Software Design Document

CSE3112: Software Engineering Lab

App Name
Swapno - An Ecommerce App

Submitted By:

Name: Md Shamsur Rahman Sami

Roll No : 57

Name: Md Rakib Hossain

Roll No : 55

Submitted On :

April 16, 2024

Submitted To :

Dr. Saifuddin Md. Tareeq
Redwan Ahmed Rizvee

Contents

1 Introduction

1.1 Purpose

The purpose of this Software Design Document (SDD) is to outline the architecture and system design of the Swapno e-commerce application. The intended audience for this document includes developers, designers, project managers, and stakeholders involved in the development and maintenance of the Swapno app.

1.2 Scope

The Swapno e-commerce app aims to provide a platform for single-vendor online shopping. The goal is to offer users a seamless shopping experience through the Flutter-based user app and a Java-based admin app. The objectives of the project include creating intuitive user interfaces, efficient backend management systems, and secure payment gateways. The benefits of Swapno include convenience for users to shop online, streamlined management processes for the admin, and potential revenue generation through transactions.

1.3 Overview

This document provides an in-depth overview of the software design for the Swapno e-commerce application. It outlines the architecture, system components, user interactions, data flows, and other crucial aspects of the software design. The organization of the document follows a structured format to ensure clarity and ease of understanding for all stakeholders involved.

1.4 Reference Material

- Project Requirements Document
- User Interface Design Mockups
- Database Schema Documentation
- Flutter Documentation
- Java Documentation

1.5 Definitions and Acronyms

- SDD: Software Design Document
- Flutter: A UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase.
- Java: A high-level programming language developed by Sun Microsystems.
- Single Vendor: A type of e-commerce platform where products are sold by a single seller or vendor.
- UI: User Interface
- API: Application Programming Interface

2 System Description

2.1 Functionality

The Swapno e-commerce application provides a platform for users to browse, search, and purchase products from a single vendor. Users can register and log in to their accounts, view product listings, add items to their cart, and proceed with secure checkout processes. The admin app allows the vendor to manage products, orders, and user accounts efficiently.

2.2 Context

In today's digital age, online shopping has become increasingly popular due to its convenience and accessibility. The Swapno app caters to this growing trend by offering a user-friendly interface for customers to explore and buy products from a single vendor. With the rise of mobile applications, the Flutter-based user app ensures compatibility across various devices, providing a seamless shopping experience for users on both Android and iOS platforms. Meanwhile, the Java-based admin app empowers the vendor to oversee and manage their online store effectively.

2.3 Design

The design of the Swapno e-commerce application encompasses both front-end and back-end components. The front-end user interface is designed to

be intuitive and visually appealing, with features such as product categorization, search functionality, and interactive cart management. On the back end, the system utilizes databases to store product information, user data, and order details securely. APIs facilitate communication between the user and admin apps, enabling seamless data exchange and synchronization.

3 Design Overview

3.1 Design Rationale

The architecture for the Swapno e-commerce app was carefully selected to meet the project's requirements while considering critical issues and trade-offs. Several architectures were evaluated, ultimately leading to the decision to use Flutter for the user app and Java for the admin app.

The rationale behind this decision includes:

- **Flutter for User App**: Flutter was chosen for the user app due to its ability to develop cross-platform applications with a single codebase. It offers a rich set of UI components, enabling rapid development of intuitive user interfaces. Additionally, Flutter's hot reload feature facilitates quick iteration and experimentation during development.

- **Java for Admin App**: Java was selected for the admin app due to its widespread usage and robustness. Java offers mature libraries and frameworks for desktop application development, making it suitable for building the admin interface. Furthermore, Java's performance and reliability align well with the requirements of an admin application.

Other architectures were considered but not chosen for the following reasons:

- **Monolithic Architecture**: While monolithic architecture is simpler to develop initially, it may become complex and difficult to maintain as the application grows. Separating the user and admin functionalities into distinct applications allows for better scalability and maintainability.

- **Microservices Architecture**: Microservices introduce complexity in terms of deployment and managing inter-service communication. For a small-scale project like Swapno, the overhead of managing microservices outweighs the benefits.

- **Serverless Architecture**: Serverless architectures were not considered suitable for Swapno as they may not provide the required flexibility and control over backend services.

