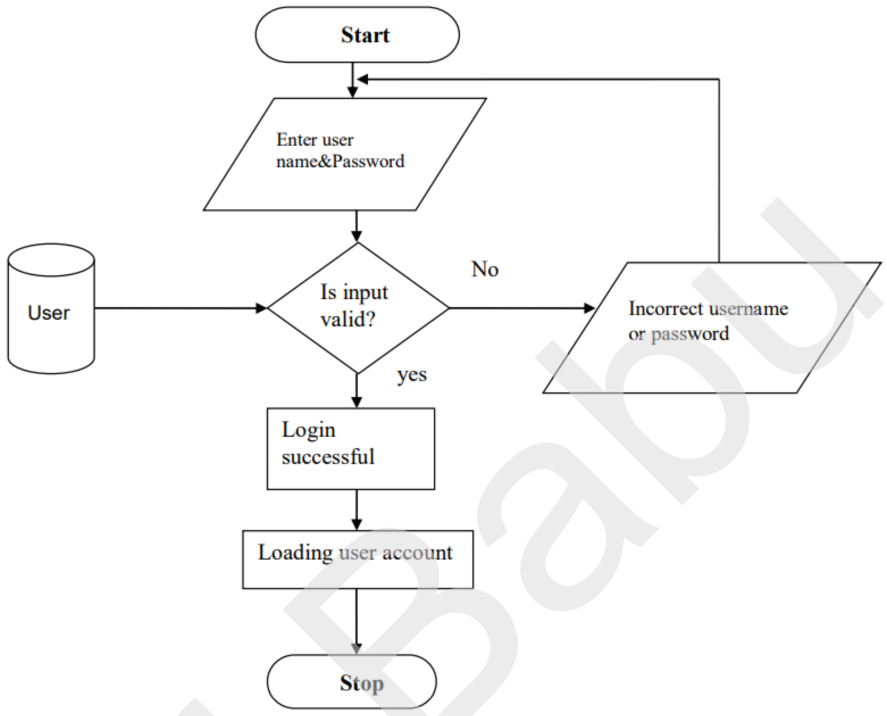


Figure 5.3 flowchart admin login

c) **File Input/output :** admin dashboard

d) **Output:** Entered Username and password will be checked for validity if it is valid Admin will be redirected to admin dashboard.

5.3.1.2. Admin add customer and user

a) **Input :** first_name, last_name, email, username,password

b) **Procedural Details (Flow Chart):**

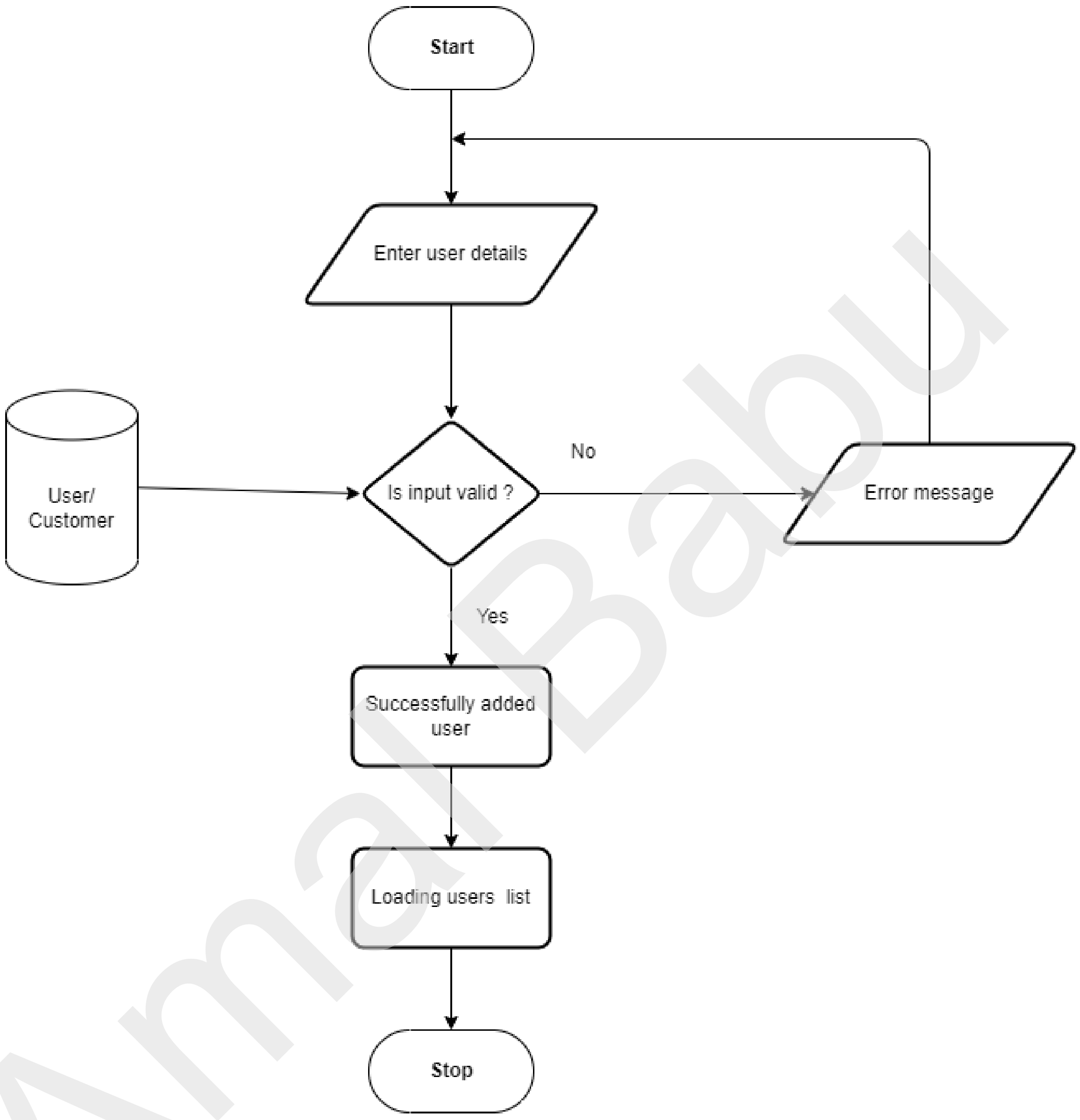


Figure 5.4 add a user by admin

c) **File input/output interface: user table**

d) **Output:** Entered user details (first_name,lastname, username and password) will be checked for validity if it is a valid user get added to user and customer table.

5.3.1.3. Admin view user/customer

a) Input : user_id, username

b) Procedural Details (Flow Chart):

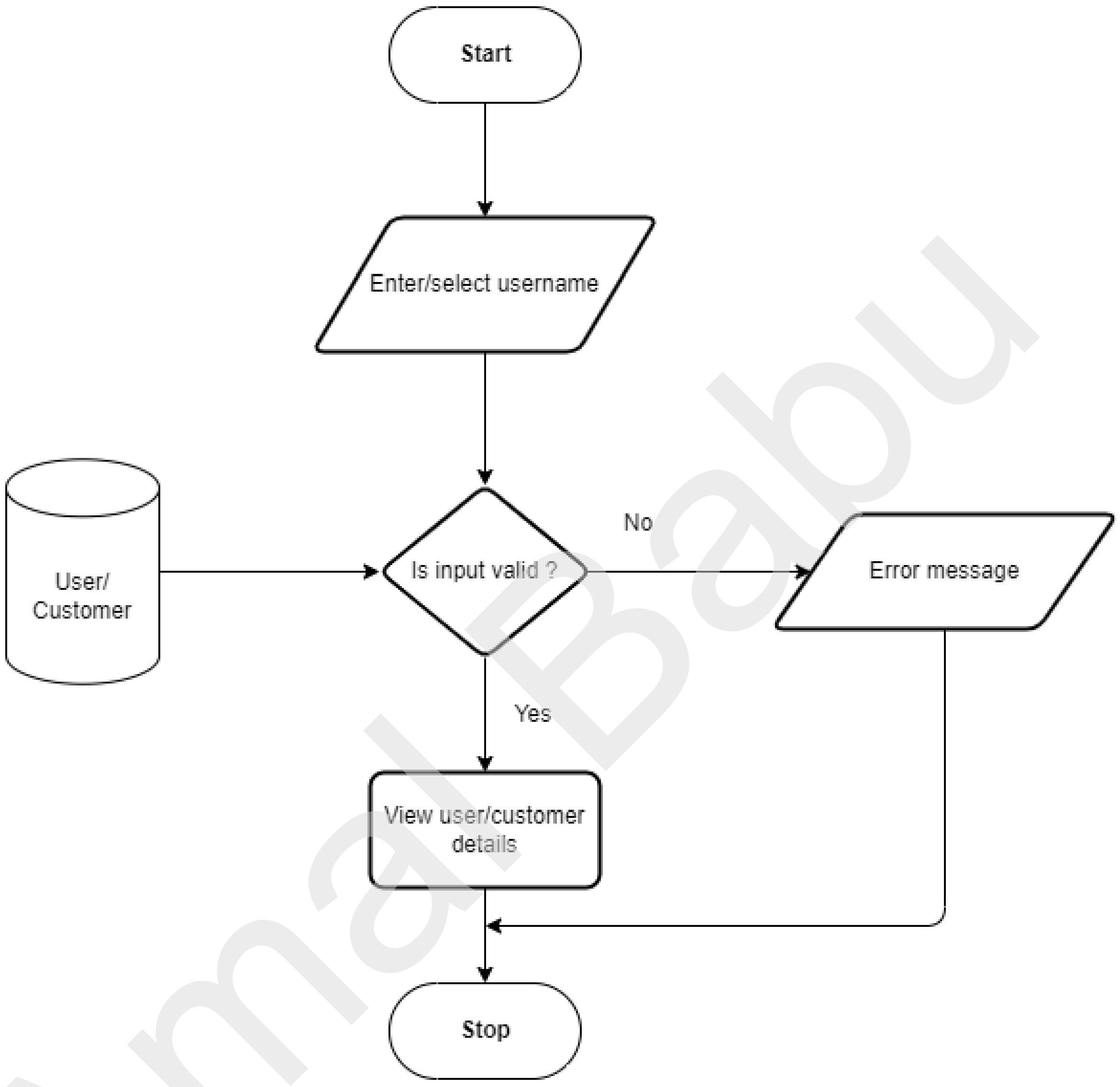


Figure 5.5 admin view user

c) File input/output interface: user table

d) **Output:** Select the user from users to list if it is valid user display user details.

5.3.1.4. Admin delete user

a) Input: user_id, username

b) Procedural Details (Flow Chart):

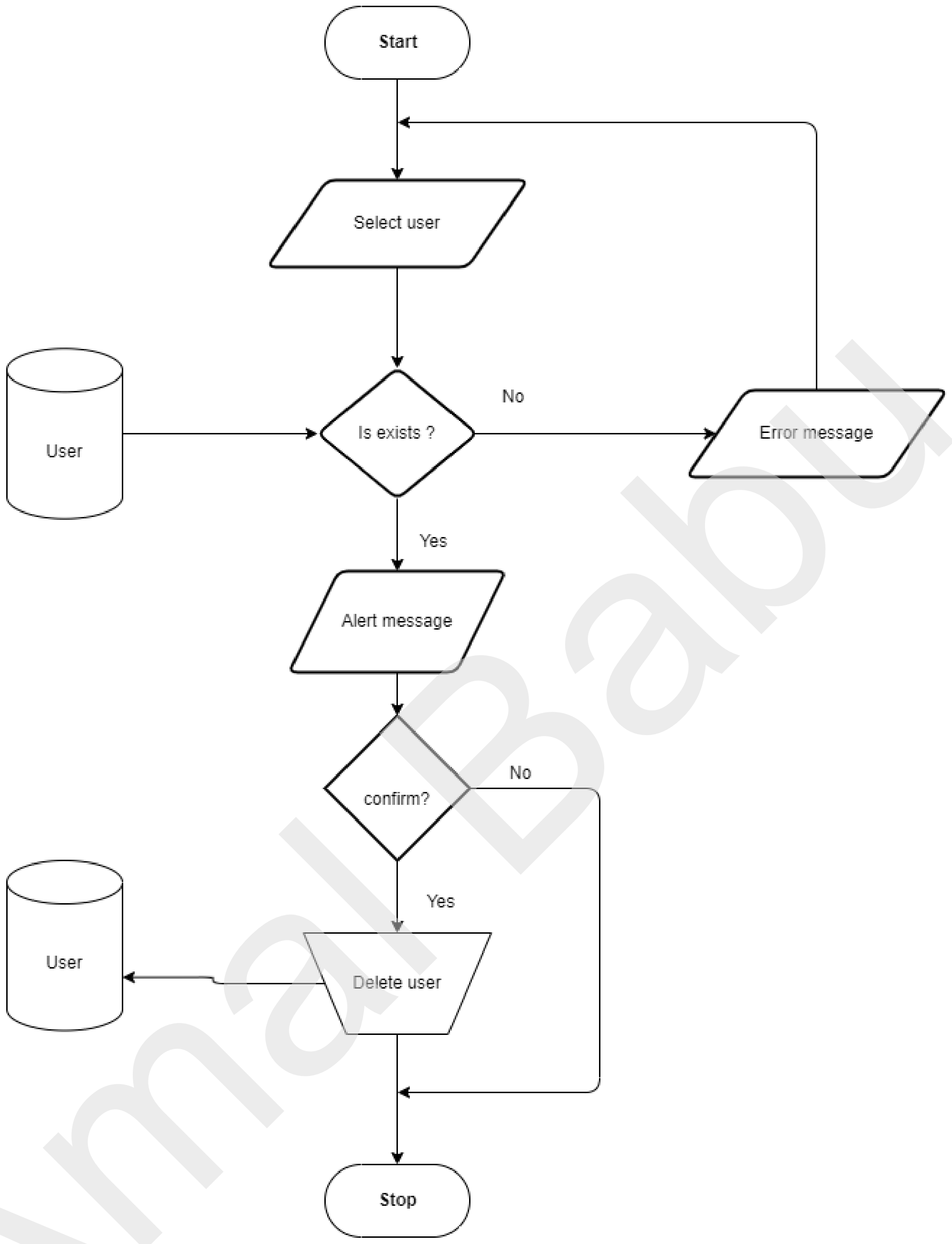


Figure 5.6 admin delete user

c) File input/output interface: user table

d) **Output:** user data get deleted.

5.3.1.5. **Admin add collection**

a) **Input: collection_title**

b) **Procedural Details (Flow Chart):**

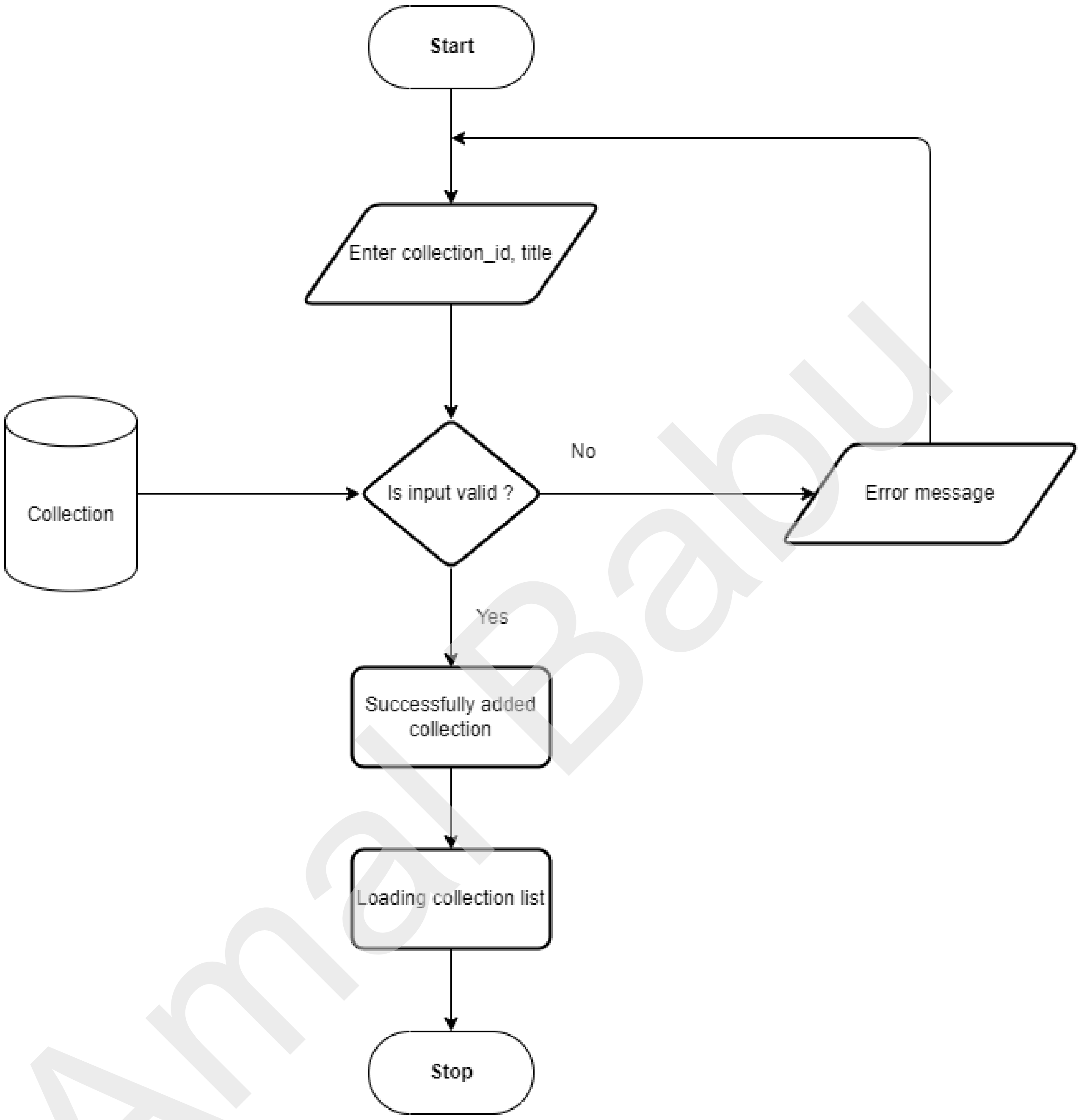


Figure 5.7 admin add collection

c) **File input/output interface:** Collection table

d) **Output:** collection added.

