**The Islamia University of Bahawalpur**

**Department of Software Engineering**

****

**SOFTWARE DESIGN DESCRIPTION**

**(SDD DOCUMENT)**

**for**

**SNAPPICK**

##### Version 1.0

***By***

Ameer Hamza

F21NSEEN1M010**01**

Ome Shifa

F21NSEEN1M010**14**

Alisha Abbas

F21NSEEN1M010**16**

**Session Fall 2021 – 2025**

***Supervisor***

**Hafiz M.Irfan Sab**

***Bachelor of Science in Software Engineering***

## Table of Contents

1. [INTRODUCTION 5](#_TOC_250126)
   1. [Purpose 5](#_TOC_250125)
   2. [Scope 5](#_TOC_250124)
   3. [Overview 5](#_TOC_250123)
   4. Definitions and Acronyms 6
   5. [Reference Material 6](#_TOC_250122)
2. [DESIGN METHODOLGY AND SOFTWARE PROCESS MODEL 7](#_TOC_250121)
   1. [Design Methodology 7](#_TOC_250125)
   2. [Process Model 7](#_TOC_250124)
3. [SYSTEM OVERVIEW 8](#_TOC_250121)
   1. [Technology to be used 8](#_TOC_250120)
   2. [Application overview 8](#_TOC_250119)
   3. [Design language 8](#_TOC_250118)
   4. [General constraints 9](#_TOC_250117)
   5. [System Environment 9](#_TOC_250116)
4. [ARCHITECTURAL DESIGN 10](#_TOC_250115)
   1. [Architecture of System 10](#_TOC_250114)
5. [DESIGN MODEL [Along with Description] 11](#_TOC_250096)

5.1 [Class Diagram 11](#_TOC_250095)

5.2 [Sequence Diagram 13](#_TOC_250094)

5.3 [State Transition Diagram (STD) 15](#_TOC_250093)

5.4 [Data Flow Diagram (DFD) 15](#_TOC_250092)

1. [DATA DESIGN 16](#_TOC_250068)
2. ALGORITHIM & [IMPLEMENTAION 20](#_TOC_250068)
3. [REQUIREMENTS MATRIX 25](#_TOC_250000)
4. [HUMAN INTERFACE DESIGN 27](#_TOC_250022)
   1. [Screen Images 28](#_TOC_250021)
5. [Appendix 31](#_TOC_250022)
   1. [Glossary 31](#_TOC_250021)

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

**Application Evaluation History**

|  |  |
| --- | --- |
| **Comments (by committee)**  **\*Include the ones given at scope time both in doc and presentation** | **Action Taken** |
|  |  |
|  |  |

**Supervised by**

**<Supervisor’s Name>**

# INTRODUCTION

## Purpose

The purpose of this Software Design Document (SDD) is to outline the architectural and design details of **SnapPick,** an innovative e-commerce application. This document serves as a guide for developers, stakeholders, and testers to understand the system's overall structure, design considerations, and implementation approach. SnapPick aims to provide a seamless shopping experience for users through a custom-built mobile application and backend system, focusing initially on the **Cash on Delivery (COD)** payment method.

## Scope

SnapPick is an e-commerce application designed to cater to small and medium businesses by providing a personalized platform for online shopping. Unlike generic solutions like Shopify, SnapPick offers tailored features, including:

* A custom Flutter-based mobile app for end-users.
* A PHP-based backend for managing inventory, orders, and user accounts.
* An easy-to-navigate user interface optimized for mobile devices.
* Initial support for **Cash on Delivery (COD),** with plans for future payment method integrations.
* Administrative tools for managing product listings, categories, and customer orders.

The system will target the Pakistani market initially and expand to other regions as needed. It includes modules for user authentication, product catalog, order management, and customer feedback.

## Overview

This document provides a comprehensive view of the design and architectural elements of SnapPick.The primary focus is to ensure that:

* The system design aligns with the functional and non-functional requirements.
* Developers can easily implement the system using the provided specifications.
* Stakeholders can validate the design and offer feedback before development begins.
  1. **Definitions and Acronyms**
* FYP: Final Year Project
* COD: Cash on Delivery
* API: Application Programming Interface
* UI/UX: User Interface/User Experience
* DBMS: Database Management System
* PHP: Hypertext Preprocessor, the backend programming language used
* Flutter: A UI toolkit for building natively compiled mobile applications
* MySQL: Relational Database Management System
* SDD: Software Design Document

## Reference Material

1. "Software Design Document Template," IEEE Recommended Practice for Software Design Descriptions.
2. Flutter Official Documentation - <https://flutter.dev/docs>
3. PHP Official Documentation - <https://www.php.net/docs.php>
4. MySQL Reference Manual - <https://dev.mysql.com/doc/>

# DESIGN METHODOLGY AND SOFTWARE PROCESS MODEL

## Design Methodology

SnapPick follows the **Object-Oriented Programming (OOP)** methodology for system design and development. This choice is justified by the following advantages:

* **Modularity**: Components such as user authentication, product catalog, and order management are developed as independent modules, enhancing maintainability and scalability.
* **Reusability**: Code can be reused across different parts of the application, reducing development time and effort.
* **Ease of Maintenance**: Debugging and updating individual components is more straightforward due to encapsulation and abstraction.
* **Alignment with Flutter**: Flutter's widget-based approach and support for object-oriented principles make OOP a natural fit.

## Process Model

The **Agile Methodology** has been chosen as the process model for SnapPick. This model involves developing the application in incremental cycles, allowing for iterative improvements based on feedback. The reasons for this choice include:

* **Flexibility**: Requirements can evolve over time, and the iterative model accommodates these changes effectively.
* **Early Feedback**: Working prototypes are delivered early in the process, enabling stakeholders to provide feedback that shapes subsequent iterations.
* **Risk Management**: Potential risks are identified and mitigated in earlier stages of development.
* **Suitability for FYP**: The iterative model aligns well with academic projects, allowing for phased development and evaluation.