3.2 System Architecture

The system architecture of Swapno is designed around two primary components:

3.2.1 User App (Flutter)

The User App, developed using Flutter, serves as the primary interface for customers to interact with the e-commerce platform. It is responsible for:

- **User Interface Development**: Utilizing Flutter's extensive set of UI components to create an intuitive and visually appealing user interface.
- **User Interaction**: Facilitating user actions such as browsing products, adding items to the cart, placing orders, and managing their account settings.
- **Cross-Platform Compatibility**: Leveraging Flutter's cross-platform capabilities to ensure seamless operation on both Android and iOS devices with a single codebase.
- **Communication with Backend**: Interacting with the backend server through API calls to fetch product information, submit orders, and handle user authentication.

3.2.2 Admin App (Java)

The Admin App, developed using Java, serves as the administrative interface for managing various aspects of the e-commerce platform. It is responsible for:

- **Administrative Tasks**: Providing functionalities for administrators to manage products, orders, user accounts, and other administrative tasks essential for the operation of the platform.
- **Desktop Application Development**: Utilizing Java's capabilities for desktop application development to create a robust and feature-rich administrative interface.

- **Integration with Backend Services**: Communicating with the backend server to retrieve and update data related to products, orders, and user accounts, ensuring synchronization between the admin interface and the underlying database.
- **Security and Access Control**: Implementing authentication and authorization mechanisms to restrict access to sensitive administrative functionalities and ensure that only authorized users can perform administrative tasks.

These two components work in tandem to provide a comprehensive e-commerce solution, with the User App catering to the needs of customers and the Admin App facilitating efficient management and administration of the platform.

3.3 Constraint and Assumptions

3.3.1 List of Assumptions

- The device running the user app meets minimal hardware requirements for running Flutter applications.
- Reliable internet connectivity is necessary for uninterrupted usage of both user and admin apps.
- No unexpected backend difficulties causing substantial delays or failure.
- Users and administrators possess basic computer literacy for using the respective applications effectively.

3.3.2 List of Dependencies

User App Dependencies (Flutter):

- flutter
- flutter_material
- http (for HTTP requests)
- provider (for state management)
- firebase_auth (for user authentication)

- firebase_core (for Firebase integration)

Admin App Dependencies (Java):

- Java (for building Mobile applications)
- Java libraries for backend services

These dependencies are managed using package managers and build tools specific to Flutter and Java development environments.

4 Object Model

4.1 Object Description

4.1.1 Customer

Description: Represents a customer registered in the e-commerce system.

Attributes:

Attribute	Type	Description
buyerid	int	Unique identifier for the customer.
buyername	string	Name of the customer.
buyerphone	string	Phone number of the customer.
buyeremail	string	Email address of the customer.
buyeraddress	string	Address of the customer.

Methods :

Method	Description
registerbuyer()	Registers the customer in the system.
loginbuyer()	Allows the customer to log in to their account.
updatebuyer()	Updates the customer's information.

4.1.2 Payment

Description: Represents a payment transaction made by a customer.

Attributes:

Attribute	Type	Description
buyerid	int	Identifier of the customer making the payment.
cart	string	Details of the items in the shopping cart.
cardnumber	string	Card number used for payment.
totalprice	int	Total amount to be paid.

Methods :

Method	Description
transaction()	Initiates a payment transaction.
view()	Views payment details.

4.1.3 Product

Description: Represents a product available for purchase in the e-commerce system.

Attributes:

Attribute	Type	Description
productid	int	Unique identifier for the product.
productname	string	Name of the product.
productdescription	string	Description of the product.
productimage	string	URL of the product image.
productprice	int	Price of the product.

Methods :

Method	Description
addtocart()	Adds the product to the shopping cart.
updateproduct()	Updates the product information.

4.1.4 Cart

Description: Represents the shopping cart of a customer.

Attributes:

Attribute	Type	Description
buyerid	int	Identifier of the customer.
productid	int	Identifier of the product.
quantity	int	Quantity of the product in the cart.

Methods :

Method	Description
view()	Views the contents of the cart.
update()	Updates the cart (e.g., adding/removing items).
purchase()	Completes the purchase transaction.

4.1.5 Order

Description: Represents an order placed by a customer.

Attributes:

Attribute	Type	Description
orderid	int	Unique identifier for the order.
buyerid	int	Identifier of the customer placing the order.

Methods :

Method	Description
view()	Views the order details.
add()	Adds a new order.

4.1.6 Admin

Description: Represents an administrator of the e-commerce system.

Attributes:

Attribute	Type	Description
adminid	int	Unique identifier for the admin.
adminemail	string	Email address of the admin.