5.3.1.6. Admin view collection

a) Input: collection_id

b) Procedural Details (Flow Chart):

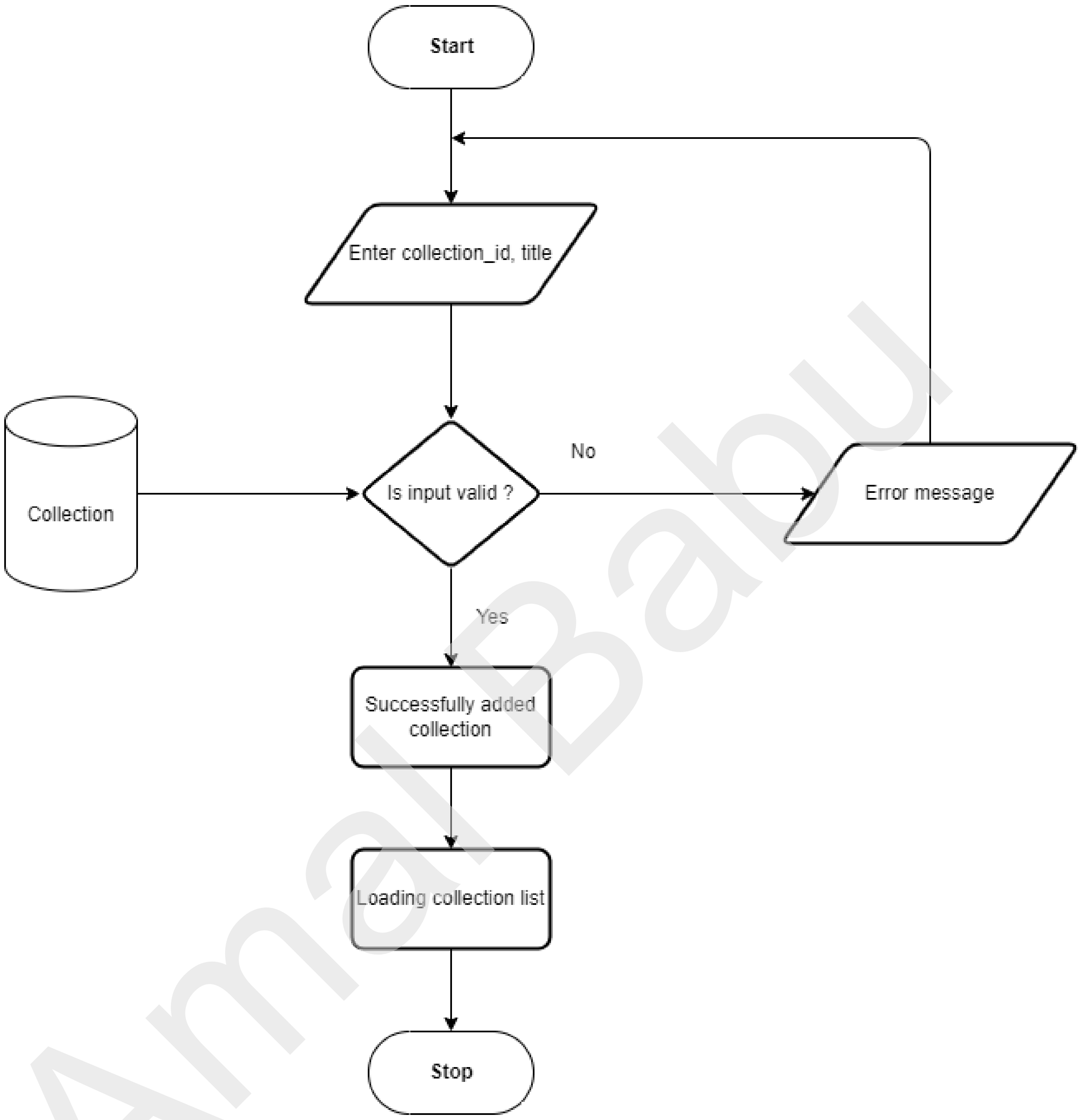


Figure 5.8 admin view collection

c) **File input/output interface:** Collection table

d) **Output:** collection details displayed.

5.3.1.7. Admin delete collection

a) Input: collection_id

b) Procedural Details (Flow Chart):

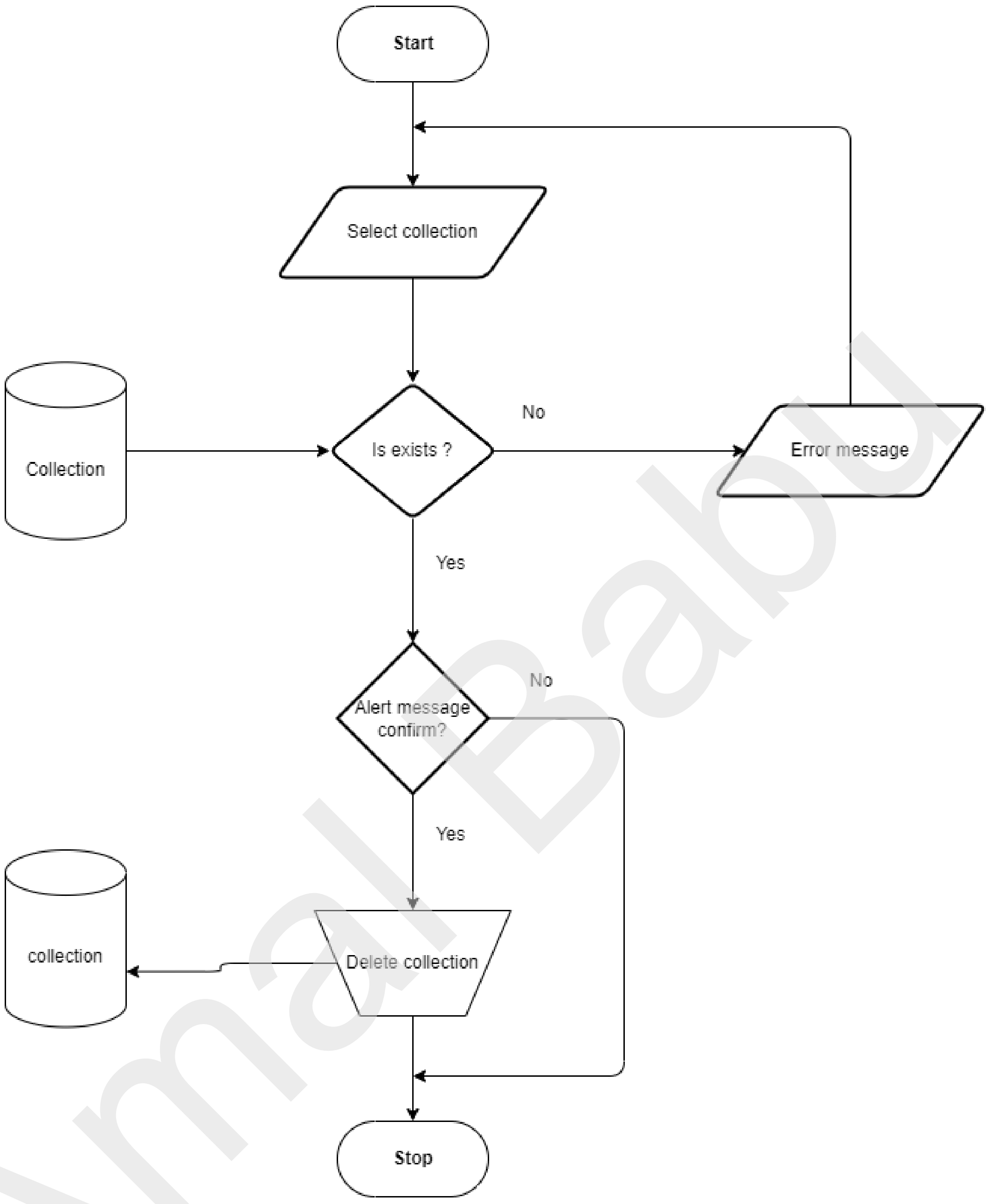


Figure 5.9 admin delete collection

c) **File input/output interface:** Collection table

d) **Output:** collection deleted.

5.3.1.8. Admin add product

a) Input: product_title,description, unit_price,collection_id

b) Procedural Details (Flow Chart):

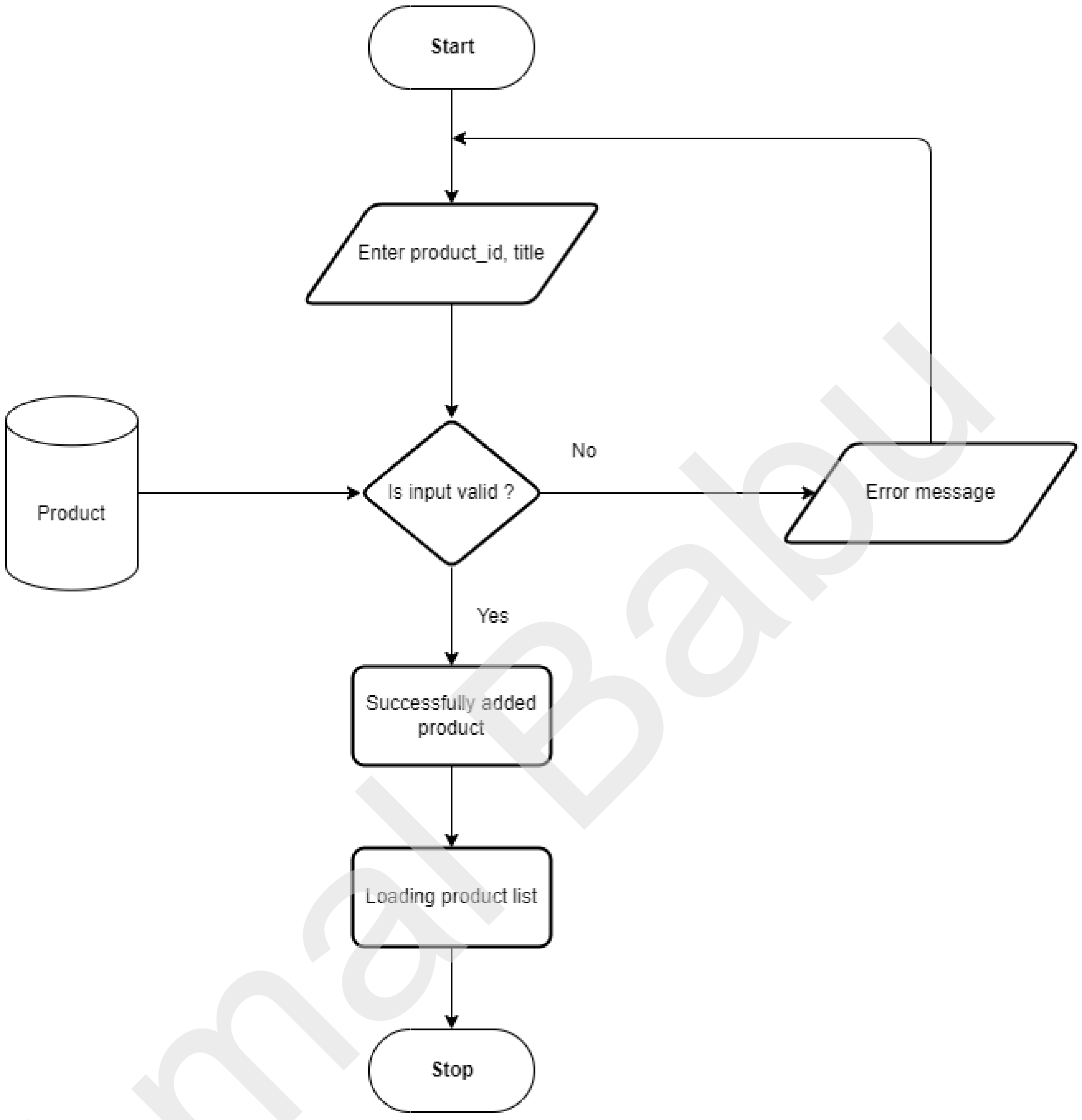


Figure 5.10 admin add product

c) **File input/output interface:** Product table

d) **Output:** product added.

5.3.1.9. Admin view product

a) Input: product_id

b) Procedural Details (Flow Chart):

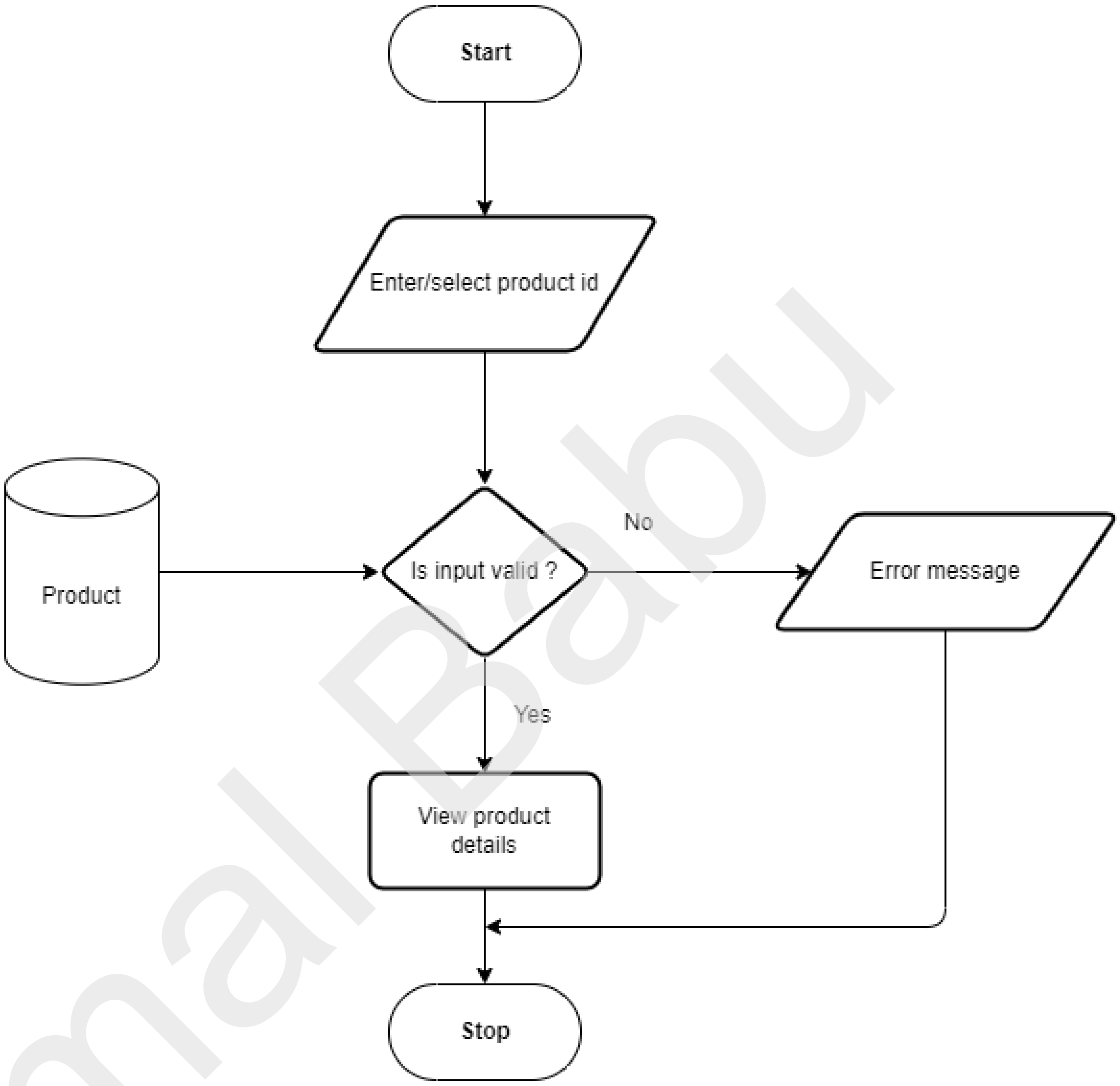


Figure 5.11 admin view product

c) **File input/output interface:** Product table

d) **Output:** product details displayed.

5.3.1.10. Admin delete product

a) **Input: product_id**

b) **Procedural Details (Flow Chart):**

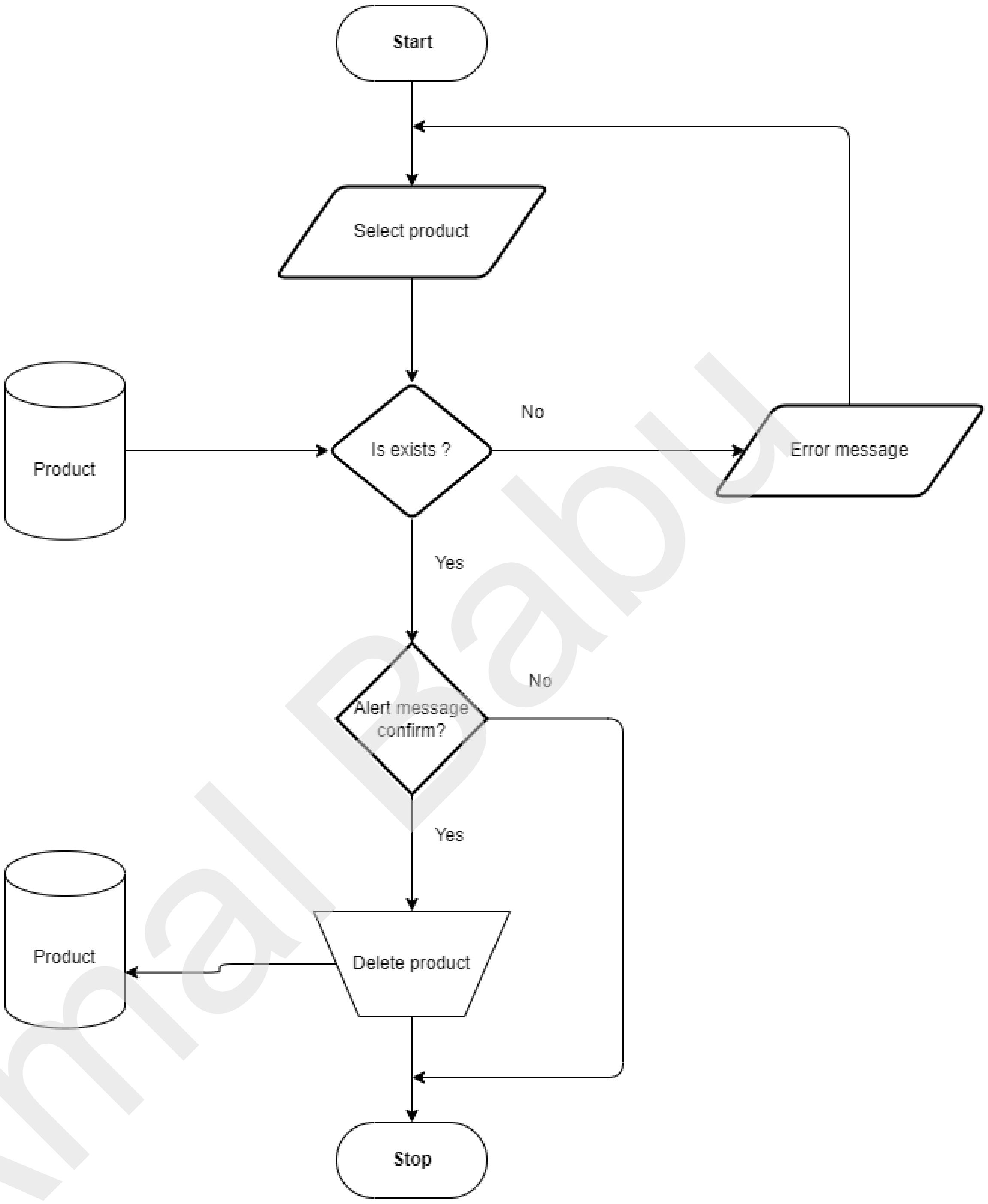


Figure 5.12 admin delete product

c) **File input/output interface:** Product table

d) **Output:** selected product deleted.