Each iteration of SnapPick will include activities such as requirement refinement, design updates, implementation, and testing. This approach ensures the delivery of a robust and user-friendly e-commerce platform.

# SYSTEM OVERVIEW

## Technology to be used

SnapPick leverages the following technologies to ensure a robust and scalable system:

* **Frontend**: Flutter framework for building cross-platform mobile applications.
* **Backend**: PHP for server-side scripting and logic implementation.
* **Database**: MySQL for efficient and reliable data storage.
* **APIs**: RESTful APIs to facilitate communication between the frontend and backend.
* **Development Tools**: Visual Studio Code, PHPStorm, and Android Studio.
* **Version Control**: Git for collaborative development and version management.

## Application overview

SnapPick is a feature-rich e-commerce application aimed at providing a seamless shopping experience. Key features include:

* User registration and authentication.
* Product catalog with categories and filters.
* Order placement and tracking.
* Cash on Delivery (COD) as the initial payment method.
* Administrative tools for inventory and order management.

The application is designed to prioritize user experience, scalability, and performance, with plans for future enhancements such as online payment integration.

## Design language

SnapPick employs a modern and intuitive design language based on Flutter’s Material Design components. This ensures:

* Consistency across all screens.
* Enhanced user engagement through responsive and accessible design.
* Aesthetic appeal tailored to the target audience.

Custom color schemes and branding elements are integrated to align with the business identity.

## General constraints

* **Device Compatibility**: The mobile app is optimized for Android devices, with plans for iOS support in the future.
* **Network Dependency**: Requires a stable internet connection for full functionality.
* **Initial Payment Limitation**: Only Cash on Delivery (COD) is supported in the initial release.
* **Time and Budget**: The development timeline and budget constraints of an academic project.

## System environment

* **Development Environment**: Windows 10/11 with necessary IDEs and tools.
* **Production Environment**: Linux-based server for hosting the backend and database.
* **Mobile Platform**: Android 8.0 (Oreo) and above.
* **Database Server**: MySQL 8.0 hosted on a secure server.

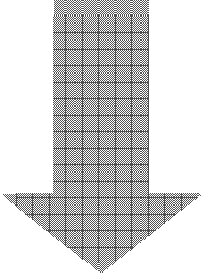
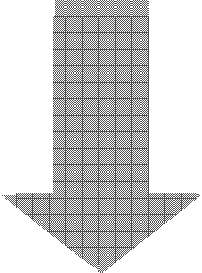
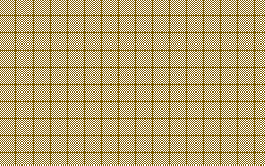
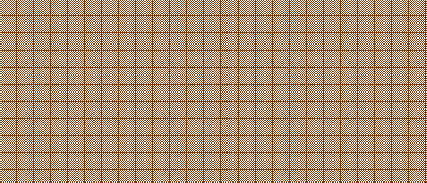
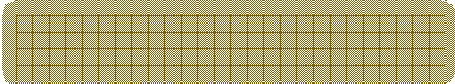
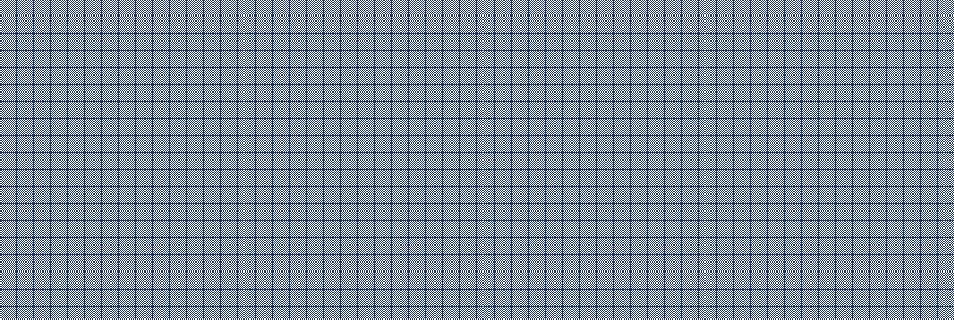
# Architectural Design

* 1. **Architecture of System**

SnapPick’s architecture integrates the following components to create a seamless and efficient e-commerce experience:

* **Internet Connectivity**: Mobile devices connect to the internet using TCP/IP protocols.
* **Mobile Application**: The Flutter-based application communicates with the backend server to fetch and display data to users.
* **Backend Server**: A PHP-based backend handles server-side logic, including user authentication, inventory management, and order processing.
* **Database**: A MySQL database stores and manages data such as user accounts, product details, order history, and feedback.
* **APIs**: RESTful APIs facilitate communication between the frontend application and backend server.

The architectural design ensures secure data transmission, scalability, and performance while maintaining compatibility with various devices and network environments.