Methods :

Method	Description
view()	Views admin details.
update()	Updates admin information.
manage()	Manages system operations.
report()	Generates reports.

4.1.7 Category

Description: Represents a product category.

Attributes:

Attribute	Type	Description
categoryid	int	Unique identifier for the category.
categoryname	string	Name of the category.

Methods :

Method	Description
addcategory()	Adds a new category.
view()	Views category details.

4.2 Object Collaboration Diagram

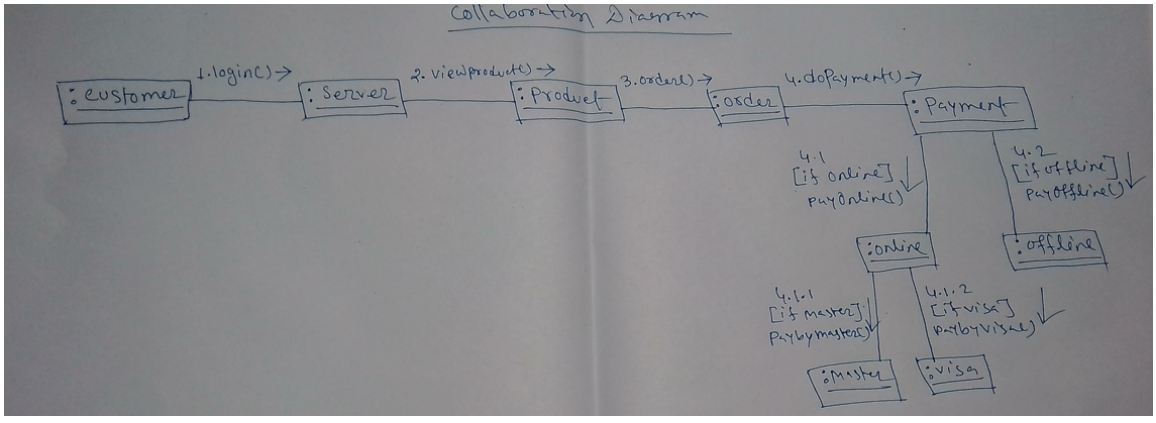


Figure 1: Object Collaboration Diagram

5 Subsystem Decomposition

5.1 Complete Package Diagram

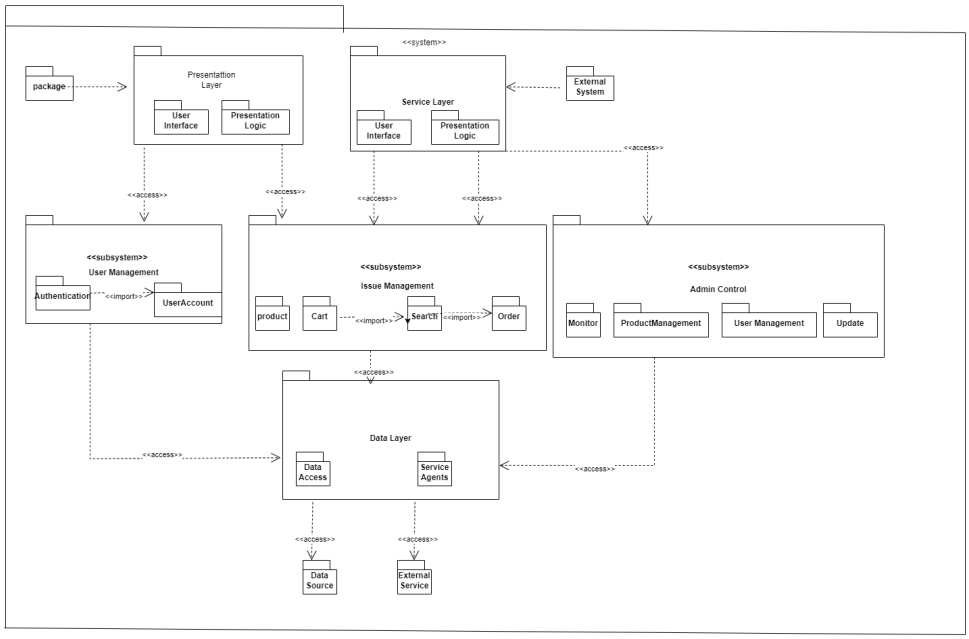


Figure 2: Package Diagram

5.2 Subsystem Detail Description

5.2.1 User Management

Module Description This module will handle user authentication, registration, and profile management. It will be responsible for user authorization and access control