5.3.1.11. Admin view payment

a) **Input: payment_id**

b) **Procedural Details (Flow Chart):**

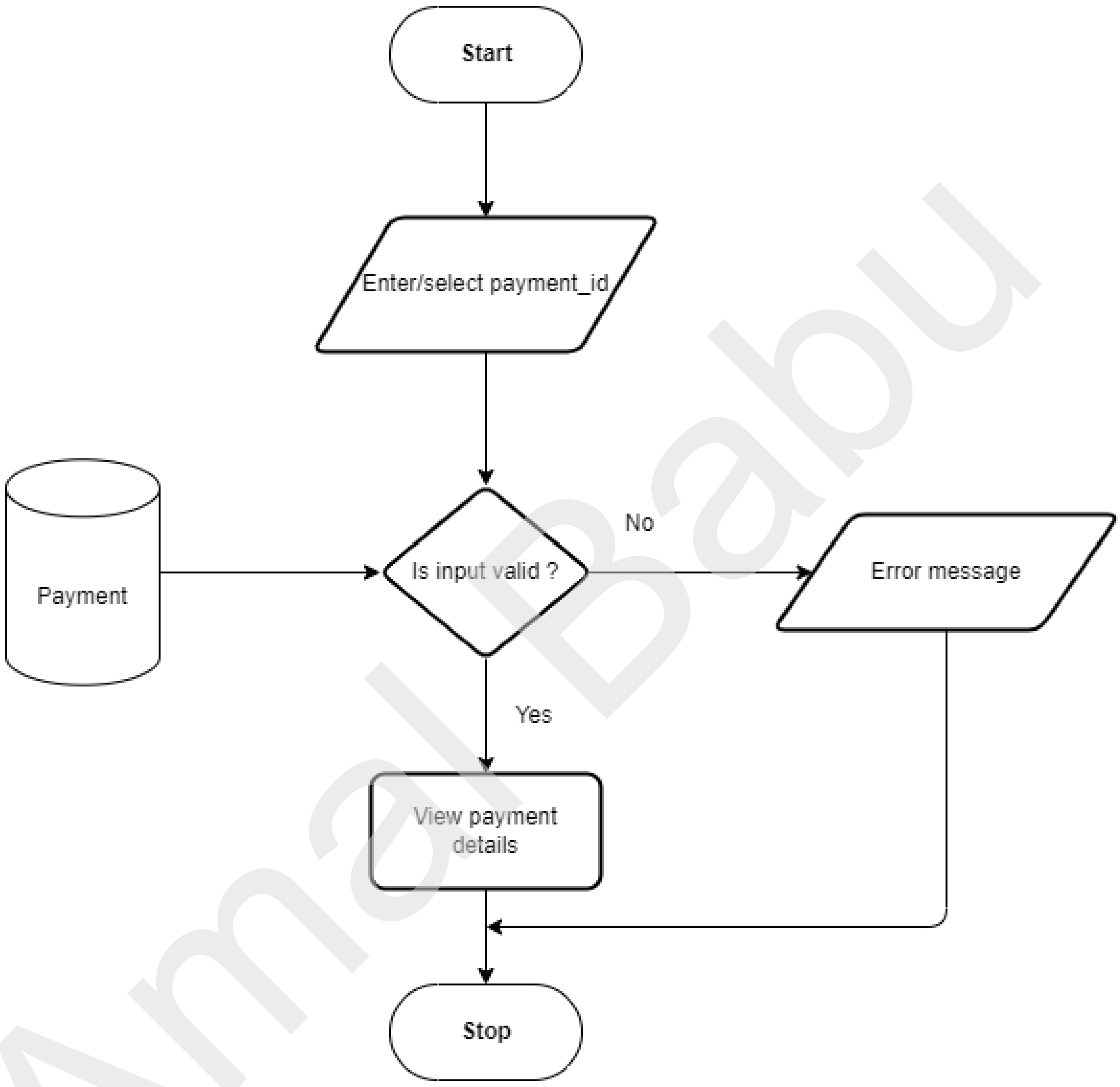


Figure 5.13 admin view payment

c) **File input/output interface:** payment table

d) **Output:** payment details displayed.

5.3.2. Customer

5.3.2.1. Customer login

a) **Input:** username,password

b) **Procedural Details (Flow Chart):**

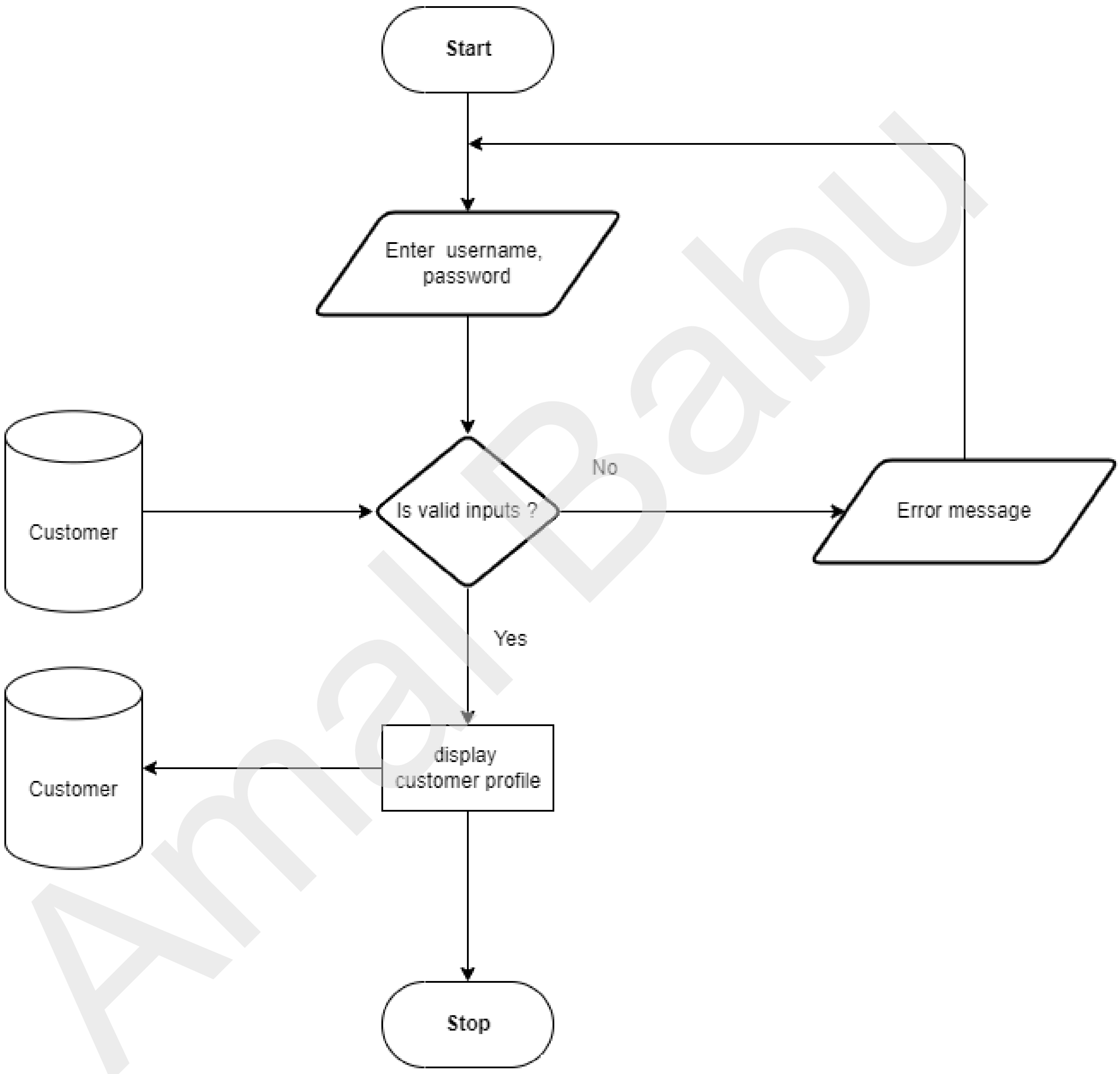


Figure 5.14 customer login

c) **File input/output interface:** Customer table

d) **Output:** customer profile displayed.

5.3.2.2. Customer view product

a) **Input:** product_id

b) **Procedural Details (Flow Chart):**

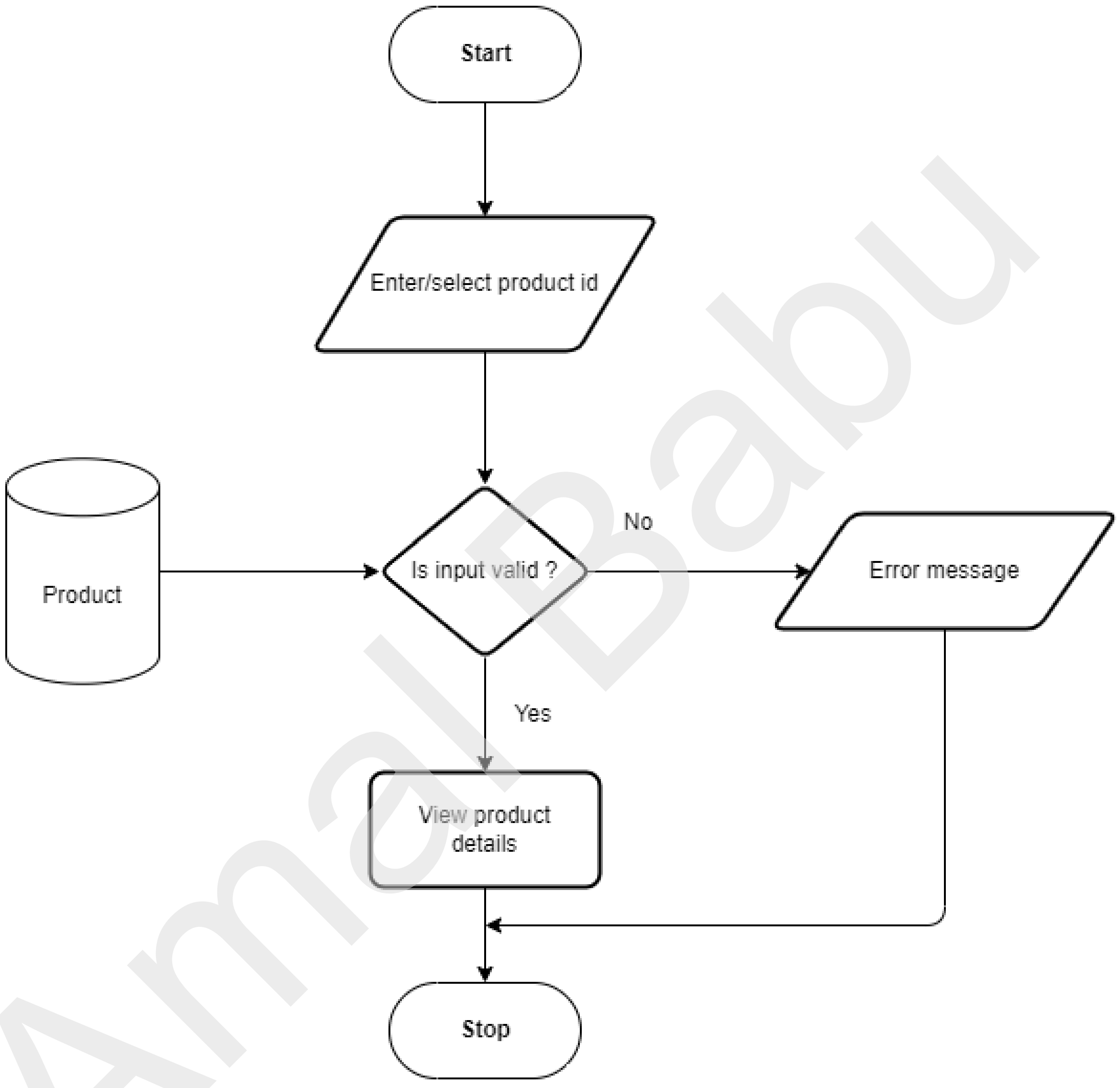


Figure 5.15 customer view product

c) **File input/output interface:** Customer table

d) **Output:** customer profile displayed.

5.3.2.3. Customer add product to cart

a) **Input:** product_id

b) **Procedural Details (Flow Chart):**

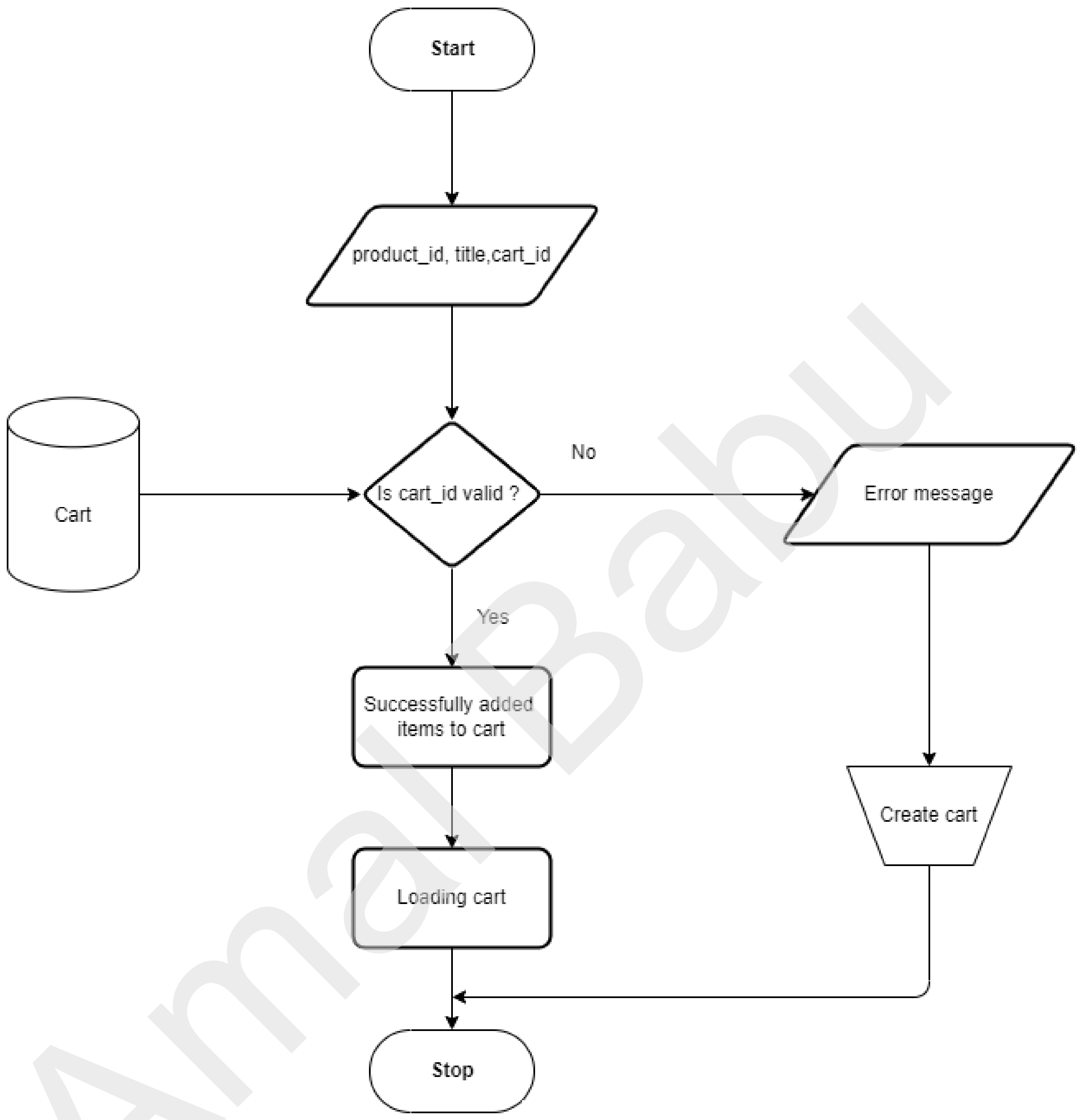


Figure 5.16 product add to cart

c) **File input/output interface:** Cart table

d) **Output:** cart items displayed

5.3.2.4. Customer make order

a) **Input:** order_id, payment_nonce, cart_id

b) **Procedural Details (Flow Chart):**

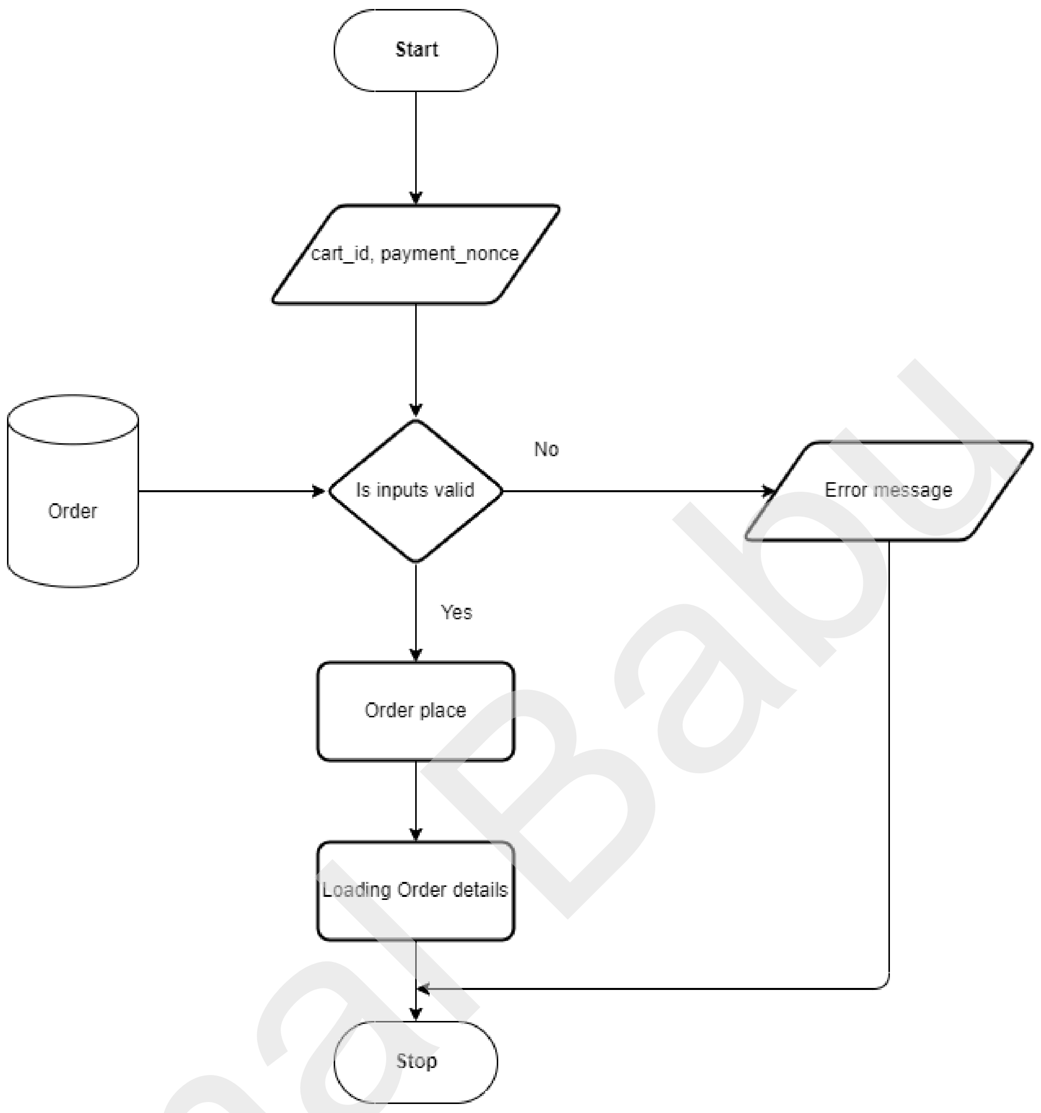


Figure 5.17 product make order

c) **File input/output interface:** Order table

d) **Output:** placed orders displayed

****

Chapter - 6

6. Program Code Listing

6.1. Database Connection

In settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'onshop-db-02',
        'USER': env('DATABASE_USER'),
        'PASSWORD': env('DATABASE_PASSWORD'),
        'HOST': 'localhost'

    },
}
```

6.2. Authorization/ Authentication

JWT(JSON Web Token) Authentication

This application using JWT authentication, This is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

Front end code snippet

```
export const signIn = createAsyncThunk('auth/signin', async ({ username,
password }) => {
    const response = await axios.post(`${API}core/auth/token/`,
        { username: username, password: password },
        { headers: { 'Content-Type': 'application/json' } })

    console.log(response.data)
    return response.data
})
```

Backend code

Django-rest-framework-simple jwt python library is used to implement JWT auth backend configuration.

settings.py

```
SIMPLE_JWT = {
    'AUTH_HEADER_TYPES': ('JWT',),
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=30),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
}
```

urls.py

```
urlpatterns = [
    path('', TemplateView.as_view(template_name='core/index.html')),
    path('auth/token/', MyTokenObtainPairView.as_view(),
name='token_obtain_pair'),
    path('auth/token/refresh/',TokenRefreshView.as_view(),
name='token_refresh'),
]
```

serializers.py

```
class UserCreateSerializer(BaseUserCreateSerializer):

    class Meta(BaseUserCreateSerializer.Meta):

        fields = ['id', 'username', 'email', 'password',
                  'email', 'first_name', 'last_name']



class UserSerializer(BaseUserSerializer):
    class Meta(BaseUserSerializer.Meta):
        model = User
        fields = ['id', 'username', 'email', 'first_name', 'last_name', ]
```

views.py

```
class MyTokenObtainPairSerializer(TokenObtainPairSerializer):
    @classmethod
    def get_token(cls, user):
        token = super().get_token(user)

        # Add custom claims
        token['username'] = user.username

        return token


class MyTokenObtainPairView(TokenObtainPairView):
    serializer_class = MyTokenObtainPairSerializer
```

User registration

Fontend code

```
REGISTER_NEW_USER_API=`${API}auth/users/`

const registerUserHandler = async (formValues) => {

    const response = await axios.post(REGISTER_NEW_USER_API, {

      username: formValues.username,
      email: formValues.email,
      password: formValues.password,
      first_name: formValues.firstname,
      last_name: formValues.lastname,

    }, {
      headers: {
        'content-Type': 'application/json'
      }
    })

      .then((response) => {

        toast.success('Signup success')
        console.log(response.data)

        navigate('/signin')
      })
      .catch((err) => {

        console.log(err)


        let serverErros = ''



        if (err?.response?.data?.password) {
          err.response.data.password.map((res) => (serverErros += '\n' +
res))

        } else if (err?.response?.data?.username) {
          err.response.data.username.map((res) => (serverErros += '\n' +
res))

        } else if (err?.response?.data?.email) {

          err.response.data.email.map((res) => (serverErros += '\n' + res))

        } else {

          serverErros += 'Something went wrong please try again!'