Internet

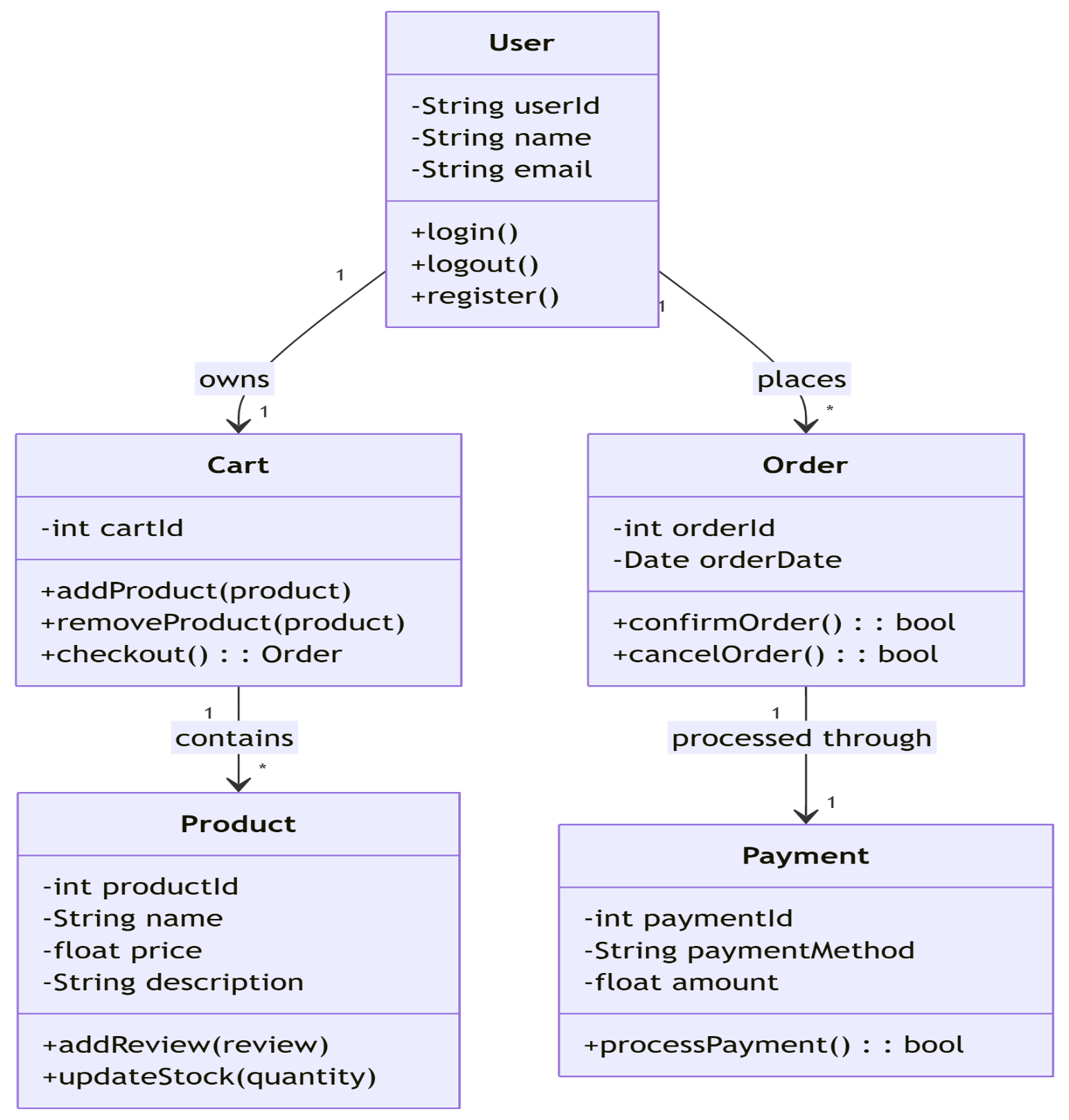
SNAPPICK APP

Database

web Server

# Design Model

## Class Diagram



5.1 Class Diagram

**Description:**

The Class Diagram models the system's structure by representing classes, their attributes, methods, and relationships such as inheritance, composition, and aggregation. In this case, the User, Product, Order, and Cart classes are modeled.

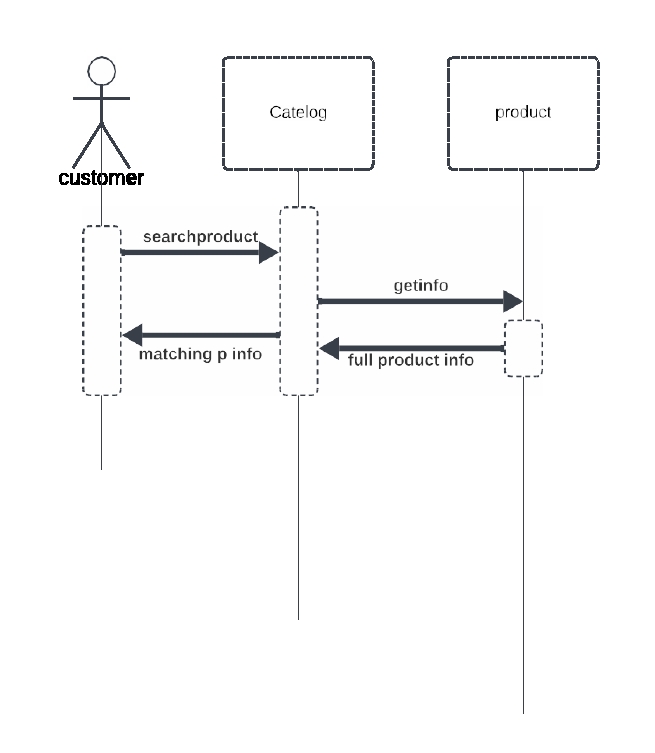
**Key Concepts**:

* Inheritance: The User class is a parent class for Customer and Admin, representing different roles in the system.
* Composition: An Order consists of multiple Products, so the Order class "owns" Product.
* Aggregation: A Cart contains multiple Products, but Products can exist independently of the cart.