5.2.2 Class Diagram

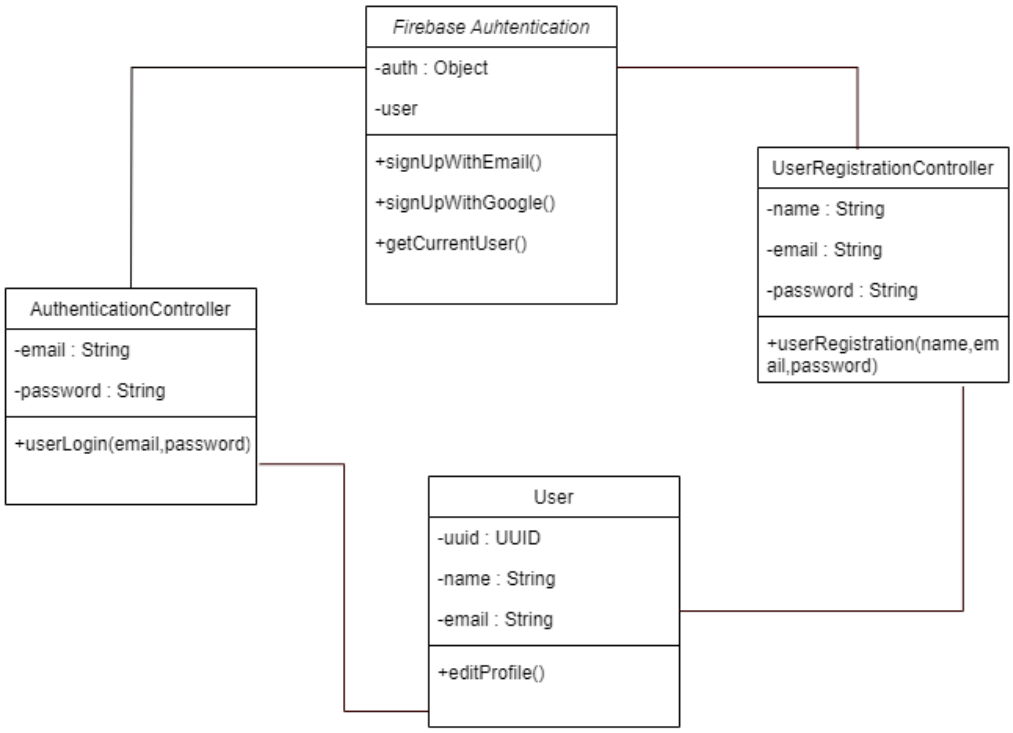


Figure 3: User Management Class Diagram

5.2.3 Subsystem Interface

From the user's perspective, the User Management module will provide a seamless experience for user authentication, registration, and profile management. Users will be able to create an account, log in, and access their profile information from a single interface. To register, users will need to provide their basic information, including their name, email address, and password. Once registered, users will be able to log in to the system using their email and password credentials. In case of incorrect login credentials, the system will display an error message to inform the user about the issue. Users will also be able to manage their profile information, including their contact information, profile picture, and other relevant details. They will be

able to update their profile information and save the changes in the system. Feedback information will be displayed to the user to confirm that their changes have been saved successfully. The User Management module will also handle user authorization and access control. Users with appropriate permissions will be able to access certain features of the system, while others will not. Feedback information will be displayed to inform users if they don't have sufficient permissions to access specific features. Overall, the User Management module will provide a secure and user-friendly interface for managing user authentication, registration, and profile information.

5.3 Issue Management

5.3.1 Module Description

This module will provide an interface for users to post their technologyrelated questions, issues, or problems. It will also allow users to search for previously posted questions and their answers. Other users and experts can collaborate to solve the issue by responding with potential solutions or asking for further clarification. Once the issue is resolved, the person who posted the question can mark it as resolved or choose the best answer .