        }


        toast.error(serverErros, { autoClose: 10000 })
```

```
                console.log(err)

            })

        }
```

6.3. Data store/ retrieval/ update

6.3.1. Product

```
class ProductViewSet(ModelViewSet):
    """
    A viewset that provides default `create()`, `retrieve()`,
`update()`,
    `partial_update()`, `destroy()` and `list()` actions.
    queryset = Product.objects.prefetch_related('images').all()
    serializer_class = ProductSerializer
    permission_classes = [IsAdminOrReadOnly]

    # Using django filter library for filtering product based on the
collection
    # define filterbackend and filteing logic in a class
    # e.g: url-->
http://127.0.0.1:8000/store/products/?collection_id=4 , filtering
query is-->products/?collection_id=4
    filter_backends = [DjangoFilterBackend, SearchFilter,
OrderingFilter]
    filterset_class = ProductFilter
    search_fields = ['title', 'description']
    # sorting based on unit_price and last_update
    ordering_fields = ['unit_price', 'last_update']
    pagination_class = DefaultPagination

    # overrid

    def get_serializer_context(self):
        return {'request': self.request}

    # overrid destroy to delete
    def destroy(self, request, *args, **kwargs):
        if Product.objects.filter(product_id=kwargs['pk']).count() >
0:
            return Response({'error': "Can't delete , product
associated with an order"},
status=status.HTTP_405_METHOD_NOT_ALLOWED)
        return super().destroy(request, *args, **kwargs)
```

6.3.2. Collection

```
class CollectionViewSet(ModelViewSet):

    serializer_class = CollectionSerializer
    queryset = Collection.objects.annotate(
        product_count=Count('products')).all().order_by('title')
    permission_classes = [IsAdminOrReadOnly]

    # Override
    def destroy(self, request, *args, **kwargs):
        collection = get_object_or_404(Collection.objects.annotate(
            product_count=Count('products')), pk=kwargs['pk'])

        if collection.products.count() > 0:
            return Response({'error': "Can't delete , it includes
one or more products"},

status=status.HTTP_405_METHOD_NOT_ALLOWED)
        return super().destroy(request, *args, **kwargs)
```

6.3.3. Review

```
class ReviewViewSet(ModelViewSet):

    # permission_classes=[IsAuthenticated]
    http_method_names = ['get', 'post',
                         'options', 'headers']  # +['put', 'delete',
]

    def get_permissions(self):
        if self.request.method in ['POST', 'PUT', 'PATCH']:
            return [IsAuthenticated()]
        return super().get_permissions()

    serializer_class = ReviewSerializer

    def get_queryset(self):
        # product_pk is from url
        return
Review.objects.filter(product_id=self.kwargs['product_pk']).order_by
('-id')

    # form this view class we can access url parameters;
    # and send it to serializer by using get_serializer_context()
    # Overriding ; send to serializer ; for getting product_id
    def get_serializer_context(self):
        return {'product_id': self.kwargs['product_pk']}
```

6.3.4. Cart

```
class CartViewSet(CreateModelMixin,  # create cart with id, pass
post request with empty ,
                  RetrieveModelMixin,  # ../carts/id/ retriving a
spesific cart
                  DestroyModelMixin,  # delete a 'cart/id/'
                  GenericViewSet
                  ):

    # prefetch_related used for fetch child table items, in
foreignkey realation we use select_related
    queryset = Cart.objects.prefetch_related('items__product').all()
    serializer_class = CartSerializer
```

6.3.5. Cart Item

```
class CartItemViewSet(ModelViewSet):

    # must be lowercase in the list
    http_method_names = ['get', 'post', 'patch', 'delete']

    def get_serializer_class(self):
        if self.request.method == 'POST':
            return AddCartItemSerilizer
        elif self.request.method == 'PATCH':
            return UpdateCartItemSerilizer
        return CartItemSerializer
    def get_queryset(self):

        return CartItem.objects \
            .filter(cart_id=self.kwargs['cart_pk']) \
            .select_related('product')

    # cart_pk value from url; add to context dict ; so we can access
this value in serializer for creating custom save methode(override
save methode)
    def get_serializer_context(self):
        return {'cart_id': self.kwargs['cart_pk']}
```

6.3.6. Address

```
class AddressViewSet(ModelViewSet):

    http_method_names = ['get', 'put', 'post', 'delete']
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        return
Address.objects.filter(customer_id=self.kwargs['customer_pk'])

    def get_serializer_context(self):
        return {'customer_id': self.kwargs['customer_pk']}

    def get_serializer_class(self):
        if self.request.method == 'POST':
            return AddAddressSerializer
        elif self.request.method == 'PUT':
            return UpdateAddressSerializer
        return AddressSerializer
```

6.3.7. Customer

```
queryset = Customer.objects.prefetch_related('address').all()
    serializer_class = CustomerSerializer
    # Allow all operation to admin user. TODO:
FullDjangoModelPermission (our customized permission class ) can
also use here
    permission_classes = [IsAdminUser]
    # Define a  custom action get customer profile
    # here detail=Flase , so it is avalilable in the list view
    # list-view means store/customers/me , detail-view means
store/customers/id/me
    # this method only available for autheticated user, overrided
the permission class to Is Autheticated

    @action(detail=False, methods=['GET', 'PUT'],
permission_classes=[IsAuthenticated])
    def me(self, request):
        customer = Customer.objects.get(
            user_id=request.user.id)
        if request.method == 'GET':
            serilalizer = CustomerSerializer(customer)
            return Response(serilalizer.data)
        elif request.method == 'PUT':
            serializer = CustomerSerializer(customer,
data=request.data)
            serializer.is_valid(raise_exception=True)
            serializer.save()
            return Response(serializer.data)

    # implimetaion of custom permission for view history . TODO:
    @action(detail=True,
permission_classes=[ViewCustomerHistoryPermission])
    def history(self, request, pk):
        return Response('Ok')
```

6.3.8. Order

```
class OrderViewSet(ModelViewSet):
    http_method_names = ['get', 'post', 'patch',
                         'delete', 'head', 'options']

    # def get_permissions(self):
    #     if self.request.method in ['PATCH', 'DELETE']:
    #         return [IsAdminUser()]
    #     return [IsAuthenticated()]
    permission_classes=[IsAuthenticated]

    def get_serializer_class(self):
        if self.request.method == 'POST':
            return CreateOrderSerializer
        elif self.request.method == 'PATCH':
            return UpdateOrderSerializer
        return OrderSerializer

    def get_queryset(self):
        user = self.request.user
        # admin or staff are able to see all orders
        if user.is_staff:
            return
Order.objects.prefetch_related('items').all().order_by('-id')

        customer_id = Customer.objects \
            .only('id').get(user_id=user.id)

        return
Order.objects.filter(customer_id=customer_id).order_by('-id')

    def create(self, request, *args, **kwargs):
        serializer = CreateOrderSerializer(
            data=request.data,
            context={'user_id': self.request.user.id}
        )
        serializer.is_valid(raise_exception=True)
        order = serializer.save()
        # deserialize the saved order using order serializer ;
CreateOrderSerializer only for creating and returning with cart_id
        serializer = OrderSerializer(order)
        return Response(serializer.data)
```

6.3.9. Product Image

```
class ProductImageViewSet(ModelViewSet):

    serializer_class = ProductImageSerializer

    def get_queryset(self):
        return
ProductImage.objects.filter(product_id=self.kwargs['product_pk'])

    def get_serializer_context(self):
        return {'product_id': self.kwargs['product_pk']}
```

6.4. Data validation.

6.4.1. Backend data validators

settings.py

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityVali
dator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

6.4.2. Frontend

6.4.2.1. signup

```
const initialValues = {
    firstname: '',
    lastname: '',
    email: '',
    username: '',
    password: '',
    confPassword: ''
  }


  const [formValues, setFormValues] = useState(initialValues)
  const [formErrors, setFormErrors] = useState({})
  const [isSubmit, setIsSubmit] = useState(false)
  const [passwordType, setPasswordType] = useState('password')



  const onChangeInputFieldsHandler = (e) => {


    const { name, value } = e.target

    setFormValues({ ...formValues, [name]: value })


  }

  const onSubmitHandler = (e) => {
    e.preventDefault();

    setFormErrors(validate(formValues))
    setIsSubmit(true)

  }

  const validate = (values) => {

    const errors = {};

    if (!EMAIL_REGEXP.test(values.email)) {
      errors.email = "This is not valid email"
    }
    if (values.password.length < 9) {
      errors.password = "Password must be more than 8 character"

    }
    if (!(values.password === values.confPassword)) {
      errors.confPassword = "Password doesn's match"
    }

    return errors
  }
```

```
useEffect(() => {

  console.log(formErrors)
  if (Object.keys(formErrors).length === 0 && isSubmit) {
    // console.log('fv', formValues)

    registerUserHandler(formValues)

  }

}, [formErrors])
```

6.4.2.2. signIn

```
const onClickSignIn = (e) => {
  e.preventDefault();

  if (username === "") {
    setErr("Enter username");

    return false;
  }
  if (parseInt(password.length) < 4) {
    setErr("Enter correct password");

    return false;
  }

  setErr("");

  console.log("username");

  dispatch(signIn({ username, password }));
  return true;
};
```

6.4.2.3. Update address

```
const dispatch = useDispatch();
  const customerInfo = useSelector(selectCustomerInfo);

  const initialValues = {
    street: customerInfo?.address[0]?.street ?? "",
    city: customerInfo?.address[0]?.city ?? "",
    landmark: customerInfo?.address[0]?.land_mark ?? "",
    house: customerInfo?.address[0]?.house_no ?? "",
    postal: customerInfo?.address[0]?.postal ?? "",
    phone: customerInfo?.address[0]?.phone_no ?? "",
  };

  const [formValues, setFormValues] = useState(initialValues);
  const [formErrors, setFormErrors] = useState({});
  const [isSubmit, setIsSubmit] = useState(false);

  const onChangeInputFieldHandler = (e) => {
    const { name, value } = e.target;
    setFormValues({ ...formValues, [name]: value });
  };

  const onSubmitHandler = (e) => {
    e.preventDefault();

    setFormErrors(validate(formValues));
    setIsSubmit(true);
  };

  const validate = (values) => {
    const errors = {};

    if (
      customerInfo?.address[0]?.street === values.street &&
      customerInfo?.address[0]?.city === values.city &&
      customerInfo?.address[0]?.land_mark === values.landmark &&
      customerInfo?.address[0]?.phone_no === values.phone &&
      customerInfo?.address[0]?.postal === parseInt(values.postal) &&
      customerInfo?.address[0]?.house_no === parseInt(values.house)
    ) {
      toast.warn("No changes found", {
        autoClose: 1000,
        hideProgressBar: true,
      });
      errors.street = "No change found";
      errors.city = "No change found";
      errors.phone = "No change found";
      errors.house = "No change found";
      errors.landmark = "No change found";
      errors.postal = "No change found";
    }
    if (!values.phone.match(INDIAN_PHONE_REGEXP)) {
      errors.phone = "Please enter valid phone number";
    }

    // TODO: can add more validation condition here

    return errors;
  };

  const updateAddressHandler = async (
    customerID,
    formValues,
    addressId = 1
  ) => {
    await axiosInstance
      .put(`${STORE_API}customers/${customerID}/address/${addressId}/`, {
        street: formValues.street,
        city: formValues.city,
        postal: formValues.postal,
        house_no: formValues.house,
        land_mark: formValues.landmark,
        phone_no: formValues.phone,
      })
      .then((response) => {
        console.log(response.data);
        toast.success("Updated", { autoClose: 1000, hideProgressBar: true
});
        dispatch(fetchCustomerInfo());
      })
      .catch((err) => {
        toast.error("Something went wrong!", { hideProgressBar: true });
        console.log(err);
      });
  };

  const addAddressHandler = async (customerID, formValues, addressId = 1)
=> {
    await axiosInstance
      .post(`${STORE_API}customers/${customerID}/address/`, {
        street: formValues.street,
        city: formValues.city,
        postal: formValues.postal,
        house_no: formValues.house,
        land_mark: formValues.landmark,
        phone_no: formValues.phone,
      })
      .then((response) => {
        console.log(response.data);
        toast.success("Updated", { autoClose: 1000, hideProgressBar: true
});
        dispatch(fetchCustomerInfo());
      })
      .catch((err) => {
        toast.error("Something went wrong!", { hideProgressBar: true });
        console.log(err);
      });
  };

  useEffect(() => {
    if (Object.keys(formErrors).length === 0 && isSubmit) {
      console.log(formValues);

      if (customerInfo.address[0]) {
        updateAddressHandler(
          customerInfo.id,
          formValues,
          customerInfo.address[0]?.id
        );
      } else {
        addAddressHandler(customerInfo.id, formValues);
      }
    }
  }, [formErrors]);
```

6.4.2.4. Add/update user information

```
const onSubmitUserInfo = (e) => {
    e.preventDefault();

    if (
      firstname === customerInfo?.first_name &&
      lastname === customerInfo?.last_name &&
      email === customerInfo?.email
    ) {
      toast.warn("No change found...", { hideProgressBar: true });
      return false;
    } else if (!(firstname && lastname && email)) {
      toast.warn("All fields are required", { hideProgressBar: true });
      return false;
    }
```

```
        dispatch(updateUserInfo({ firstname, lastname, email }));
        return true;
      };
```

6.4.2.5. update customer info

```
    const onSubmitCustomerInfo = (e) => {
        e.preventDefault();

        if (!phone.match(INDIAN_PHONE_REGEXP)) {
          toast.error("Enter valid phone number", { autoClose: 2000 });
          return false;
        } else if (new Date().getFullYear() - new Date(dob).getFullYear() < 15)
{
          toast.error("Date of birth is not valid, you should be above 15");
          return false;
        } else if (
          phone === customerInfo.phone &&
          dob === customerInfo.birth_date &&
          membership === customerInfo.membership
        ) {
          toast.warn("No change found", { autoClose: 2000, hideProgressBar:
true });
          return false;
        }

        dispatch(updateCustomerInfo({ phone, dob, membership }));
        return true;
      };
```

6.5. Search

6.5.1. Search for products

Frontend

Product search API (localhost) =

`` `http://127.0.0.1:8000/store/products/?page=1&search=${searchQuery}` ``

```
 With collection id=
```

`` `http://127.0.0.1:8000/store/products/?collection_id=${currentCollectionId}&ordering=unit_price&page=1&unit_price__gt=&unit_price__lt=` ``

```
const searchButtonClickHandler = (e) => {
    e.preventDefault()
    navigate('/products')

    if (!searchQuery == '') {
      dispatch(fetchProducts({ page:
`http://127.0.0.1:8000/store/products/?page=1&search=${searchQuery}`
}))
      dispatch(setPaginationNumber(1))
    } else {

      toast.error("Enter search query",{ autoClose: 1000 })

    }

  }
```

Backend :

```
class ProductViewSet(ModelViewSet):
    """

    A viewset that provides default `create()`,
`retrieve()`, `update()`,
    `partial_update()`, `destroy()` and `list()` actions.
    """
    queryset =
Product.objects.prefetch_related('images').all()
    serializer_class = ProductSerializer
    permission_classes = [IsAdminOrReadOnly]

    filter_backends = [DjangoFilterBackend, SearchFilter,
OrderingFilter]
    filterset_class = ProductFilter
    search_fields = ['title', 'description']
    # sorting based on unit_price and last_update
    ordering_fields = ['unit_price', 'last_update']
    pagination_class = DefaultPagination

    # overrid

    def get_serializer_context(self):
        return {'request': self.request}

    # overrid destroy to delete
    def destroy(self, request, *args, **kwargs):
        if
Product.objects.filter(product_id=kwargs['pk']).count() > 0:
            return Response({'error': "Can't delete ,
product associated with an order"},

status=status.HTTP_405_METHOD_NOT_ALLOWED)
        return super().destroy(request, *args, **kwargs)
```

6.6. Named procedures/functions

Braintree payment

Frontend

```
export const getPaymentToken = async () => {

    return (
        await axiosInstance.get('payment/braintree/gettoken/')
            .then((response) => {
                console.log(response.data)
                return response.data
            })
            .catch((error) => {
                console.log(error)
                toast.error('token generation failed', {
hideProgressBar: true })
            })
    )

}

export const processPayment = async ({ paymentData }) => {

    return (
        await
axiosInstance.post('payment/braintree/process_payment/', {
paymentData })
```

```
                .then((response) => {
                    console.log(response.data)
                    return response.data
                })
                .catch((error) => {
                    console.log('paymentData',paymentData    )
                    console.log(error)
                    toast.error('Payment failed', { hideProgressBar:
true })

                })
        )
}



const onPayment = () => {
            setInfo({ ...info, loading: true })
            let nonce;
            console.log(info.instance)
            let getNonce = info.instance.requestPaymentMethod()
                .then((data) => {
                    nonce = data.nonce
                    console.log('nonce', nonce)
                    const paymentData = {
                        paymentMethodNonce: nonce,
                        amount: totalAmount
                    };
                    processPayment({ paymentData })
                        .then(((response) => {
                            if (response.error) {
                                if (response.code == '1') {
                                    toast.error('Payment failed', {
hideProgressBar: true })
                                    setInfo({ ...info, loading: false })
                                    //payment failed
                                }
                            } else {
                                //no error all good!
                                setInfo({
                                    ...info,
                                    success: response.success,
                                    loading: false
                                })
```

```
                                        toast.success('Payment success', {
hideProgressBar: true })
                                        toast.info('You can place your order', {
hideProgressBar: true })
                                        console.log('paymentSucess')
                                        const paymentResponseData = {
                                            transactionId:
response.transaction.id,
                                            amount: response.transaction.amount,
                                            paymentMethod:'ON',
                                            paymentStatus:'C',

                                        }

dispatch(setPaymentDetails(paymentResponseData))
                                        console.log(paymentResponseData)

navigate('/user/place-order/',{replace:true})
                                    }
                                }))
                        })
                        .catch((err) => {
                            console.log('err')
                            console.log('nonceErr', err)
                            toast.error('Payment failed', { hideProgressBar:
true })
                            setInfo({ ...info, loading: false })

                        })
    }
```

Backend

```
class PaymentBApiViewSet(ViewSet):

    permission_classes = [IsAuthenticated]

    # Braintree

    @csrf_exempt
    @action(detail=False, methods=['GET'])
    def gettoken(self, request):
        # pass client_token to your front-end
        user_id = request.user.id
        queryset = Customer.objects.get(user_id=user_id)
        customer = CustomerSerializer(queryset)
        client_token = gateway.client_token.generate()
        return Response({'client_token': client_token, 'success':
True, 'customer': customer.data})

    @csrf_exempt
    @action(detail=False, methods=['POST'])
    def process_payment(self, request):

        print('dattta', request.data)

        data = request.data['paymentData']
        nonce_from_the_client = data['paymentMethodNonce']
        amount = data['amount']

        # print('dattta',nonce_from_the_client)

        # return Response(request.data)

        result = gateway.transaction.sale({
            "amount": amount,
            "payment_method_nonce": nonce_from_the_client,
            "options": {
                "submit_for_settlement": True
            }
        })

        if (result.is_success):
            return Response({'success': result.is_success,
                             'transaction': {
                                 'id': result.transaction.id,
                                 'amount': result.transaction.amount
                             }
                             })
        else:
            return Response({'error': True, 'success': False})
```

Razorpay payment

Frontend

```
const loadScript = async () => {
    const script = document.createElement("script");
    script.src = "https://checkout.razorpay.com/v1/checkout.js";
    document.body.appendChild(script);
```

```
  };

  const showRazorpay = async () => {
    const res = await loadScript();

    let bodyData = new FormData();

    // we will pass the totalAmount and product name to the backend
using form data
    bodyData.append("amount", totalAmount.toString());

    const data = await axiosInstance({
      url: `${API}payment/razorpay/start_payment/`,
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
      data: bodyData,
    }).then((res) => {
      console.log('response', res)
      return res;
    }).catch((err) => {

      console.log(err)
      toast.warn('Something went wrong!')