**Classes and Example Methods**:

1. **User**
   * Attributes: userID, email, password
   * Methods: register(), login(), updateProfile()
2. **Product**
   * Attributes: productID, name, price, stock
   * Methods: addProduct(), updateStock()
3. **Order**
   * Attributes: orderID, userID, status
   * Methods: createOrder(), trackOrder()
4. **Cart**
   * Attributes: cartID, items[], totalPrice
   * Methods: addItem(), removeItem(), calculateTotal()

## 5.2 Sequence Diagram



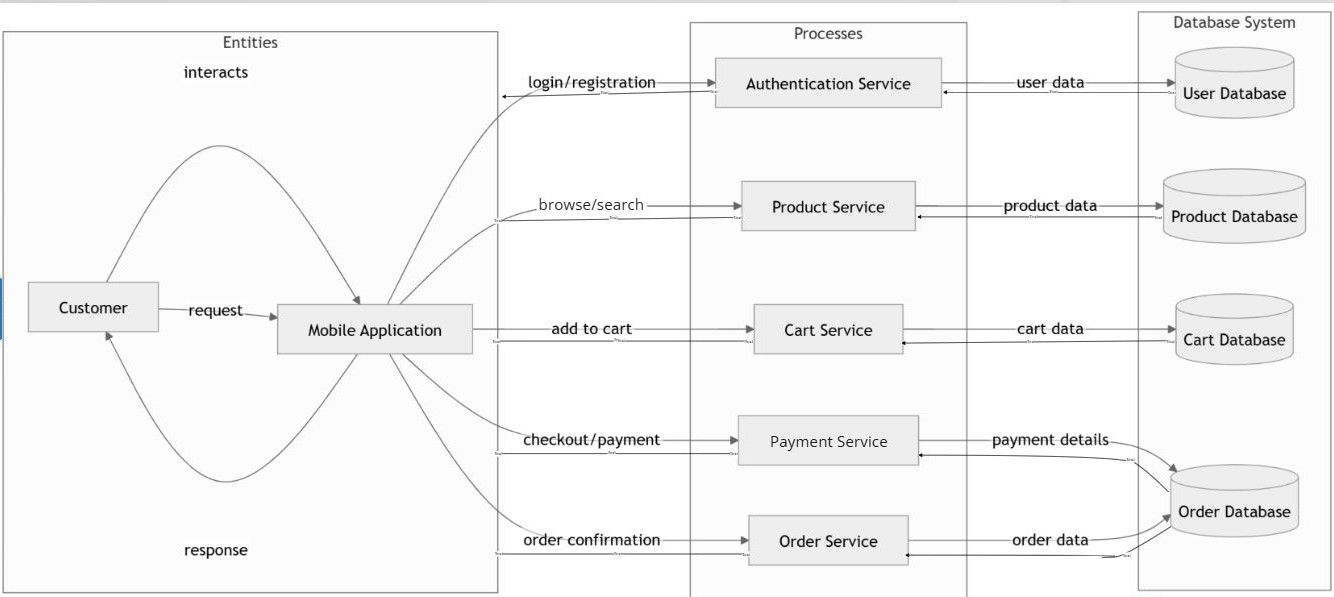
5.2 Sequence Diagram

**Description:** The Sequence Diagram illustrates the sequence of events between objects to perform a specific task, such as logging in, adding products to the cart, or placing an order.

**Example Process: User Login**

1. User enters credentials → System validates credentials → User is authenticated → Session is created → Login Confirmation sent to user.

## 5.3 Data Flow Diagram



5.3 Data Flow Diagram

**Description:** The Data Flow Diagram (DFD) represents how data flows through the system, showing processes, data stores, and data sources. It helps visualize the data movement in key processes like login, browsing products, and checkout.

**Level 0 DFD**:  
Shows major system processes and interactions:

1. User Login → User Data Store
2. Product Browsing → Product Database
3. Checkout → Order Creation Process
4. Order Confirmation → Order Data Store

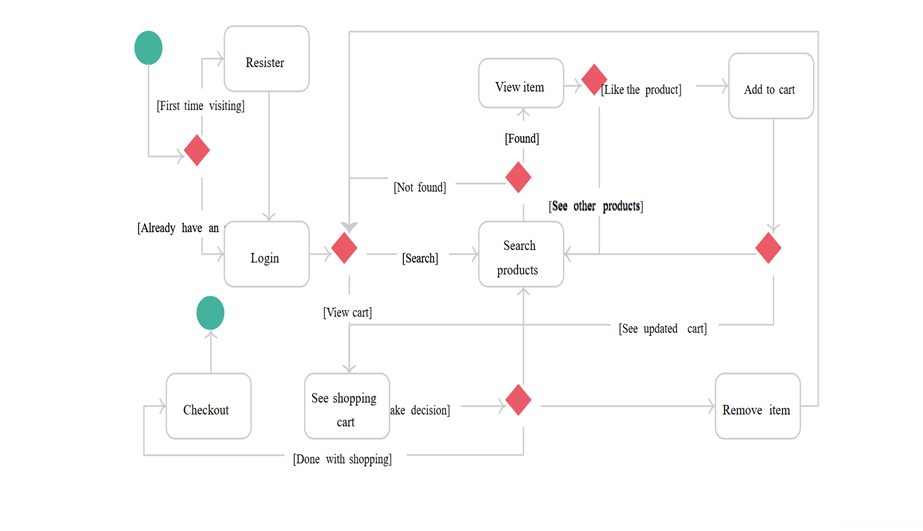
**Level 1 DFD**:  
Breaks down **Login Process** into sub-processes:

1. User Inputs Credentials → Authenticate Credentials → Create Session → Send Confirmation

**Level 1 DFD for Checkout**:

1. Select Products → Calculate Total → Process Payment → Create Order

## State Transition Diagram



5.4 State Transition Diagram

**Description:** The State Transition Diagram models the various states an Order goes through, such as Placed, Shipped, Delivered, and Completed. This helps manage order processing states based on actions.