5.3.2 Class Diagram

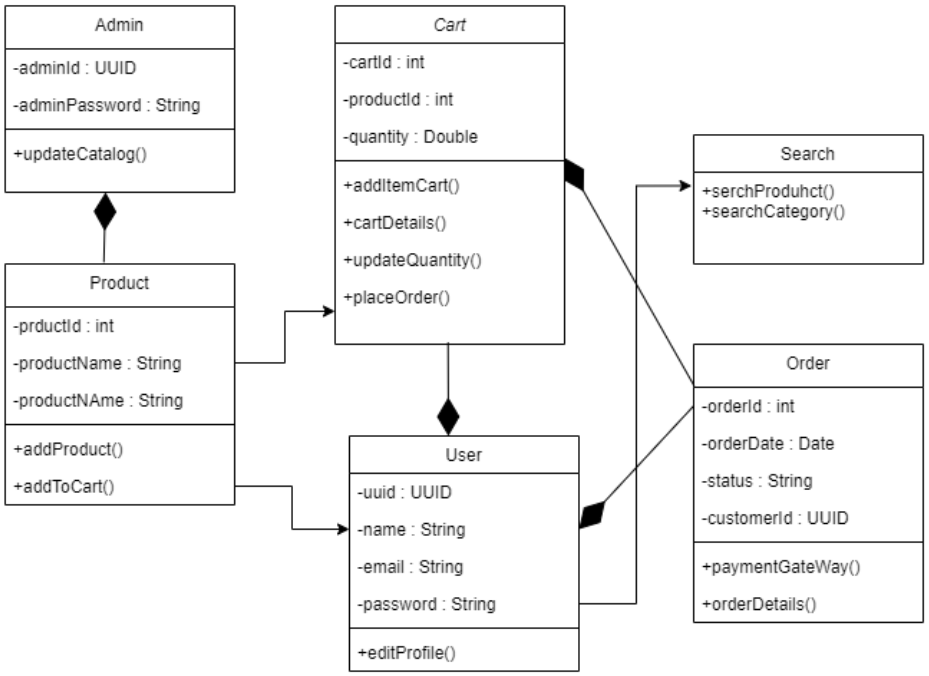
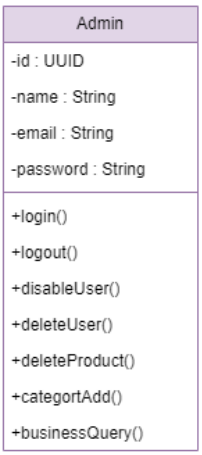


Figure 4: Issue Management Subsystem Class Diagram

5.4 Admin Control

5.4.1 Module Description The module will handle all admin operations. Verifying Service provider, monitoring overall system will be handled by this module. Only admin users have the access to this subsystem



6 Data Design

In the Swapno e-commerce app, the information domain is transformed into various data structures to efficiently store, process, and organize the major entities. These entities include customers, products, orders, payments, carts, admins, and categories. The data is typically stored in Firestore.

6.1 Data Description

- **Customers**: Representing users who interact with the application to browse products, add them to carts, and place orders.
- **Products**: Describing the items available for sale on the platform, including their names, descriptions, prices, and images.
- **Orders**: Recording details of customer purchases, including the products bought, quantities, total prices, and order statuses.
- **Payments**: Capturing payment transactions made by customers, including payment methods, transaction IDs, and amounts.
- **Carts**: Managing the items selected by customers for purchase before checkout, including product IDs and quantities.
- **Admins**: Representing the administrators who manage the e-commerce platform, including their IDs and authentication credentials.
- **Categories**: Categorizing products into different groups or types to facilitate browsing and searching for items.

The data structures used to represent these entities in Firebase are typically JSON objects, with each entity type having its own collection or node in the database. For example, the "customers" collection would contain JSON objects representing individual customer profiles, each with fields such as "name", "email", "address", etc.

Additionally, Firebase provides authentication services to secure user accounts and access to data, ensuring that only authenticated users can perform certain actions like placing orders or updating their profiles.

6.2 Data Dictionary

Entity	Type	Description
Customer	*Entity*	*Storesinformationaboutcustomers.*
Product	*Entity*	*Containsdetailsaboutproductsavailableforsale.*
Order	*Entity*	*Storesinformationaboutordersplacedbycustomers.*
Payment	*Entity*	*Recordspaymenttransactionsmadebycustomers.*
Cart	*Entity*	*Managesitemsaddedtoshoppingcarts.*
Admin	*Entity*	*Storesinformationaboutsystemadministrators.*
Category	*Entity*	*Managesproductcategories.*
description	*String*	*Anoptionaldescriptionprovidedbytheuser.*
email	*String*	*Theemailaddressoftheuser.*
id	*String*	*Auniqueidentifierforthepostoruser.*
name	*String*	*Thenameoftheuser.*
uid	*String*	*Auniqueidentifierfortheuser*
user	*UserorFirebaseUser*	*Theuserobjectrepresentingtheuserwhomadethepostoriscurrent*
searches	*List*	*Alistofsearchqueriesmadebytheuser*