    });
    var options = {
      key_id: process.env.REACT_APP_PUBLIC_KEY, // in react your
environment variable must start with REACT_APP_
      key_secret: process.env.REACT_APP_SECRET_KEY,
      totalAmount: data.data.payment.totalAmount,
      currency: "INR",
      name: "OnShop",
      description: "Online payment",
      image: "", // add image url
      order_id: data.data.payment.id,
      handler: (response) => {
        // we will handle success by calling handlePaymentSuccess
method and
        // will pass the response that we've got from razorpay
        // handlePaymentSuccess(response);
        console.log(response)
```

```
          if (response.status_code === 200 ||
response.razorpay_order_id !== null) {
            toast.success("Payment success", { hideProgressBar: true
});
            toast.info("You can place your order", { hideProgressBar:
true });
            console.log("paymentSucess");
            const paymentResponseData = {
              transactionId:
                "pay_id=" +
                response.razorpay_payment_id +
                ",orderid=" +
                response.razorpay_order_id,
              amount: totalAmount,
              paymentMethod: "ON",
              paymentStatus: "C",
            };
            dispatch(setPaymentDetails(paymentResponseData));
            console.log(paymentResponseData);
            navigate("/user/place-order/", { replace: true });
          } else {
            toast.error("Payment failed", { hideProgressBar: true });
          }
        },
        prefill: {
          name: username,
          email: customerEmail,
          contact: customerphone,
        },
        notes: {
          address: "Razorpay Corporate Office",
        },
        theme: {
          color: "#1266F1",
        },
      };

      var rzp1 = new window.Razorpay(options);
      rzp1.open();
    };
```

Backend

```
class PaymentRApiViewSet(ViewSet):

    permission_classes = [IsAuthenticated]

    @csrf_exempt
    @action(detail=False, methods=['POST'])
    def start_payment(self, request):
        # pass client_token to your front-end
        user_id = request.user.id
        username = request.user.username
        queryset = Customer.objects.get(user_id=user_id)
        customer = CustomerSerializer(queryset)

        amount = request.data['amount']
        print(amount)

        DATA = {"amount": int(amount) * 100,
                "currency": "INR",
                "payment_capture": "1"}

        client = razorpay.Client(
            auth=('rzp_test_jj05mMUix1fD9r',
'aOo31hAP4J62uMGBisX8IKkg'))

        payment = client.order.create(DATA)