**Order States**:

1. New Order → Payment Pending → Payment Processed → Shipped → Delivered → Completed
2. If cancelled, transition to Cancelled state.

# Data Design

## 6.1. Data Design

In an E-commerce Mobile Application, the information domain of the system is transformed into data structures in the following manner:

* + 1. **Entity Representation**:

The major entities such as User, Product, Order, Cart, etc., are modeled using classes or tables (depending on the design approach). Each entity is described with its attributes and how it interacts with other entities.

* + 1. **Data Storage**

These entities are stored in a relational database or NoSQL database. For the e-commerce system, relational databases (like MySQL, PostgreSQL) are typically used. For example:

* **User** data can be stored in a table called users, containing attributes like userID, name, email, etc.
* **Product** data is stored in a table called products, containing attributes like productID, name, price, etc.
* **Order** data is stored in an orders table with attributes like orderID, userID (foreign key), order date, status, etc.
  + 1. **Data Processing**

Data is processed through functions and operations like creating an order, updating the cart, calculating the total price, etc. For example:

* When the user places an order, an order record is created in the database.
* When an item is added to the cart, the cart is updated by adding the product to the cart items.
  + 1. **Data Organization**

Data is typically normalized to avoid redundancy. For example:

Users are stored in one table (users), and products are stored in another (products). The orders table links users and products through foreign keys, ensuring that the relationships between entities are maintained.

* + 1. **Databases or Data Storage Items**
  + **Relational Database**: For transactional consistency, relational databases are commonly used to store user, product, and order data.

## 6.2 Data Dictionary

A **Data Dictionary** is a collection of system entities (or data items) listed alphabetically along with their types, descriptions, and the attributes/methods of objects (if an object-oriented approach is followed).

**Entities & Attributes (for an E-commerce Mobile Application)**

1. **User**

* **Type**: Object/Database Table
* **Attributes**:
* userID: Unique identifier for each user (Primary Key).
* email: User's email address.
* password: Encrypted password for login.
* firstName: User's first name.
* lastName: User's last name.
* address: User's shipping address.
* phoneNumber: User's contact number.
* **Methods**:
* register(): Creates a new user.
* login(): Authenticates the user and starts a session.
* updateProfile(): Updates the user's profile information.

1. **Product**

* **Type**: Object/Database Table
* **Attributes**:
* productID: Unique identifier for each product.
* name: Name of the product.
* price: Price of the product.
* description: Description of the product.
* stockQuantity: Available stock for the product.
* **Methods**:
* addProduct(): Adds a new product to the catalog.
* updateStock(): Updates the stock quantity after a purchase.

1. **Order**

* **Type**: Object/Database Table
* **Attributes**:
* orderID: Unique identifier for each order.
* userID: The user who placed the order (Foreign Key).
* orderDate: Date and time the order was placed.
* status: Current status of the order (Pending, Shipped, Delivered, etc.).
* totalPrice: Total price of the order.
* **Methods**:
* createOrder(): Creates an order based on the user's cart.
* updateStatus(): Updates the status of the order (e.g., from "Pending" to "Shipped").

1. **Cart**

* **Type**: Object/Database Table
* **Attributes**:
* cartID: Unique identifier for each cart.
* userID: User associated with the cart (Foreign Key).
* items[]: List of products in the cart.
* totalPrice: Total price of the cart's contents.
* **Methods**:
* addItem(): Adds an item to the cart.
* removeItem(): Removes an item from the cart.
* calculateTotal(): Calculates the total price of the cart.

1. **Payment**

* **Type**: Object/Database Table
* **Attributes**:
* paymentID: Unique identifier for the payment.
* orderID: Order associated with the payment.
* paymentStatus: Status of the payment (Completed, Pending, Failed).
* paymentMethod: Method used for payment (Credit Card, PayPal, etc.).
* **Methods**:
* processPayment(): Processes the payment for an order.
* refundPayment(): Refunds the payment if necessary.

# 6.3 Functions & Parameters (for the E-commerce Mobile Application)

1. **register()**:

* **Parameters**: firstName, lastName, email, password, phoneNumber, address
* **Description**: Registers a new user by storing their information in the database.

1. **login()**:

* **Parameters**: email, password
* **Description**: Authenticates the user and initiates a session.

1. **addProduct()**:

* **Parameters**: productName, price, description, stockQuantity
* **Description**: Adds a new product to the catalog.

1. **createOrder()**:

* **Parameters**: userID, cartItems[], totalPrice
* **Description**: Creates a new order based on the items in the user's cart.

1. **processPayment()**:

* **Parameters**: orderID, paymentMethod
* **Description**: Processes the payment for a specific order.

# Algorithm & Implementation

In this section, we systematically analyze the key components of the E-Commerce Mobile Application by breaking down the major functions and their underlying algorithms. The algorithms are presented using Procedural Description Language (PDL) or pseudo-code to explain their logical flow. For each object-oriented design (OO), we summarize the member functions and attributes, providing an in-depth understanding of how the application operates.

## User Registration Algorithm

* **Function**: registerUser()
* **Purpose**: This function registers a new user by capturing their personal details, validating input, encrypting the password, and storing the data in the system's database

**Pseudo-code:**

*registerUser(firstName, lastName, email, password, phoneNumber, shippingAddress):*

*if isValidInput(firstName, lastName, email, password, phoneNumber, shippingAddress):*

*encryptedPassword = encrypt(password)*

*userID = generateUniqueID()*

*insertIntoDatabase('users', userID, firstName, lastName, email, encryptedPassword, phoneNumber, shippingAddress)*

*return "User Registered Successfully"*

*else:*

*return "Invalid Input. Please check the entered details."*

* **Local Data:**
  + encryptedPassword: The password is encrypted before being stored in the database for security purposes.
  + userID: A unique identifier generated for each user.