6.3 Entity Relationship Diagram

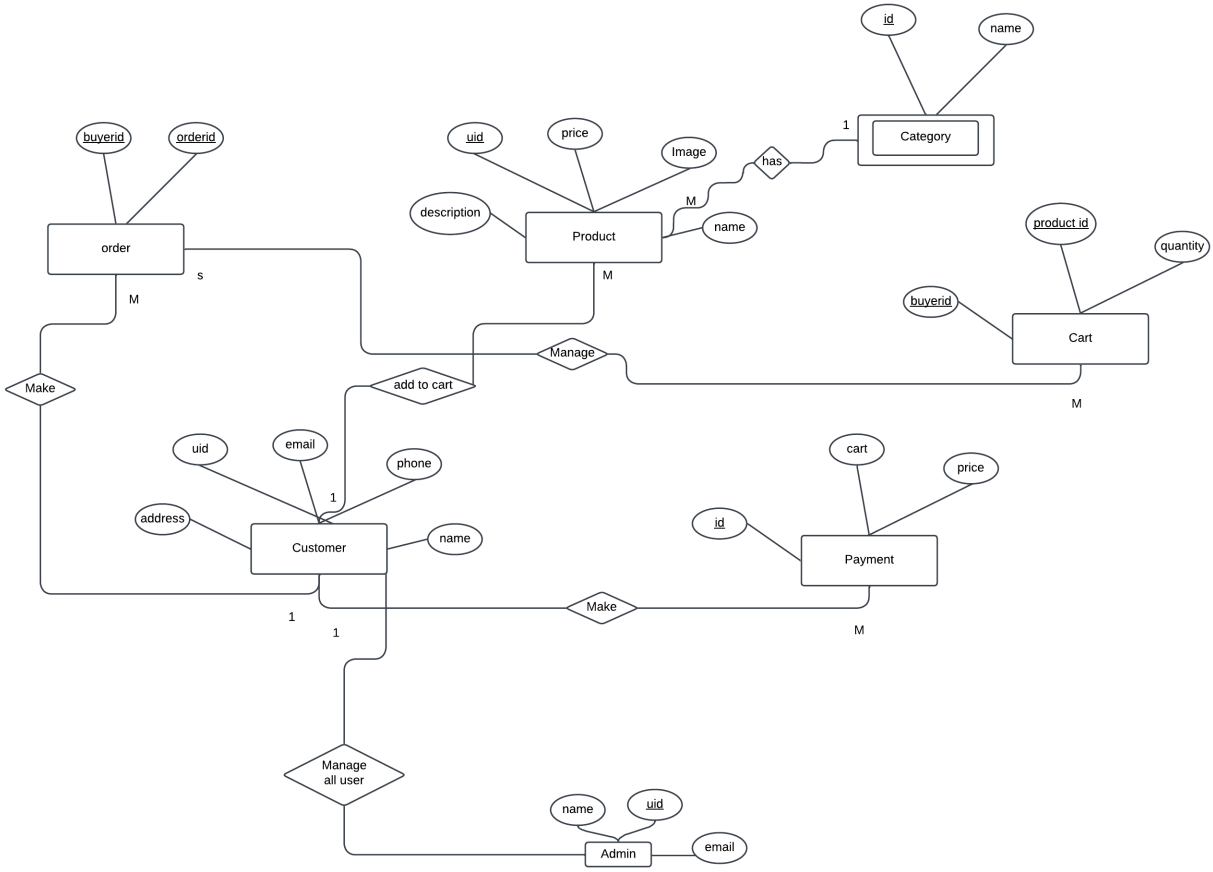


Figure 6: Entity Relationship Diagram

This diagram visually represents the relationships between the entities present in the system.

7 User Requirement and Component Traceability Matrix

User Requirements (RAD)	User Management	Issue Management	Admin Control
1. User Visits as Guest	x	x	-
2. User Registration and Authentication	x	-	-
3. User Post Functionality x	-	x	-
4. Commenting on Posts	-	x	-
5. Upvoting and Downvoting Solutions	-	x	-
6. Live Chat Support	-	x	-
7. Supportability	-	-	x

Table 1: Component Traceability Matrix

8 Supporting Information

As of this moment, we do not have any supporting information that needs to be documented. If we face such an information, we will update it with the necessary information.

.