        return Response({'success': True, 'customer': customer.data,
'username': username, 'payment': payment})
```

****

Chapter - 7

7. User Interfaces

7.1. Login page

7.1.1. Admin login

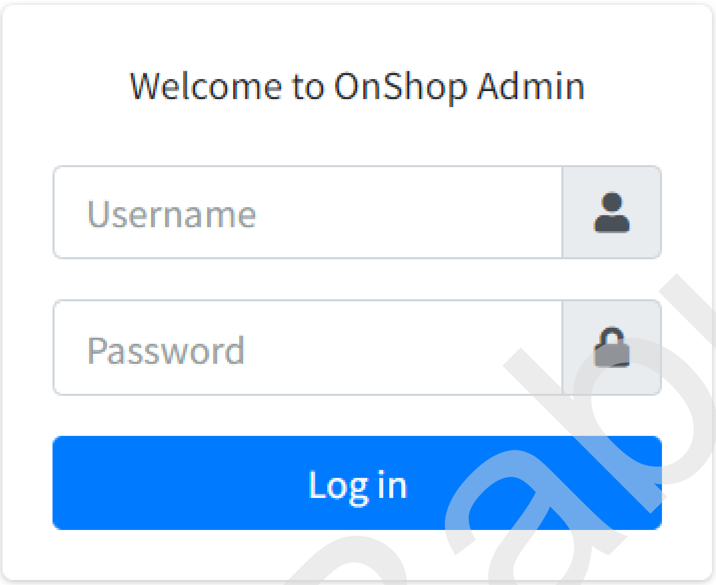


Figure 7.1 admin login

Purpose

This is an admin login page. Admin can login to the dashboard by providing username and password.

Navigation

Admin can click 'go to admin dashboard button' from the admin profile to navigate to this page.

Elements

- **Username**
  - **Type :** text
  - **Label:** Username
  - **Content :** To enter username

- **Password**
    - **Type :** password
    - **Label:** Password
    - **Content :** To enter password
- **Login**
    - **Type :** button (submit)
    - **Label:** Log in
    - **Content :** It is used to navigate to the admin dashboard if the login details are correct.

7.1.2. Customer login

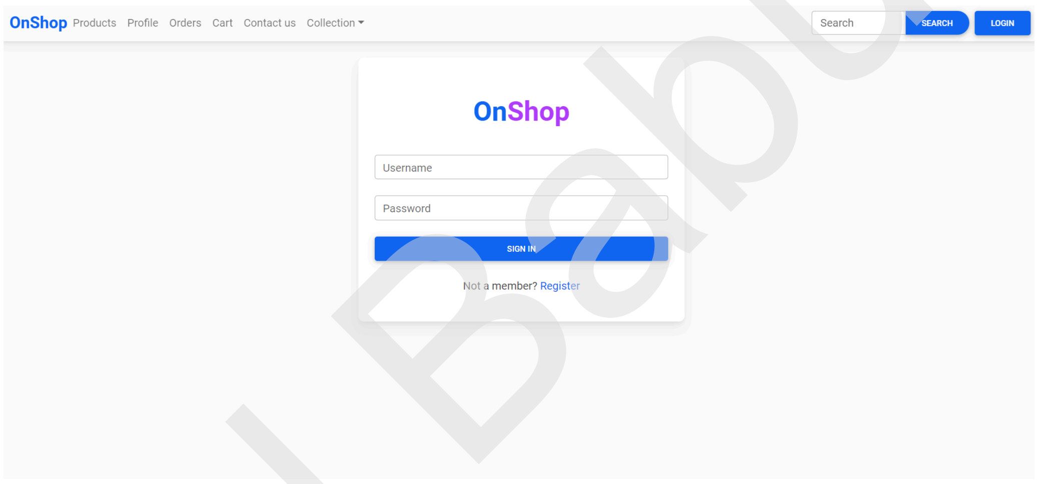


Figure 7.2 customer login

Purpose

This is a customer login page. Customers can login by providing username and password.

Navigation

Customers can click the login button in the top left of the navbar to navigate to this window.

Elements

- **Username**
    - **Type :** text
    - **Label:** username
    - **Content :** To enter username

- **Password**
    - **Type :** password
    - **Label:** Password
    - **Content :** To enter password
- **SignIn**
    - **Type :** button (submit)
    - **Label:** SignIn
    - **Content :** It is used to navigate to the customer profile if the login details are correct.

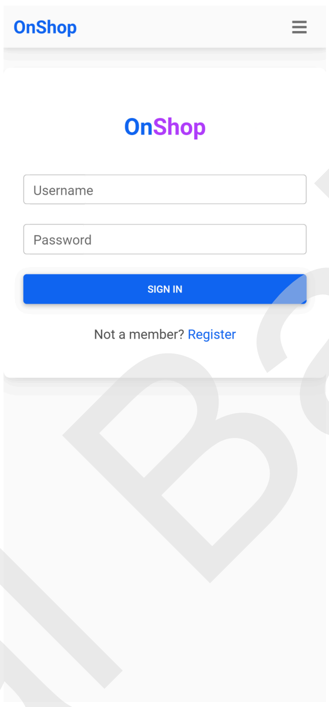


Figure 7.3 customer signin mobile screen view

Customer signup(registration)

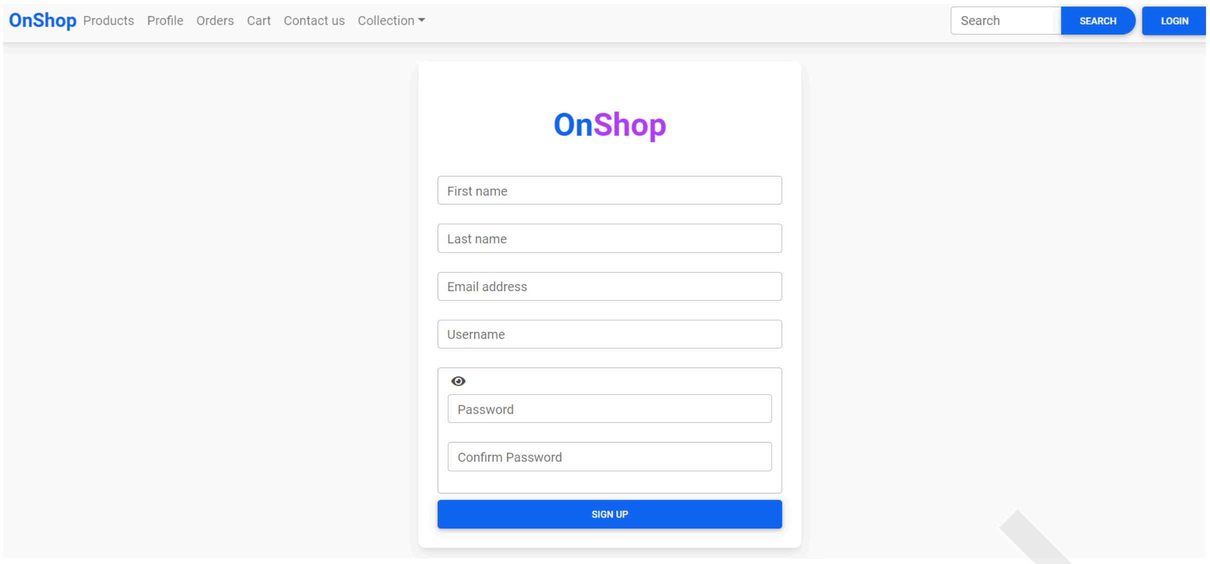


Figure 7.4 customer signup

Purpose

This is a customer signup component/page . A new customer can register by entering valid information.

Navigation

From signin page customer can navigate to this page ,by clicking register link

Elements

- **FirstName**
  - **Type :** Text
  - **Label:** FirstName
  - **Content :** To enter first name of customer
- **LastName**
  - **Type :** Text
  - **Label:** LastName
  - **Content :** To enter last name of customer
- **Email**
  - **Type :** Text
  - **Label:** Email
  - **Content :** To enter email

- **Username**
    - **Type :** Text
    - **Label:** Username
    - **Content :** To enter username
- **Password**
    - **Type :** password
    - **Label:** Password
    - **Content :** To enter email
- **ConfirmPassword**
    - **Type :** password
    - **Label:** ConfirmPassword
    - **Content :** To enter confirm password
- **SignUp**
    - **Type :** button (submit)
    - **Label:** SignUp
    - **Content :** It is used to navigate to the login page if the customer successfully registered.

7.2. Home

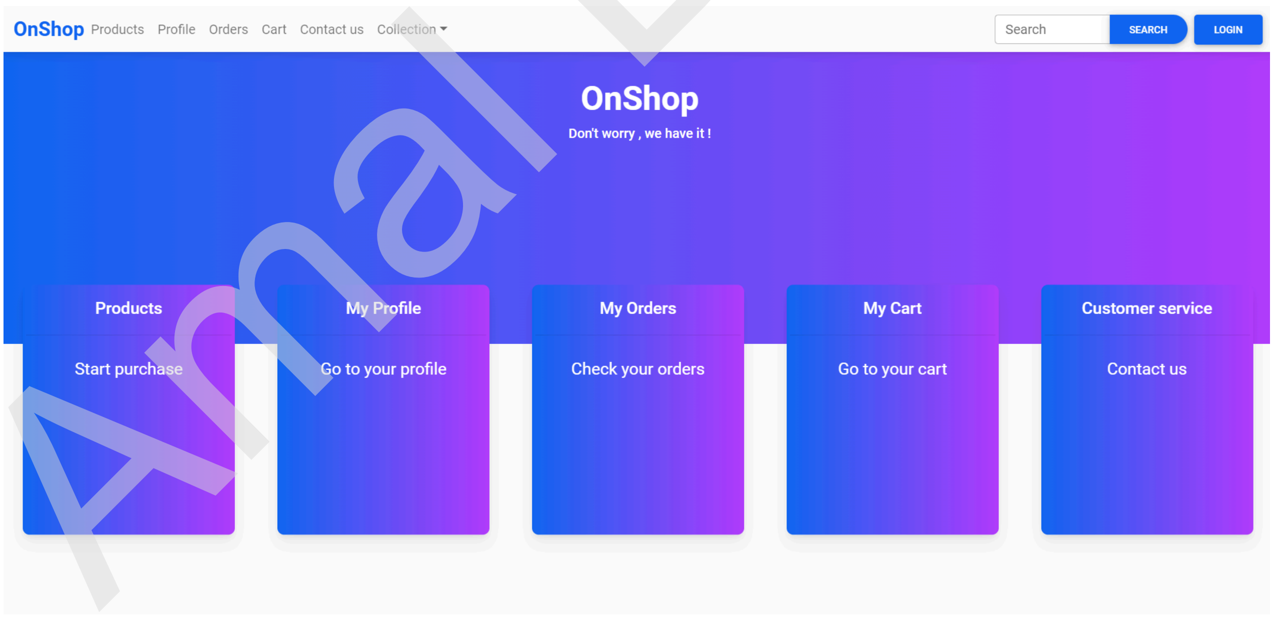


Figure 7.5: Home page

Purpose

This is the home screen of the application. From this page user can navigate to different pages of the application.

Navigation

Customers can click the ‘OnShop’ logo in the main navbar to navigate to this page.

7.3. Menu

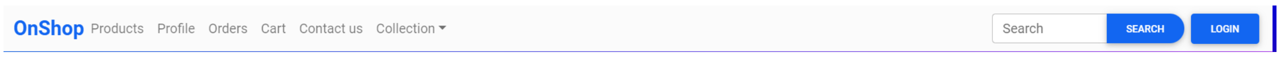


Figure 7.6: main menu

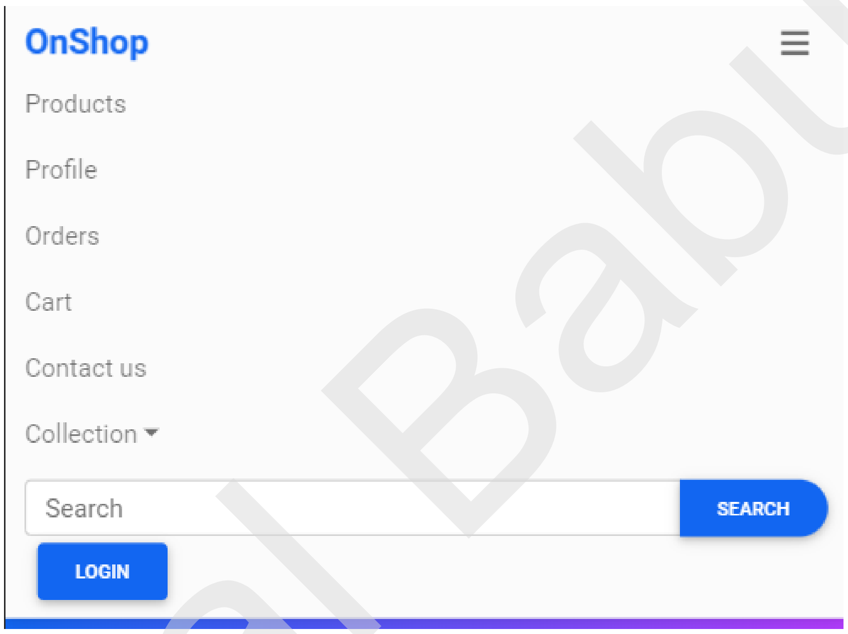


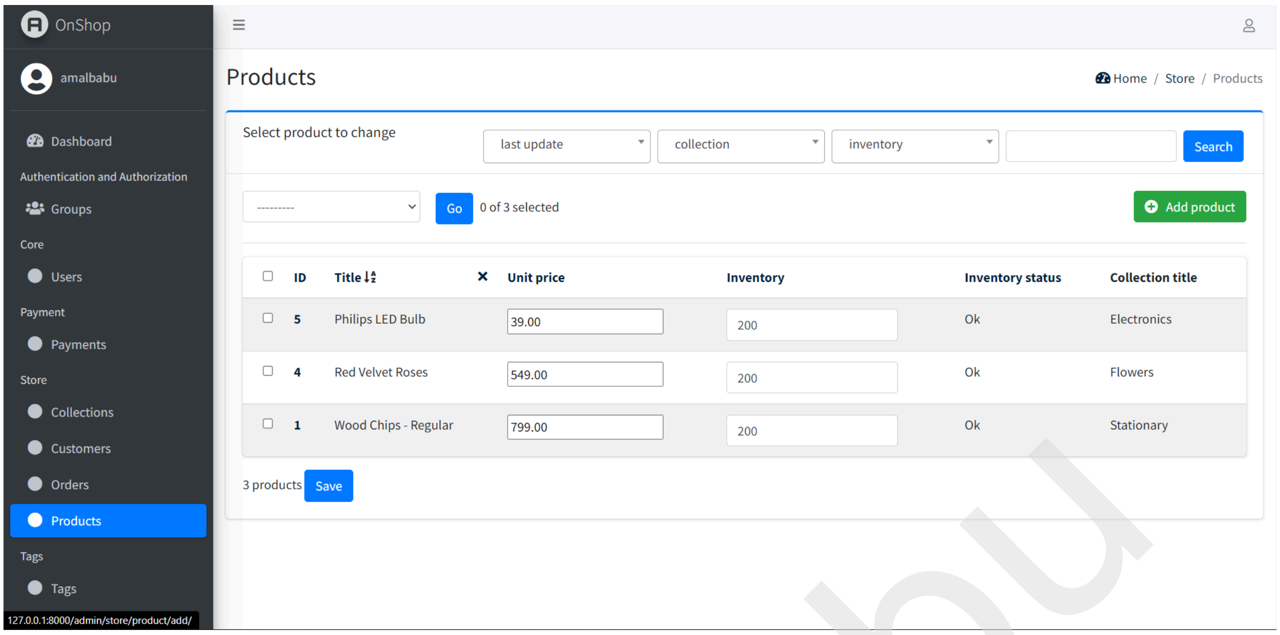
Figure 7.7: main menu smartphone view

Purpose

This is the main menu/navbar of the application.it includes all main navigation in the application.

7.4. Purpose: Data store/ retrieval/ update

7.4.1. Product



Products

Select product to change

0 of 3 selected

ID	Title	Unit price	Inventory	Inventory status	Collection title
5	Philips LED Bulb	39.00	200	Ok	Electronics
4	Red Velvet Roses	549.00	200	Ok	Flowers
1	Wood Chips - Regular	799.00	200	Ok	Stationary

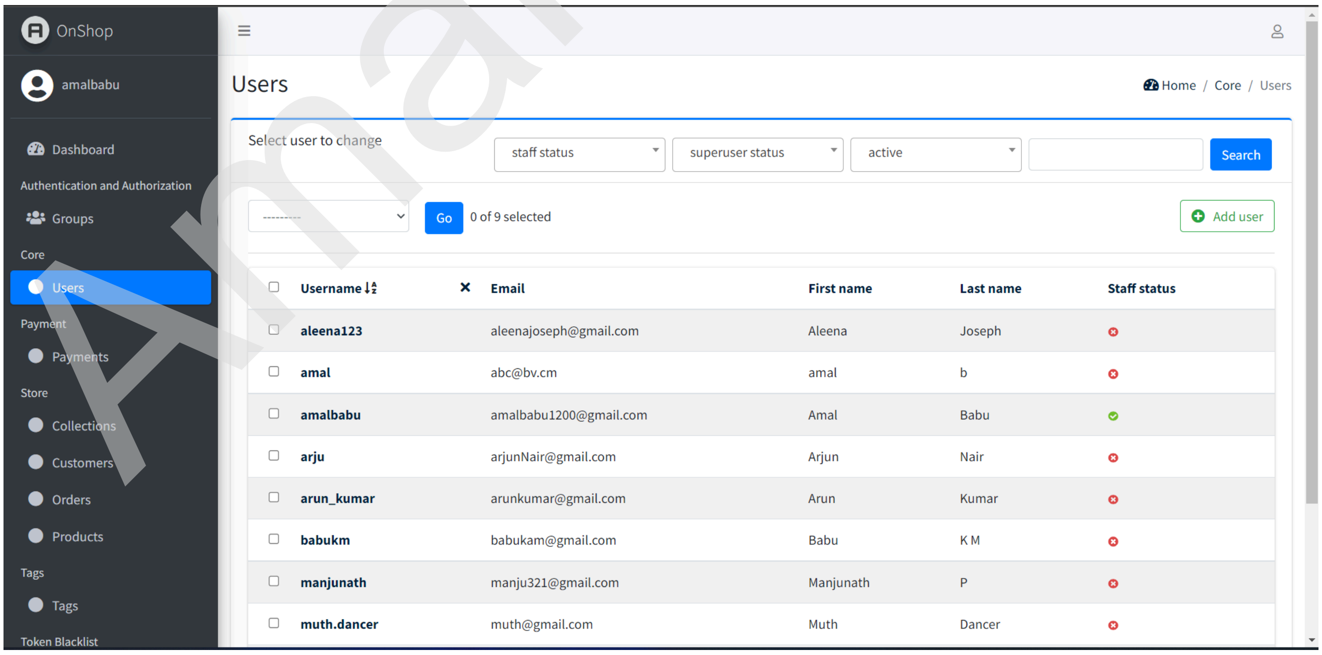
3 products

Figure 7.8 product retrieve update

Purpose

This is product management page. From here admin/staff can add/update/retrieve/delete products.

7.4.2. User



Users

Select user to change

0 of 9 selected

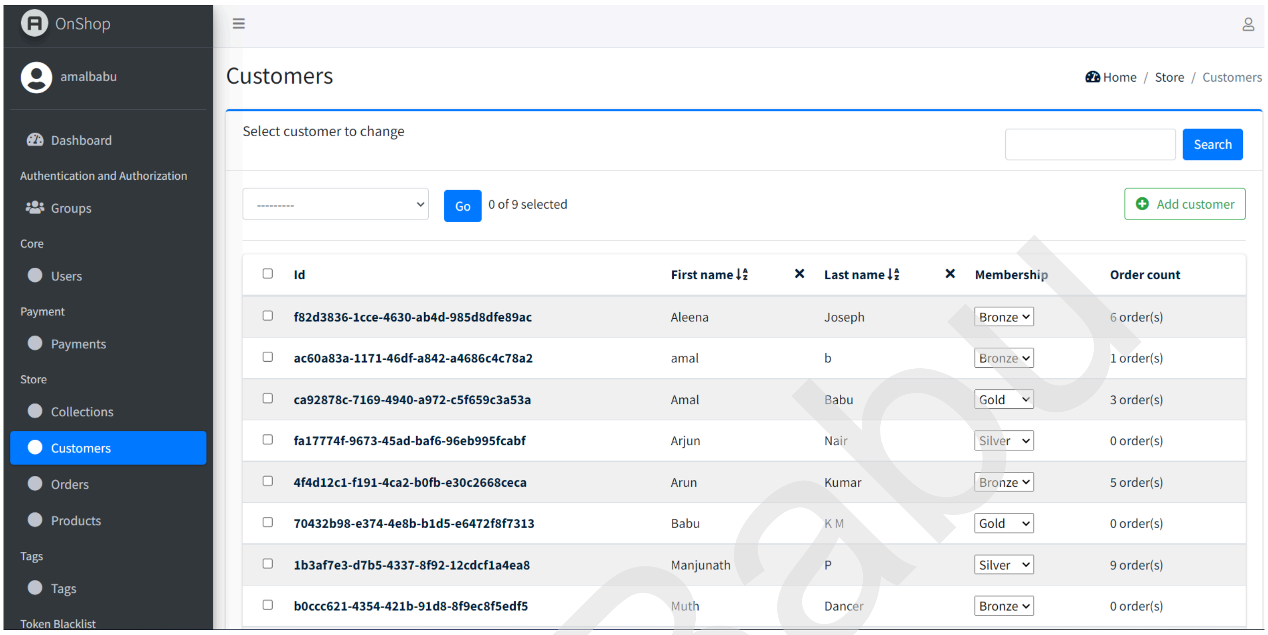
Username	Email	First name	Last name	Staff status
aleena123	aleenajoseph@gmail.com	Aleena	Joseph	✗
amal	abc@bv.cm	amal	b	✗
amalbabu	amalbabu1200@gmail.com	Amal	Babu	✓
arju	arjunNair@gmail.com	Arjun	Nair	✗
arun_kumar	arunkumar@gmail.com	Arun	Kumar	✗
babukm	babukam@gmail.com	Babu	K M	✗
manjunath	manju321@gmail.com	Manjunath	P	✗
muth.dancer	muth@gmail.com	Muth	Dancer	✗

Figure 7.9 user retrieve update

Purpose

This is the User manage interface in the admin panel. From here admin can add or view updates and retrieve users.

7.4.3. Customer



Customers

Select customer to change

--------- 0 of 9 selected

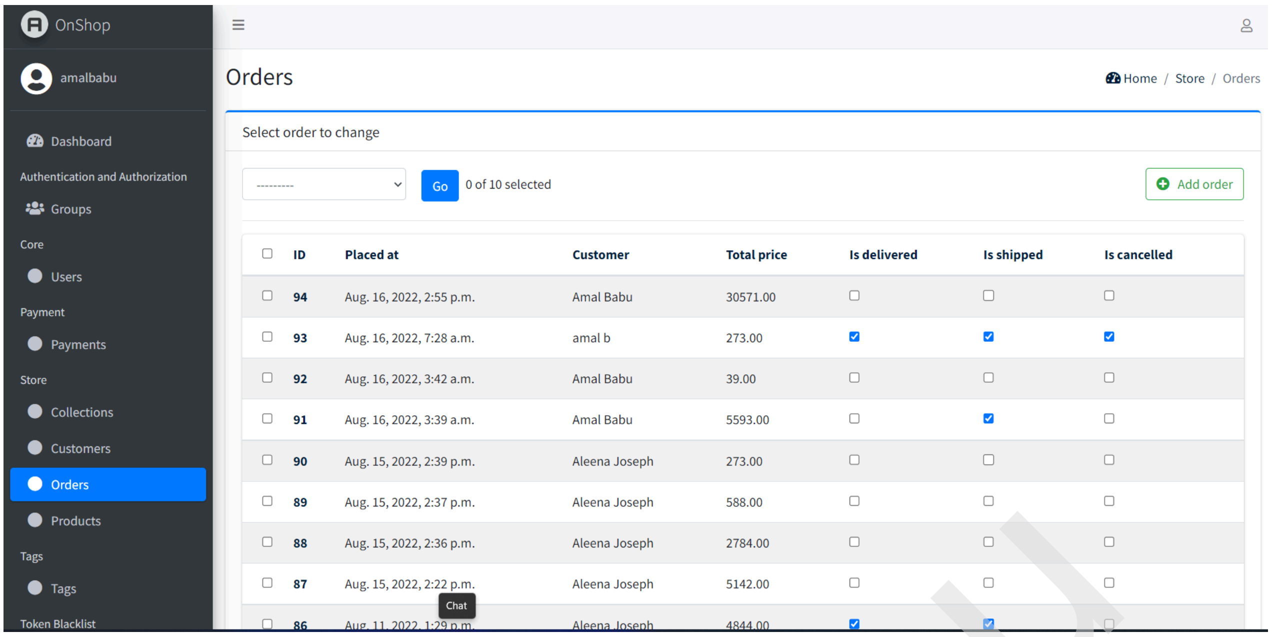
Id	First name	Last name	Membership	Order count
f82d3836-1cce-4630-ab4d-985d8dfe89ac	Aleena	Joseph	Bronze	6 order(s)
ac60a83a-1171-46df-a842-a4686c4c78a2	amal	b	Bronze	1 order(s)
ca92878c-7169-4940-a972-c5f659c3a53a	Amal	Babu	Gold	3 order(s)
fa17774f-9673-45ad-baf6-96eb995fcabf	Arjun	Nair	Silver	0 order(s)
4f4d12c1-f191-4ca2-b0fb-e30c2668ceca	Arun	Kumar	Bronze	5 order(s)
70432b98-e374-4e8b-b1d5-e6472f8f7313	Babu	K M	Gold	0 order(s)
1b3af7e3-d7b5-4337-8f92-12cdcf1a4ea8	Manjunath	P	Silver	9 order(s)
b0ccc621-4354-421b-91d8-8f9ec8f5edf5	Muth	Dancer	Bronze	0 order(s)

Figure 7.10 Customer retrieve update

Purpose :

This is the Customer manage interface in the admin panel. From here admin can add or view updates and retrieve customer details.

7.4.4. Orders



Orders

Select order to change

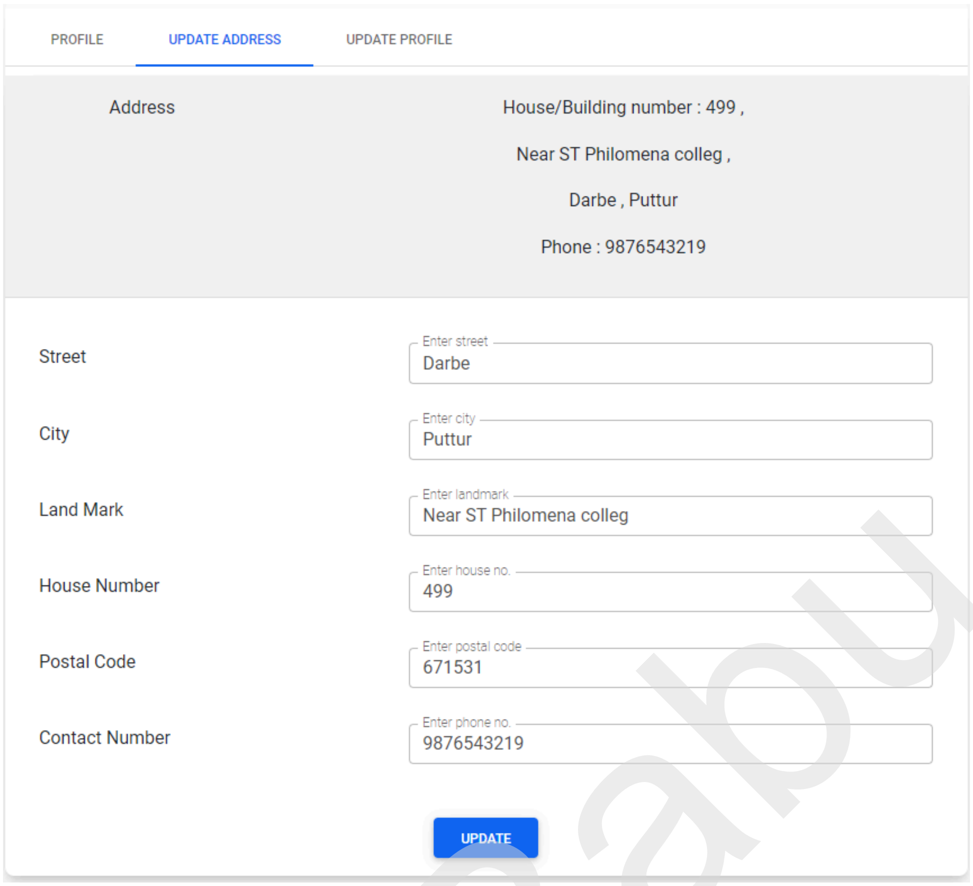
--------- 0 of 10 selected

☐	ID	Placed at	Customer	Total price	Is delivered	Is shipped	Is cancelled
☐	94	Aug. 16, 2022, 2:55 p.m.	Amal Babu	30571.00	☐	☐	☐
☐	93	Aug. 16, 2022, 7:28 a.m.	amal b	273.00	☑	☑	☑
☐	92	Aug. 16, 2022, 3:42 a.m.	Amal Babu	39.00	☐	☐	☐
☐	91	Aug. 16, 2022, 3:39 a.m.	Amal Babu	5593.00	☐	☑	☐
☐	90	Aug. 15, 2022, 2:39 p.m.	Aleena Joseph	273.00	☐	☐	☐
☐	89	Aug. 15, 2022, 2:37 p.m.	Aleena Joseph	588.00	☐	☐	☐
☐	88	Aug. 15, 2022, 2:36 p.m.	Aleena Joseph	2784.00	☐	☐	☐
☐	87	Aug. 15, 2022, 2:22 p.m.	Aleena Joseph	5142.00	☐	☐	☐
☐	86	Aug. 11, 2022, 1:29 p.m.	Aleena Joseph	4844.00	☑	☑	☐

Figure 7.11 Order retrieve update

Purpose : This is the Order manage interface in the admin panel. From here admin can add or view updates and retrieve all orders.

7.4.5. Update address



Address

House/Building number : 499 ,

Near ST Philomena colleg ,

Darbe , Puttur

Phone : 9876543219

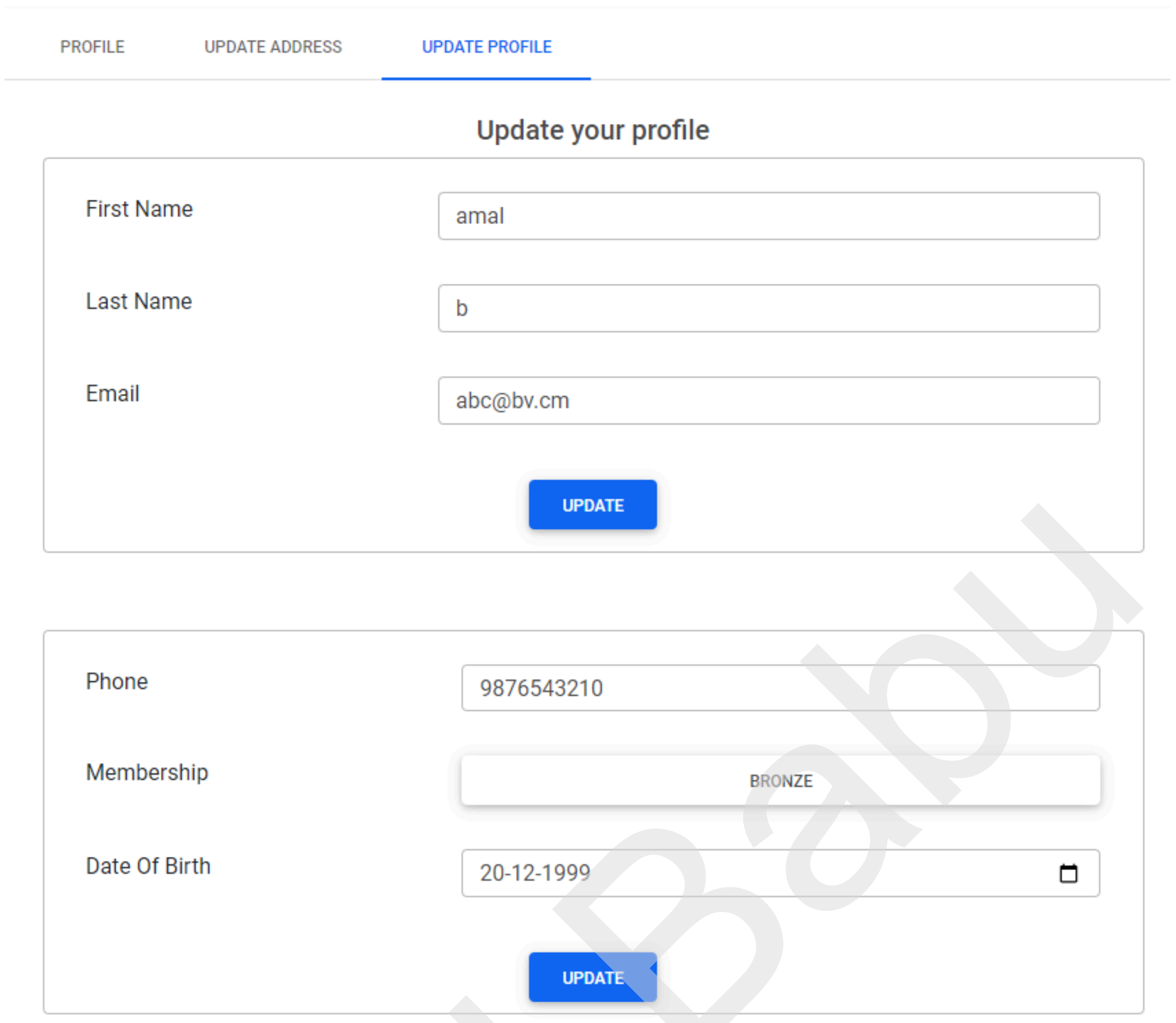
Street	Enter street: Darbe
City	Enter city: Puttur
Land Mark	Enter landmark: Near ST Philomena colleg
House Number	Enter house no.: 499
Postal Code	Enter postal code: 671531
Contact Number	Enter phone no.: 9876543219

Figure 7.12 address retrieve update

Purpose

This is an address update card. From this UI customer can edit addresses by providing valid information.

7.4.6. Update profile



First Name	amal
Last Name	b
Email	abc@bv.cm

Phone	9876543210
Membership	BRONZE
Date Of Birth	20-12-1999

Figure 7.13 profile update

Purpose: This is customer profile update card. from here customer can view and update personal information by providing valid information.

7.5. Validation

7.5.1. SignIn

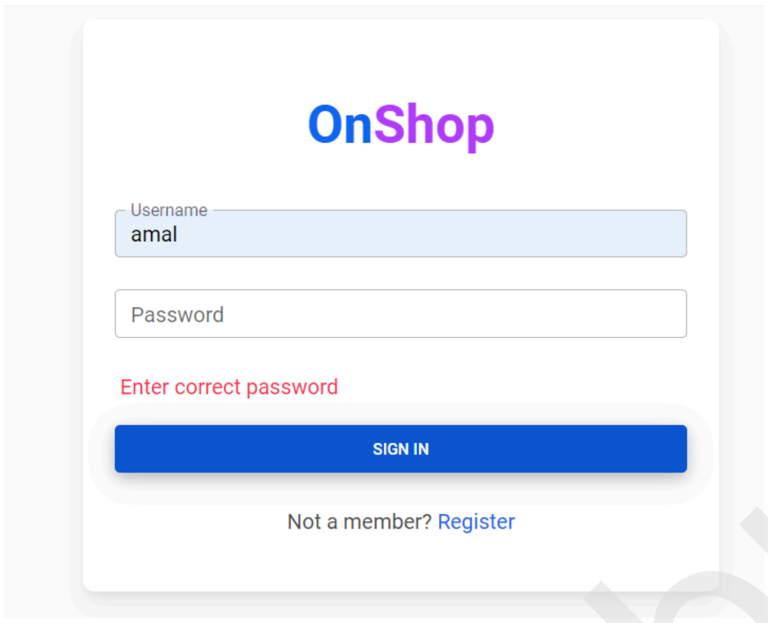


Figure 7.14 signin

Purpose: Customer can login by entering username and password from this window.

7.5.2. SignUp

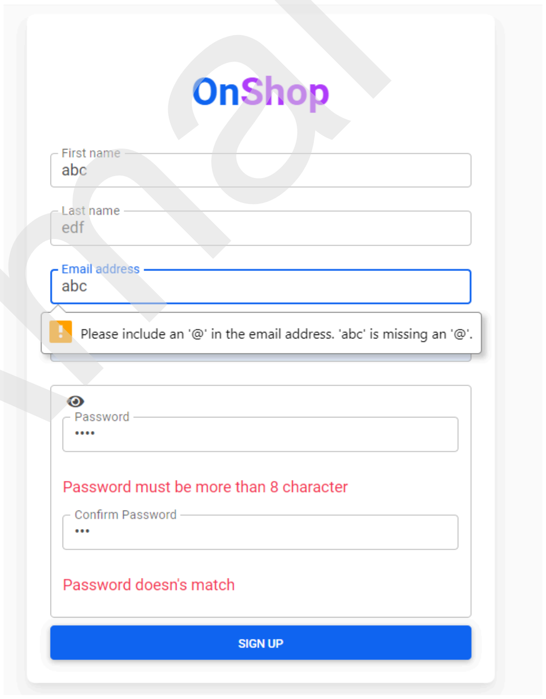


Figure 7.15 signup

Purpose

This is a customer signup component/page . A new customer can register by entering valid information.