## User Login Algorithm

* **Function**: loginUser()
* **Purpose**: This function validates the user's credentials (email and password) against the database. If credentials match, it generates a session token and returns it for user authentication.

**Pseudo-code:**

*loginUser(email, password):*

*user = retrieveUserFromDatabase('users', email)*

*if user exists:*

*if validatePassword(password, user.encryptedPassword):*

*sessionID = createSession(user.userID)*

*return "Login Successful", sessionID*

*else:*

*return "Invalid Password. Please try again."*

*else:*

*return "User not found. Please register."*

* **Local Data**:
  + sessionID: A unique session token generated after a successful login.
  + user: Object containing the user's data retrieved from the database.

## 7.3 Product Browsing Algorithm

* **Function**: browseProducts()
* **Purpose**: This function allows the user to browse products. It supports searching by keywords and filtering based on categories and other attributes (e.g., price range).

**Pseudo-code:**

*browseProducts(searchQuery, filters):*

*if searchQuery or filters are provided:*

*results = searchDatabase('products', searchQuery, filters)*

*else:*

*results = getAllProductsFromDatabase('products')*

*return results*

**Local Data**:

* results: List of products matching the search query or filter criteria

## Add Product to Cart Algorithm

* Function: addProductToCart()
* Purpose: This function adds a selected product to the user's shopping cart. If the product is already in the cart, it updates the quantity.

**Pseudo-code:**

*addProductToCart(userID, productID):*

*cart = retrieveUserCart(userID)*

*if productID exists in cart:*

*updateCartItemQuantity(cart, productID, cart[productID].quantity + 1)*

*else:*

*addNewItemToCart(cart, productID, 1)*

*updateCartTotal(cart)*

*return "Product successfully added to cart"*

**Local Data**:

* cart: The user's shopping cart object, which stores all the items added to the cart.
* productID: A unique identifier for the product being added to the cart.

## Checkout Process Algorithm

* **Function**: checkout()
* **Purpose**: This function processes the user's cart, calculates the total price, and creates an order in the system.

**Pseudo-code:**

*checkout(userID):*

*cart = retrieveUserCart(userID)*

*if cart is not empty:*

*totalPrice = calculateTotal(cart)*

*orderID = createOrder(userID, cart, totalPrice)*

*clearCart(userID)*

*return "Order successfully created", orderID*

*else:*

*return "Your cart is empty. Please add products to the cart before proceeding."*

**Local Data:**

* totalPrice: The total cost of all items in the user's cart.
* orderID: The unique identifier of the order generated after the checkout process.

## Payment Process Algorithm

* **Function**: processPayment()
* **Purpose**: This function handles the payment process for the user’s order, updating the order status upon successful payment.

**Pseudo-code:**

*processPayment(orderID, paymentMethod):*

*order = retrieveOrderFromDatabase(orderID)*

*if order.status == "Pending":*

*paymentStatus = initiatePayment(paymentMethod, order.totalPrice)*

*if paymentStatus == "Success":*

*updateOrderStatus(orderID, "Paid")*

*return "Payment successful. Your order is now confirmed."*

*else:*

*return "Payment failed. Please try again."*

*else:*

*return "This order has already been paid."*

**Local Data**:

* paymentStatus: Represents the result of the payment attempt (e.g., "Success" or "Failure").
* order: The order object that holds the details of the order being process

## Object-Oriented Design Implementation

In an object-oriented design, the application’s major entities are represented as objects with attributes and methods that describe the behavior of these objects. Below is a summary of key objects, their attributes, and methods used in the application.

## 7.7.1 User Class

* **Attributes:**
* userID: Unique identifier for each user.
* firstName: User's first name.
* lastName: User's last name.
* email: Email address used for login.
* password: Encrypted password for secure login.
* phoneNumber: User’s contact number.
* shippingAddress: User's address for product delivery.
* **Methods**:
* register(): Registers the user by storing their details in the database.
* login(): Authenticates the user based on email and password.
* updateProfile(): Allows the user to update their personal information.
* changePassword(): Allows the user to change their password.

## 7.7.2 Product Class

* **Attributes:**
* productID: Unique identifier for the product.
* name: Name of the product.
* price: Price of the product.
* stockQuantity: Available stock of the product.
* category: Category of the product (e.g., Electronics, Clothing).
* description: Description providing more details about the product.
* **Methods:**
* addProduct(): Adds a new product to the product catalog.
* updateStock(): Updates the product stock when an item is sold.
* searchProduct(): Searches for products based on user-defined criteria (e.g., price range, category).

## 7.7.3Cart Class

* **Attributes:**
* cartID: Unique identifier for the user's shopping cart.
* userID: The user associated with the cart.
* items[]: List of products added to the cart.
* totalPrice: The total value of all items in the cart.
* **Methods:**
* addItem(): Adds a product to the cart.
* removeItem(): Removes a product from the cart.
* calculateTotal(): Calculates the total price of all items in the cart.