Navigation

From signin page customer can navigate to this page ,by clicking register link

Elements

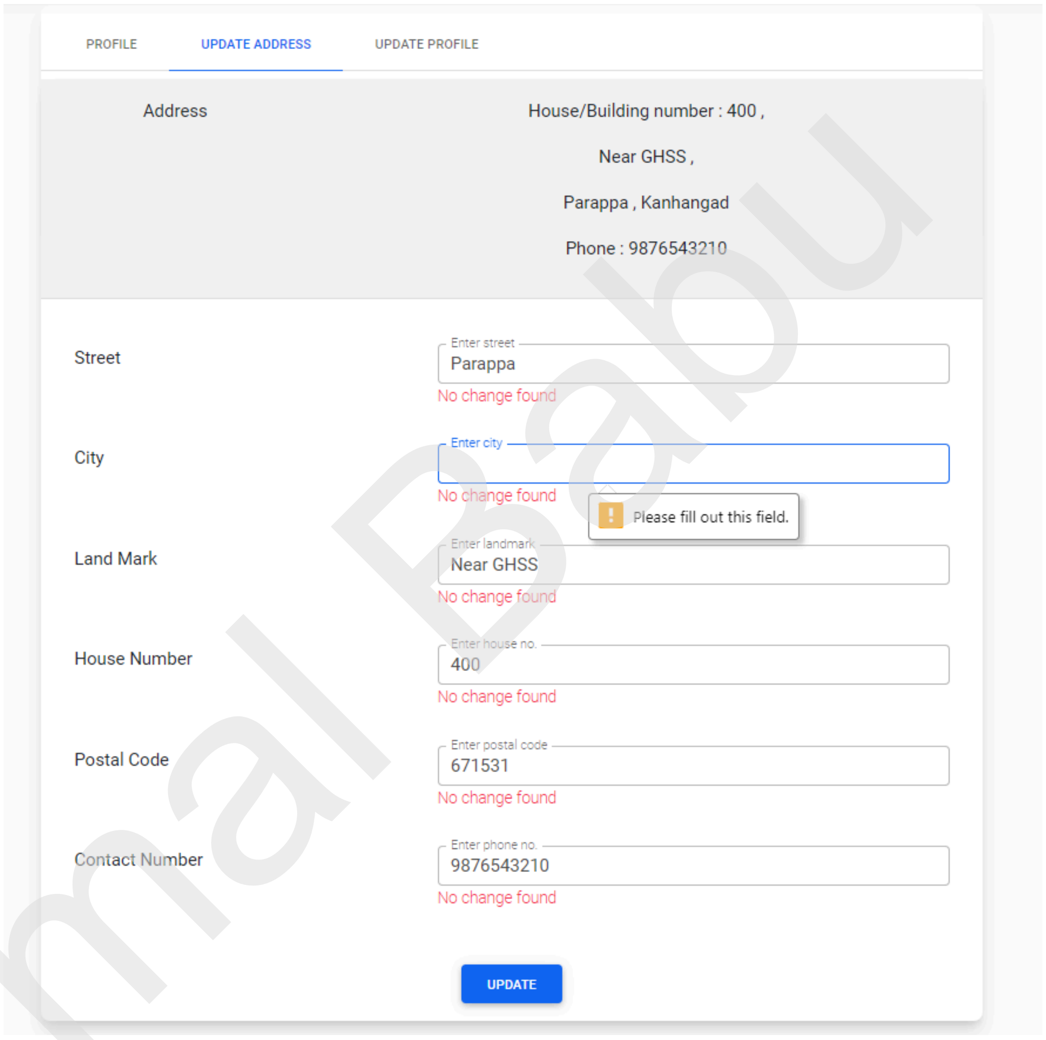
- **FirstName**
    - **Type :** Text
    - **Label:** FirstName
    - **Content :** To enter first name of customer
- **LastName**
    - **Type :** Text
    - **Label:** LastName
    - **Content :** To enter last name of customer

- **Email**
    - **Type :** Text
    - **Label:** Email
    - **Content :** To enter email

- **Username**
    - **Type :** Text
    - **Label:** Username
    - **Content :** To enter username
- **Password**
    - **Type :** password
    - **Label:** Password
    - **Content :** To enter email
- **ConfirmPassword**
    - **Type :** password
    - **Label:** ConfirmPassword
    - **Content :** To enter confirm password

- **SignUp**
  - **Type :** button (submit)
  - **Label:** SignUp
  - **Content :** It is used to navigate to the login page if the customer successfully registered.

7.5.3. address



Address
House/Building number : 400 ,
Near GHSS ,
Parappa , Kanhangad
Phone : 9876543210

Street	Enter street: Parappa — No change found
City	Enter city: — No change found — Please fill out this field.
Land Mark	Enter landmark: Near GHSS — No change found
House Number	Enter house no.: 400 — No change found
Postal Code	Enter postal code: 671531 — No change found
Contact Number	Enter phone no.: 9876543210 — No change found

Figure 7.16 address

Purpose

This is customer address card/page. From this customer can view update/add address.

Navigation

From the profile page by clicking the update address tab, customers can navigate to this page.

Elements

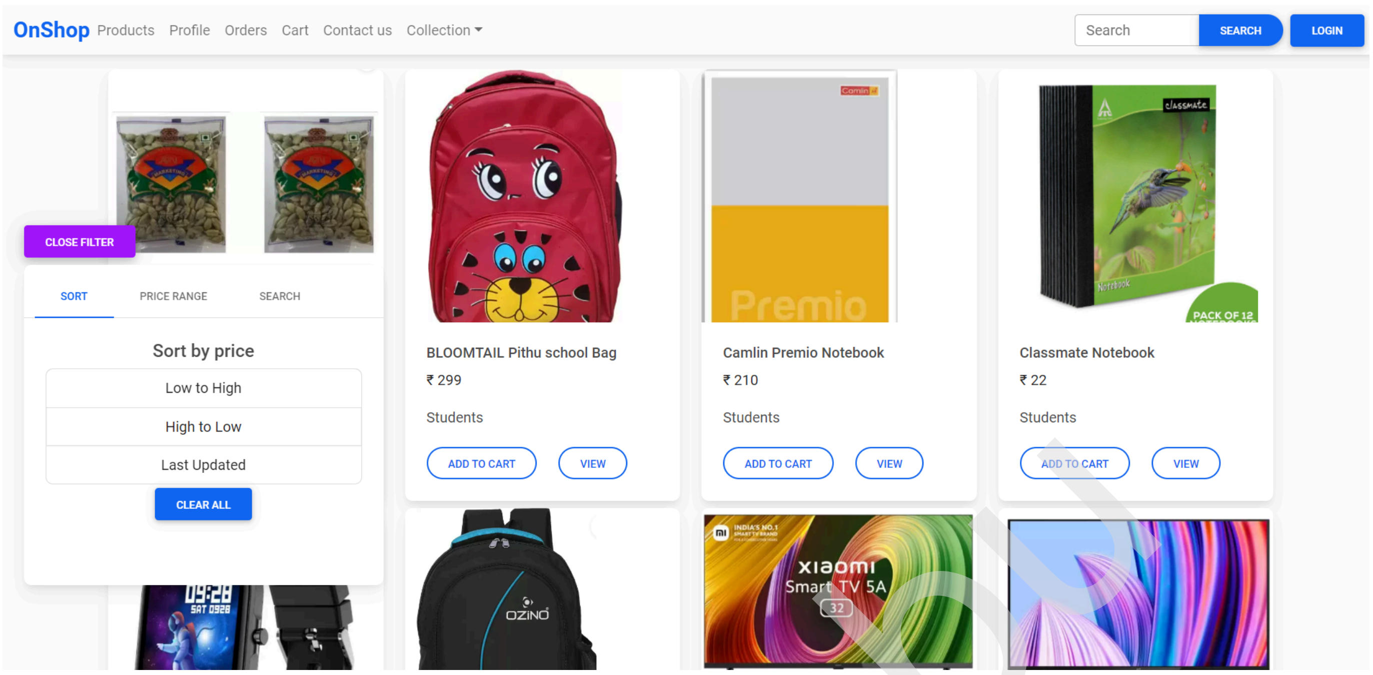
- **Street**
  - **Type :** Text
  - **Label:** Enter street
  - **Content :** To enter street of customer
- **City**
  - **Type :** Text
  - **Label:** Enter city
  - **Content :** To enter city of customer

- **Landmark**
  - **Type :** Text
  - **Label:** Enter landmark
  - **Content :** To enter landmark

- **HouseNumber**
  - **Type :** Number
  - **Label:** Enter house number
  - **Content :** To enter house number
- **PostalCode**
  - **Type :** Number
  - **Label:** Postal code
  - **Content :** To enter postal code
- **ContactNumber**
  - **Type :** Number
  - **Label:** Enter contact number.
  - **Content :** To enter contact number(phone)
- **Update/Add**
  - **Type :** button (submit)
  - **Label:** Update/Add
  - **Content :** It is used to send PUT request to update address.

7.6. View/ data report.

7.6.1. View products



BLOOMTAIL Pithu school Bag
₹ 299
Students

Camlin Premio Notebook
₹ 210
Students

Classmate Notebook
₹ 22
Students

Figure 7.17 product view

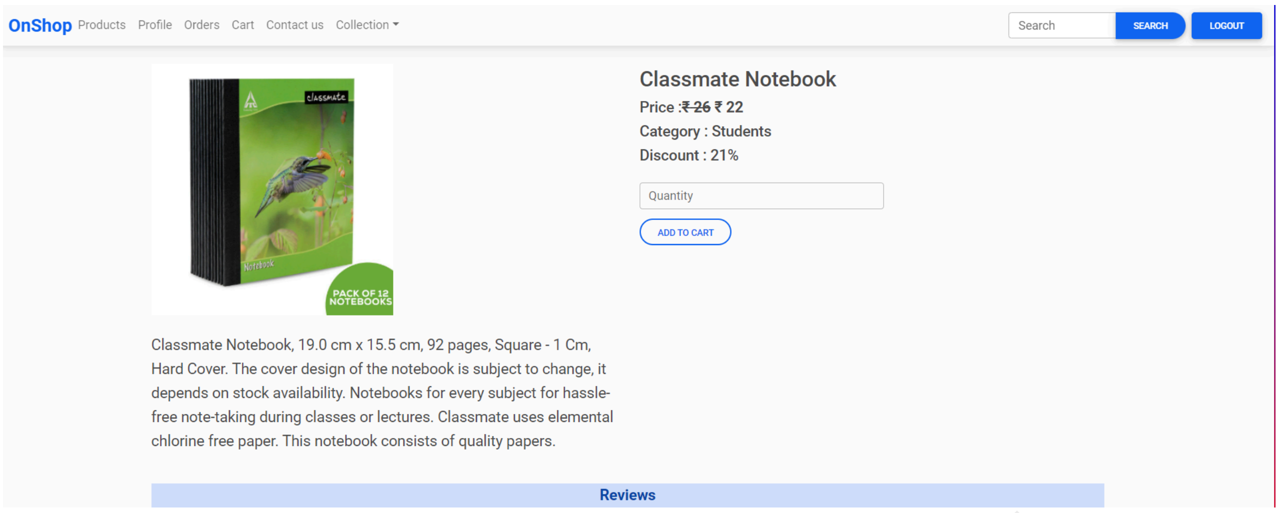
Purpose

This is the product view page. All available products are displayed here. customers can navigate to view product details page or add product to cart from this page.

Navigation

By click the products option in the navbar, customers can navigate to this page.

7.6.2. View product detail



Classmate Notebook

Price :~~₹ 26~~ ₹ 22

Category : Students

Discount : 21%

Classmate Notebook, 19.0 cm x 15.5 cm, 92 pages, Square - 1 Cm, Hard Cover. The cover design of the notebook is subject to change, it depends on stock availability. Notebooks for every subject for hassle-free note-taking during classes or lectures. Classmate uses elemental chlorine free paper. This notebook consists of quality papers.

Figure 7.18 product detail

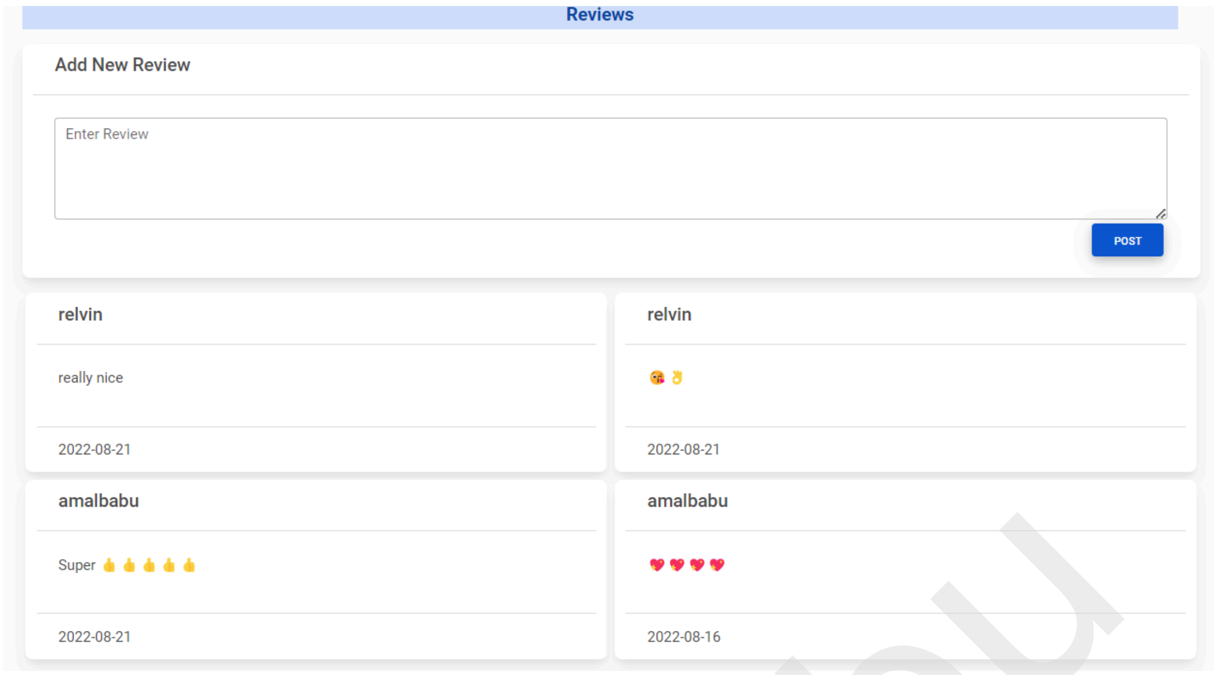
Purpose

This is the product details page. Customers can view product details and reviews from this page. From here, customers can add products to cart.

Navigation

By click the view product button in the product card, customer can navigate to this page.

7.6.3. View reviews


Reviews

Add New Review

Enter Review

POST

relvin

really nice

2022-08-21

relvin

😘👌

2022-08-21

amalbabu

Super 👍👍👍👍👍

2022-08-21

amalbabu

💖💖💖💖

2022-08-16

Figure 7.19 reviews

Purpose

This is review part of product details page. User can add/read review from here.

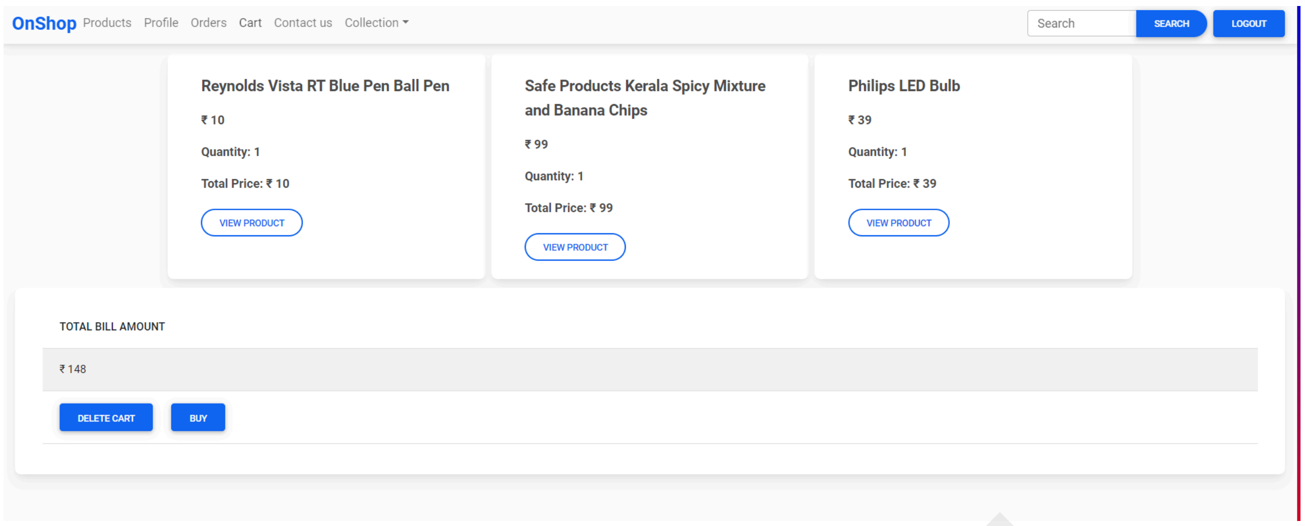
Navigation

By click the view product button in the product card, customer can navigate to this page.

Elements

- **Review description**
    - **Type :** Text
    - **Label:** Enter review
    - **Content :** To enter review
- **Post**
    - **Type :** Text
    - **Label:** post
    - **Content :** To post the review

7.6.4. View cart



Reynolds Vista RT Blue Pen Ball Pen

₹ 10

Quantity: 1

Total Price: ₹ 10

Safe Products Kerala Spicy Mixture and Banana Chips

₹ 99

Quantity: 1

Total Price: ₹ 99

Philips LED Bulb

₹ 39

Quantity: 1

Total Price: ₹ 39

TOTAL BILL AMOUNT

₹ 148

Figure 7.20 cart items

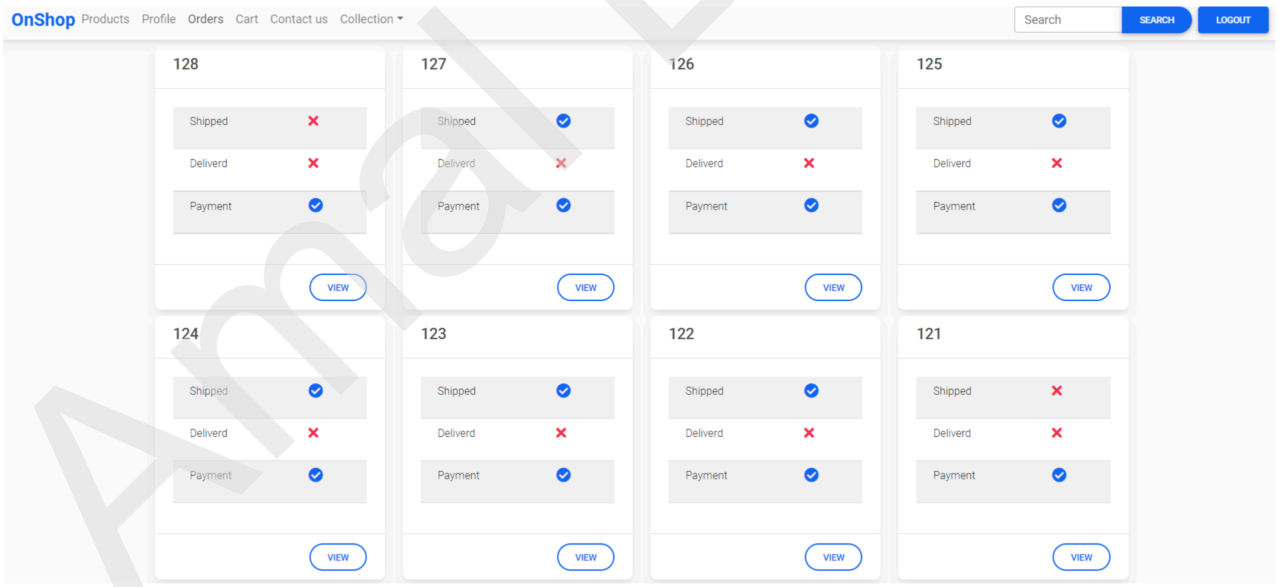
Purpose

This is cart. Customer can view all added cart items.

Navigation

By click the cart option in the navbar, customer can navigate to cart.

7.6.5. View orders



Order	Shipped	Deliverd	Payment
128	✗	✗	✓
127	✓	✗	✓
126	✓	✗	✓
125	✓	✗	✓
124	✓	✗	✓
123	✓	✗	✓
122	✓	✗	✓
121	✗	✗	✓

Figure 7.21 orders

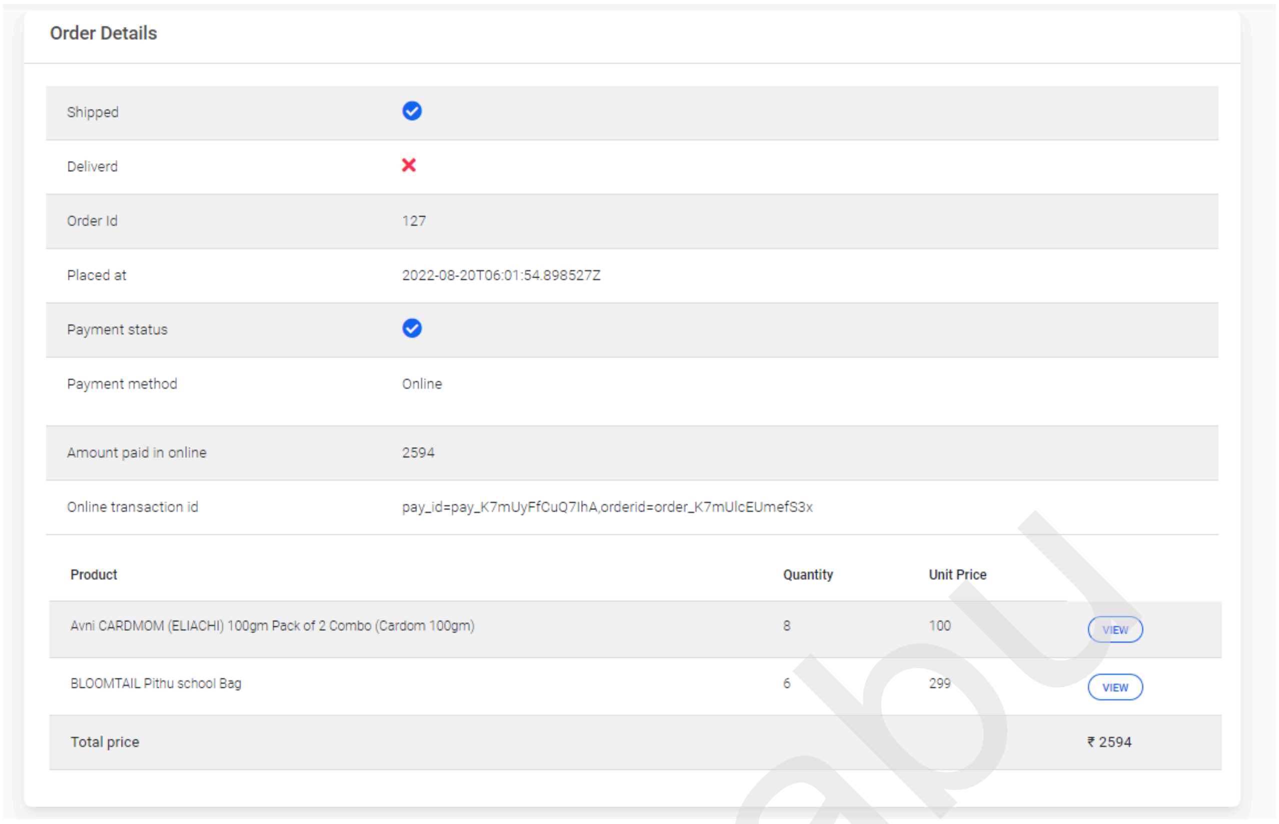
Purpose

This is orders page . Customer can view all orders this page.

Navigation

By clicking the orders option in the navbar, customers can navigate .

7.6.6. View order details



Order Details

Shipped	✓
Deliverd	✗
Order Id	127
Placed at	2022-08-20T06:01:54.898527Z
Payment status	✓
Payment method	Online
Amount paid in online	2594
Online transaction id	pay_id=pay_K7mUyFfCuQ7IhA,orderid=order_K7mUlcEUmefS3x

Product	Quantity	Unit Price	
Avni CARDMOM (ELIACHI) 100gm Pack of 2 Combo (Cardom 100gm)	8	100	VIEW
BLOOMTAIL Pithu school Bag	6	299	VIEW
Total price			₹ 2594

Figure 7.22 order details

Purpose

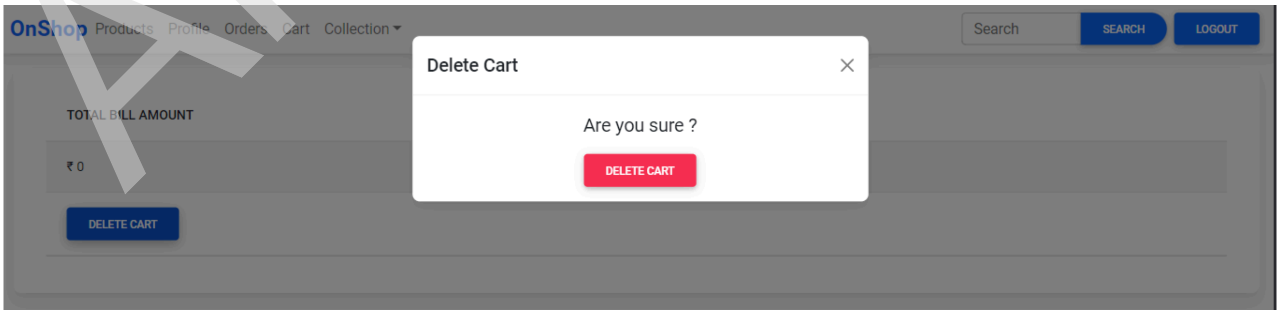
This is orders-detail page . Customer can view all orders-details this page.

Navigation

By clicking the view option in the order card, customers can navigate .

7.7. Alerts

7.7.1. Delete Cart



Delete Cart

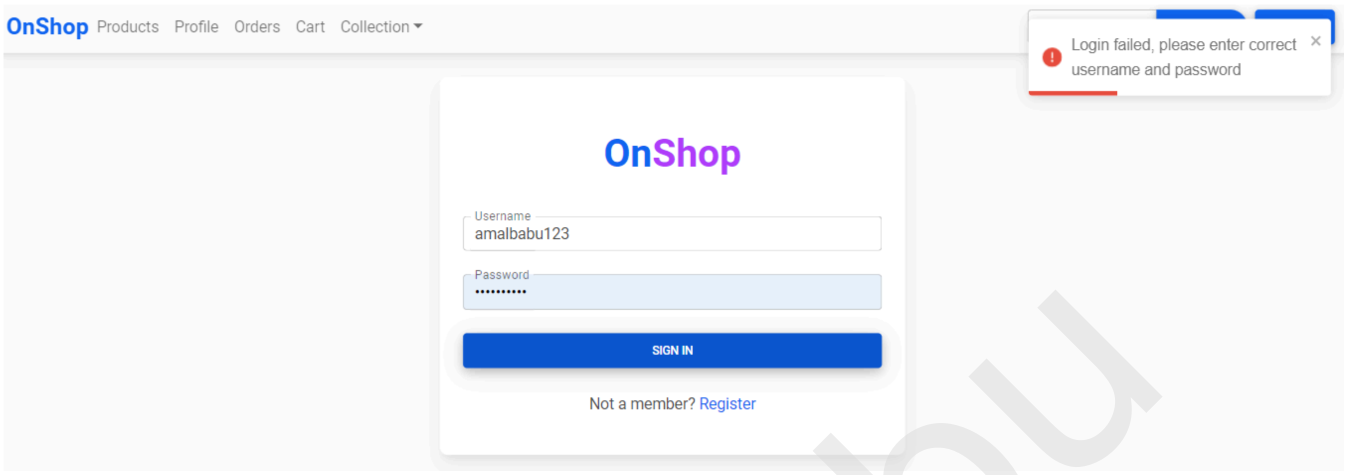
Are you sure ?

DELETE CART

Figure 7.23 cart delete alert

Purpose: When you click the delete cart button in the cart this alert will popup. If the user clicks the delete cart button then system sends delete request to server with cart_id.

7.7.2. Login failed

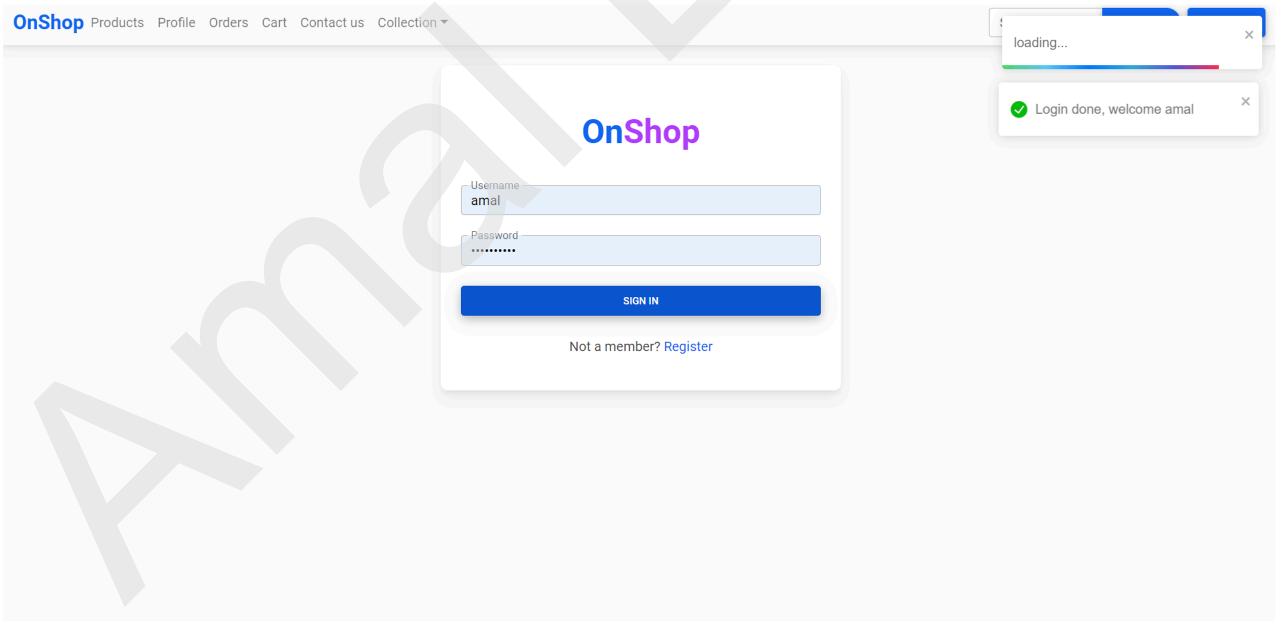


Login failed, please enter correct username and password

Figure 7.24 login failed alert

Purpose: If user enter wrong password or username this toast alert will display.

7.7.3. Delete success



loading...

Login done, welcome amal

Figure 7.25 login success alert

Purpose: If the user entered the correct username and password a success toast will display.

****

Chapter - 8

8. Testing

8.1. Introduction

Software testing is an investigation conducted to provide stack holders with information about the quality of the product or service under test. Testing has been defined as the process of analysing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item. Software testing is the process used to assess the quality of computer software.

It involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they should. It is oriented to 'detection'.

8.2. Test reports

8.2.1. Unit testing

In this testing, each unit is tested separately or individually to know its performance when checked solely. Any faults in each unit are corrected in this phase.

8.2.1.1. Admin login

Table 8.1: Admin login test

Serial No.	**Condition to be tested**	**Test Data**	**Expected output**	**Test Result**
1.	If username is not entered	abc124	Please enter valid username	SUCCESSFUL
2.	If password is not entered	apassword	Please enter a valid password.	SUCCESSFUL

3.	If username and password are not valid	name, apassword	Invalid username or password	SUCCESSFUL

8.2.1.2. Admin add product

Table 8.2:Add product test

Serial No.	**Condition to be tested**	**Test Data**	**Expected output**	**Test Result**
1	If title,description, unit_price,inventory, collection fields are empty	not data entered	This field is required.	SUCCESSFUL
2	If unit_price is negative	-1	Ensure the value is greater than or equal to 1.	SUCCESSFUL
3	If all details are entered properly	bag, black bag, 100, 200, student_collection	product added successfully.	SUCCESSFUL

8.2.1.3. Add collection

Table 8.3 : add collection test

Serial No.	Condition to be tested	Test Data	Expected output	Remarks
1.	If collection title is not entered	-	This field is required.	SUCCESSFUL
2.	If Title is entered.	test	Collection added successfully	SUCCESSFUL

8.2.1.4. Add user

Table 8.4 : Add user test

Serial No.	Condition to be tested	Test Data	Expected output	Remarks
1.	If fields are empty.	test	This field is empty.	SUCCESSFUL
2	If password less than 8	abc	Password should be greater than 8,	SUCCESSFUL
3	If password and username are similar	amal1234567	password is similar to username	SUCCESSFUL
4	If password is entirely numeric	123456789	Passwords are too common	SUCCESSFUL

			and entirely numeric.	
5	If email is in invalid format	abc@.cm	Email is not valid	SUCCESSFUL
6	If all fields are entered properly		User added	SUCCESSFUL

8.2.1.5. customer login

Table 8.5 customer login test

Serial No.	Condition to be tested	Test Data	Expected output	Test Result
1.	If username is not entered	abc124	Please enter valid username	SUCCESSFUL
2.	If password is not entered	apassword	Please enter a valid password.	SUCCESSFUL
3.	If username and password are not valid	name, apassword	Invalid username or password	SUCCESSFUL

8.2.1.6. Customer signup

Table 8.6: customer signup test

Serial No.	Condition to be tested	Test Data	Expected output	Test Result
1.	If form fields are empty	-	This Field is required	SUCCESSFUL
2.	If password less than 9	p123	password should be more than 8	SUCCESSFUL
3.	If password and confirm password doesn't match	password123, password456	password doesn’t match	SUCCESSFUL
4.	If all details are properly entered.	fname,lname, uname, email@gmail.com passwo12346, passwo12346	Signup success	SUCCESSFUL

8.2.2. Integration Testing

In this type of testing, two or more modules are combined together and tested for their performance. When two or more modules are integrated, the working of those modules along with each other is identified. If the modules don’t perform well in this phase they are tested again individually and then modified and integrated again.

8.2.2.1. Customer update address

Table 8.7: update address test

Serial No.	**Condition to be tested**	**Test Data**	**Expected output**	**Test Result**
1.	If all fields are empty	-	This Field is required	SUCCESSFUL
2.	If postal code invalid format	1234	Not valid format	SUCCESSFUL
3.	If Phone number is only 9 digit	987654321	Enter valid contact number	SUCCESSFUL
4.	If all details are properly entered.	House/Building number : 499, Near ST Philomena college Darbe , Puttur Phone : 9876543219	updated	SUCCESSFUL

8.2.3. System Testing

The system testing is the last level of testing, here all modules in system are put together and tested for any errors and ambiguities. If the system performs well then it is further processed and the system is approved otherwise the testing phase is performed again until the system performs correctly.

Table 8.7: system test

Test Case ID	Date Tested	Test Conditions	Pass/Fail	Severity of Defect
1	12/08/2022	System loading	Pass	No
2	12/08/2022	System Run Procedure	Pass	No
3	12/08/2022	File I/O Operation	Pass	No
4	12/08/2022	Database Communication	Pass	No
5	12/08/2022	Server/Client Interaction	Pass	No
6	12/08/2022	Memory Usage	Pass	No
7	12/08/2022	System Processor usage	Pass	No
8	12/08/2022	Authentication / Authorization	Pass	No

****

Conclusion

Working on this project is a good experience. We understand the importance of planning and designing as part of software development. When the website is implemented, it will ensure the perfect e-commerce online shopping system. The system is developed in such a way that the user with common knowledge of computers can handle it easily. The module has a user-friendly interface. The reports requested by the client have been generated and all documentation required for operation and maintenance of the module has been provided. The future enhancement to the system can be made as technology improves or changes.

This system developed by using Django and Django rest framework in the backend and reactjs as frontend. So all services of this system can be accessed through RESTful apis. In the future it is easy to connect a mobile app to the same backend.

LIMITATION

The limitations of this application are as follows:

- Forgot password handling option is not available, for this customer should contact the customer service.
- Customers can add only one address to the profile.
- Customers can't add profile pictures.
- Reviewers can't delete or edit their review.
- Only one image of the product is possible to view.
- Email verification of customers (email-authentication) not included in this version.

SCOPE FOR ENHANCEMENT (FUTURE SCOPE)

In future Enhancements that are possible in the project are as follows.

- In the area of data security and system security.
- Forgot password option can be added.
- Mobile applications(android/ios) can be added.
- In the area of authentication signin with more options can be added (google,facebook,email-verification)
- Real Time inventory management.
- Email notification after successful order.
- New module for handling sellers and shipping agencies.
- Sub category for main product categories.

ABBREVIATIONS AND ACRONYMS

- SRS: Software Requirement Specification CFD: Context Flow Diagram.
- DFD: Data Flow Diagram.
- HTTP: Hypertext Transfer Protocol.
- I/O: Input/Output.
- OS: Operating System.
- DFD: Data Flow Diagram.
- CSS: Cascading Style Sheet.
- ADMIN: The Administrator.
- CFD: Control Flow Diagram.
- CPU: Central Processing Unit.
- GUI: Graphical User Interface.
- RAM: Random Access Memory.
- SRS: Software Requirement Specifications.
- DB: Database

BIBLIOGRAPHY / REFERENCES

- Python/Django framework documentation https://www.djangoproject.com/
- Django REST framework documentation https://www.django-rest-framework.org/
- ReactJs documentation https://reactjs.org/
- ReduxJs documentation https://redux.js.org/
- Material Design Bootstrap https://mdbootstrap.com/
- Braintree developer documentation https://developer.paypal.com/braintree/docs
- Other references

→ Django developers google group https://groups.google.com/g/django-users

→ Django forum https://forum.djangoproject.com/

→ Stackoverflow https://stackoverflow.com/

→ Medium django articles.

→ JS coder community https://web.codercommunity.io/g/javascript-community

→ DJOSER docs https://djoser.readthedocs.io/en/latest/

→ JWT docs https://jwt.io/introduction

****