# REQUIREMENTS MATRIX

Requirements Traceability Matrix (RTM) establishes a clear mapping between the functional requirements, design components, and specific items in the SnapPick app's system design. It ensures that each requirement is traceable throughout the development process.

|  |  |  |  |
| --- | --- | --- | --- |
| **Req. Number** | **Ref. Item** | **Design Component** | **Component Items** |
| FR01 | Use Case Diagram | UC-0001 | User Registration and Login |
| FR02 | Class Diagram | User, Product | Attributes, Methods (e.g., Login()) |
| FR03 | ER Diagram | User Table | user\_id, email, password |
| FR04 | Data Flow Diagram | Level 1 | Process: Product Search |
| FR05 | Sequence Diagram | Checkout Process | Steps for payment and confirmation |
| FR06 | Static Diagram | Product Module | Interaction: Cart and Checkout |
| FR07 | Component Diagram | Backend API | CRUD operations for products |
| FR08 | Data Flow Diagram | Level 2 | Process: Location-Based Deals |
| FR09 | Architecture | Mobile App | Flutter frontend, PHP backend |

**FR01 - Use Case Diagram**

* Ref. Item: Captures user interactions for registration and login.
* Design Component: UC-0001 represents the primary use case for handling user authentication.
* Component Items: Processes include Sign Up, Login, and Password Recovery.

**FR02 - Class Diagram**

* Ref. Item:Illustrates the object-oriented structure of the system.
* Design Component: The User and Product classes define system entities and their behaviors.

**Component Items:**

* Attributes: username, password, email, price, etc.
* Methods: Functions like Login(), AddToCart().

**FR03 - ER Diagram**

* Ref. Item**:** Represents the database schema.
* Design Component: User Table structure, which includes:
* Primary Key:user\_id
* Attributes: email, password.

**FR04 - Data Flow Diagram (Level 1)**

* Ref. Item: Maps processes like Product Search and data movement.
* Design Component: High-level interactions between the app and backend, showing:
* Inputs (search terms) → Query database → Display results.

**FR05 - Sequence Diagram**

* Ref. Item: Describes the sequence of actions in the checkout process.
* Design Component:Includes steps for user interaction, backend processing, and payment confirmation.

**FR06 - Static Diagram**

* Ref. Item: Shows system modules like the Product Module.
* Design Component: Interaction between the Cart and Checkout processes ensures data flows correctly.

**FR07 - Component Diagram**

* Ref. Item:Depicts the backend system.
* Design Component:CRUD operations for managing Products, Users, and Orders.

**FR08 - Data Flow Diagram (Level 2)**

**Ref. Item:** Expands on location-based deals functionality.

**Design Component:**

* Input: User's postal code.
* Process: Fetch relevant deals.
* Output: Display deals in the app.

**FR09 - Architecture Diagram**

**Ref. Item:** High-level system architecture.

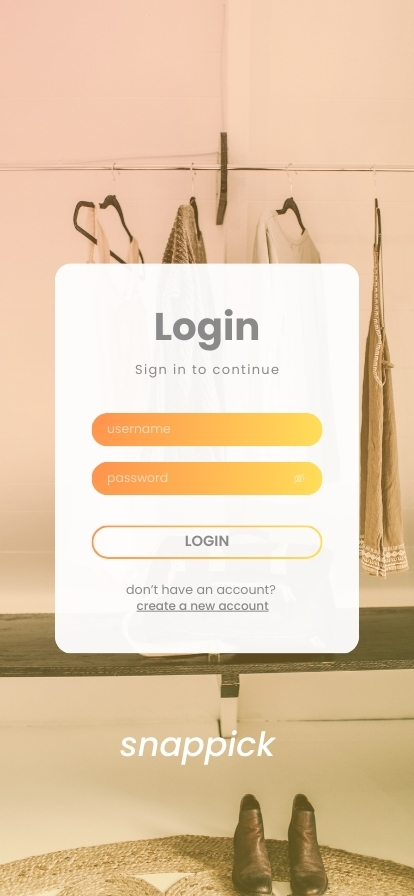
**Design Component:**

* Frontend: Developed using Flutter for cross-platform compatibility.
* Backend: Built with PHP to handle APIs and database operations.

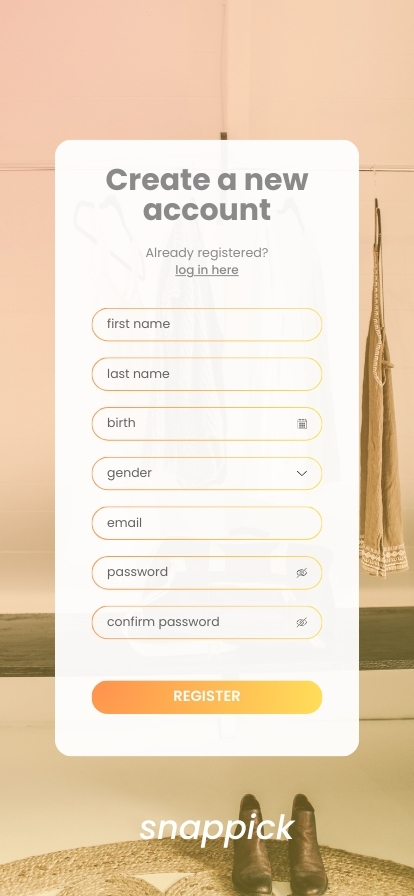
### 9. Human Interface Design

SnapPick is designed with user-centric functionality to ensure a seamless shopping experience. Users can perform the following key operations:

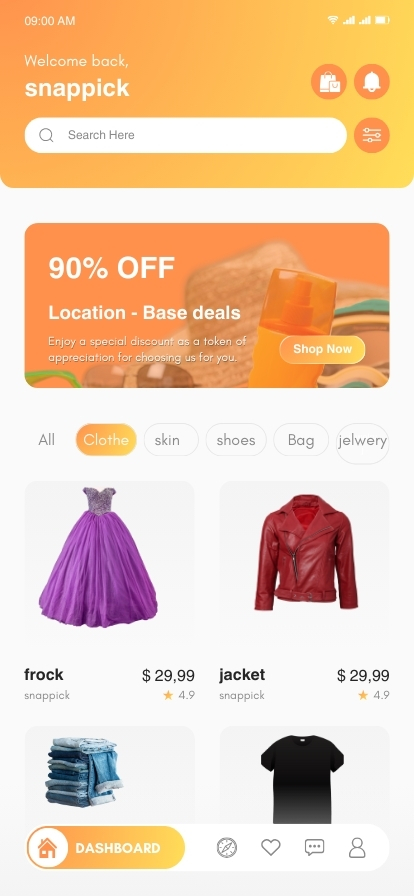
1. **User Registration and Login:**
   * **Registration:** Users can create an account by providing their personal details such as name, email, password, phone number, and shipping address. The system validates the input, encrypts passwords, and confirms successful registration.
   * **Login:** Existing users can log in using their email and password. Feedback is provided for incorrect login attempts.
2. **Browsing Products:**
   * Users can explore the product catalog with features such as keyword search, category filters, and sorting by price or popularity. Results are displayed dynamically.
3. **Adding Products to Cart:**
   * Users can add items to their cart with a single tap. The cart updates in real-time to reflect the quantity and total price of items added.
4. **Managing Cart:**
   * Users can view their cart, update quantities, remove items, or clear the cart entirely.
5. **Placing Orders:**
   * Users can proceed to checkout, confirm the shipping details, and place an order. A confirmation message and order details are displayed.
6. **Order Tracking:**
   * Users can track the status of their orders (e.g., "Pending," "Shipped") through the order history section.
7. **Feedback and Support:**
   * Users can provide feedback or seek assistance through a dedicated support section.
   1. Screen Images



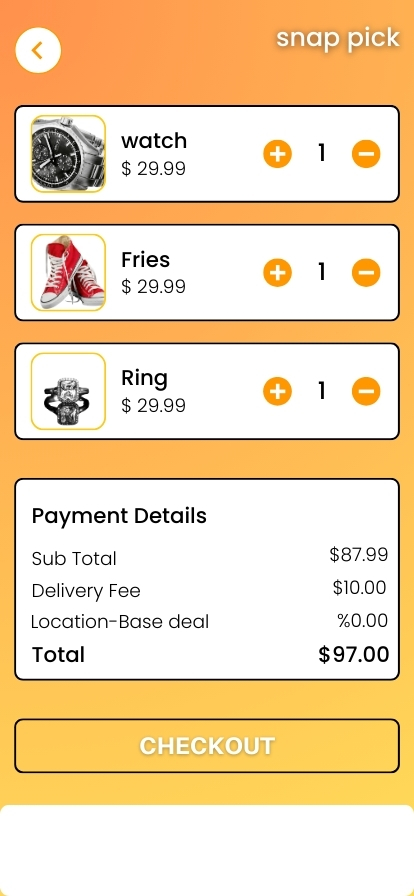
9.1.1 Login Screen



9.1.2 Login Screen



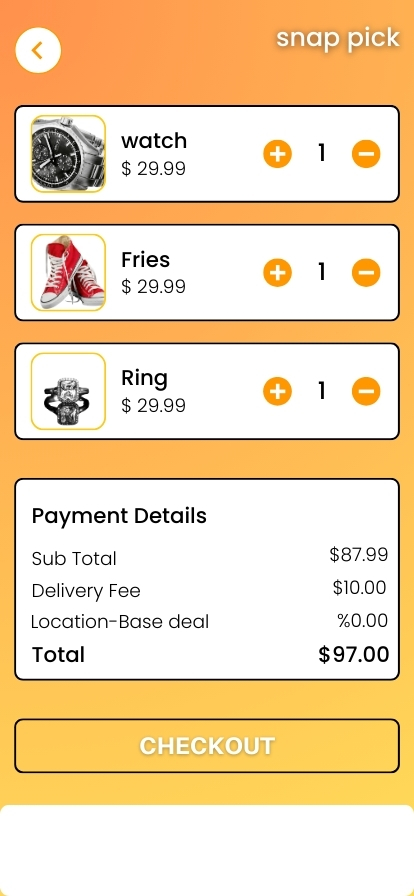
9.1.3 Home Page



9.1.4 Shopping Cart



9.1.5 Deal on Home Screen



9.1.6 Checkout Page

### 10. Appendix

### 10.1 Glossary

* **SnapPick**  
  The name of the e-commerce mobile application developed as part of the final year project. It serves as a platform for users to browse, shop, and manage their purchases efficiently.
* **Cart**  
  A virtual shopping basket where users can add, view, and manage the products they intend to purchase.
* **Order ID**  
  A unique identifier for tracking and managing individual orders placed by users.
* **Filters**  
  Search criteria used to refine product browsing, such as price range, category, brand, or other attributes.
* **COD (Cash on Delivery)**  
  A payment method that allows users to pay for their orders in cash upon delivery.
* **Product Browsing**  
  The process through which users explore and search for products, using keywords or filters to narrow down the options.
* **Checkout**  
  The process of finalizing an order, including reviewing the cart, confirming details, and initiating payment.
* **Payment Gateway**  
  An online service that processes payment transactions securely, facilitating payment through various methods like credit cards or e-wallets.
* **Encrypted Password**  
  A password that has been securely hashed or encrypted to protect user information from unauthorized access.
* **Database**  
  A structured repository where all user, product, order, and cart information is stored and managed.
* **Flutter**  
  An open-source framework by Google used to develop the frontend of the SnapPick mobile application.
* **PHP**  
  A server-side scripting language used for developing the backend of the application and handling server logic.
* **MySQL**  
  A relational database management system used for storing and managing application data such as users, products, orders, and carts.
* **Order Status**  
  The current state of an order (e.g., “Pending,” “Shipped,” “Delivered”) to inform users of its progress.