



**RAJALAKSHMI  
ENGINEERING COLLEGE**

An AUTONOMOUS Institution  
Affiliated to ANNA UNIVERSITY, Chennai

## **E-COMMERCE PRODUCT CATALOG**

Submitted by:

**PRASANNA S (221801038)**

**SUKISH M (221801053)**

**IT19541 WEB TECHNOLOGY**

**Department of Artificial Intelligence and Data Science**

**Rajalakshmi Engineering College, Thandalam**

## **BONAFIDE CERTIFICATE**

Certified that this project report “**E-COMMERCE PRODUCT CATALOG**” is the bonafide work of “**PRASANNA S (221801038),      SUKISH M (221801053),**” who carried out the project work under my supervision.

### **SIGNATURE**

**Mrs.Renuga Devi S**  
Associate Professor, Artificial  
Intelligence and Data Science,  
Rajalakshmi Engineering College  
(Autonomous), Thandalam,  
Chennai-602105

### **SIGNATURE**

**Dr. GNANASEKAR J M**  
Head of the Department,  
Artificial intelligence and data  
Science,Rajalakshmi Engineering  
College (Autonomous),Chennai-  
602105

**Submitted for the Practical Examination held on \_\_\_\_\_**

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## **TABLE OF CONTENT**

<b>S.NO</b>	<b>CHAPTER</b>	<b>PAGE NUMBER</b>
1.	<b>INTRODUCTION</b>	
1.1	OVERVIEW	1
1.2	OBJECTIVES	2
1.3	MODULES	3
2.	<b>SURVEY OF TECHNOLOGIES</b>	
2.1	SOFTWARE DESCRIPTION	7
2.2	PROGRAMMING LANGUAGES	9
2.2.1	SQLite	10
2.2.2	PYTHON	15
3.	<b>REUIREMENT AND ANALYSIS</b>	
3.1	REUIREMENT SPECIFICATION	20
3.1.1	HARDWARE	25
3.1.2	SOFTWARE	26
4.	<b>ARCHITECTURE DIAGRAM</b>	30
5.	<b>MODULES</b>	35
6.	<b>SOFTWARE DEVELOPMENT MODEL</b>	36

7.	<b>TESTING SOFTWARE</b>	40
8.	<b>SOURCE CODE</b>	43
9.	<b>RESULTS AND DISCUSSIONS</b>	
9.1	DATABASE DESIGN	53
10.	<b>CONCLUSION</b>	56
11.	<b>REFERENCES</b>	58

## **TABLE OF FIGURES**

<b>S.NO</b>	<b>FIGURE</b>	<b>PAGE NUMBER</b>
1	REQUIREMENTS	22
2	ARCHITECTURE DIAGRAM	30
3	DATAFLOW DIAGRAM	31
4	ENTITY RELATIONSHIP DIAGRAM	32
5	ENTITY RELATIONSHIP MAPPING RULES	33
6	UML CLASS DIAGRAM	33
7	USE CASE DIAGRAM	34
8	TESTING OUTPUT	42

## **ABSTRACT**

The E-Commerce Product Catalog project aims to deliver an efficient, user-friendly online shopping platform, enhancing both the customer experience and operational efficiency for businesses. Key features include an intuitive interface for browsing, product selection, and secure purchasing. For businesses, the system offers robust product and order management through a centralized platform, featuring dynamic inventory control, predictive demand analytics, and flexible pricing tools. Secure payment integration and personalized product recommendations further improve user trust and engagement.

The proposed system addresses limitations in current solutions, such as complex management and limited analytics, by offering simplified dashboards and AI-driven insights. Additionally, the system leverages cloud-based solutions for scalability, making it adaptable for businesses of various sizes. An Agile development model supports iterative enhancements and timely responses to user feedback. Built with Python, Tkinter, and SQLite, this versatile platform is poised to meet evolving e-commerce demands, ensuring sustainability and growth in a competitive digital marketplace.

# **1. INTRODUCTION**

## **1.1 OVERVIEW:**

The e-commerce management system revolutionizes the way customers engage with products and services, enabling them to purchase goods online directly from merchants who operate on the internet. Since the advent of the World Wide Web, businesses have sought to expand their reach by offering their products to internet users. Customers can now visit e-commerce websites from the comfort of their homes and shop as they sit in front of their computers or use their mobile devices. They can browse a wide variety of products, compare prices, and make informed purchasing decisions.

The convenience of online shopping is a primary reason why many people prefer it over traditional in-store shopping. For instance, when a person shops online, they avoid the hassle of driving to a store, finding a parking spot, and navigating through crowded aisles. The e-commerce system offers a seamless shopping experience where consumers can find and purchase items with just a few clicks.

With the continuous advancements in technology, there have been significant improvements in e-commerce systems. Users can now enjoy features like detailed product descriptions, high-resolution images, customer reviews, and personalized recommendations. The integration of secure online payment gateways ensures that transactions are safe and efficient. Additionally, users can track their orders in real-time, from the moment of purchase to delivery.

Digitalization has been one of the most transformative outcomes of modern technology, providing easy access to a plethora of resources. The convenience of e-commerce eliminates the need to stand in long checkout lines, making shopping a more pleasant experience. Moreover, the ability to read reviews and ratings online helps users make better-informed decisions about their purchases. Online payments have also bolstered the use of e-banking,

contributing to the robust development of the global economy. The e-commerce management system enables customers to purchase products directly over the internet and pay using online banking methods. This system is measurable, cost-effective, and offers an excellent user interface.

This web-based PHP & MySQL project encompasses all the functionalities of a comprehensive e-commerce management system. It is beneficial for both merchants and customers alike. Customers can make purchases at any time of the day, 24/7, and since it is a web-based application, they can shop from anywhere in the world. This system eliminates the time wasted in traditional shopping, providing a more efficient shopping experience.

The system includes a user interface specification that outlines the standards to be used in designing the system, ensuring a consistent and user-friendly experience. It also addresses non-functional requirements and considerations for system evolution, ensuring the e-commerce platform remains scalable and adaptable to future needs.

In conclusion, the e-commerce management system offers numerous advantages for both sellers and buyers. It provides a convenient, efficient, and secure shopping environment, supporting the dynamic needs of the modern consumer. This digital transformation in retail not only enhances the shopping experience but also contributes significantly to the economic growth by promoting e-banking and online transactions.

## **1.2 OBJECTIVES:**

The objective of the E-Commerce Management System project is to design and develop a robust, scalable, and user-friendly web-based platform that facilitates seamless online shopping experiences for customers and efficient sales management for merchants. This system aims to leverage the latest advancements in web technologies to offer a comprehensive suite of features that enhance the convenience, security, and efficiency of online transactions. Specifically, the project seeks to achieve the following goals:



1. **Enhance User Convenience:** Provide customers with the ability to browse, select, and purchase a wide range of products from the comfort of their homes, eliminating the need for physical store visits and reducing time spent on shopping.
2. **Streamline Merchant Operations:** Enable merchants to manage their online stores effectively through a centralized platform that supports product listings, inventory management, order processing, and customer relationship management.
3. **Ensure Secure Transactions:** Integrate secure payment gateways to safeguard customer information and financial transactions, fostering trust and encouraging the adoption of e-commerce practices.
4. **Improve User Experience:** Develop an intuitive and engaging user interface that simplifies navigation, enhances product discovery, and provides a personalized shopping experience based on user preferences and behavior.
5. **Support 24/7 Accessibility:** Ensure the platform is accessible at all times, allowing customers to shop and merchants to manage their stores around the clock, regardless of their geographic location.
6. **Incorporate Advanced Features:** Include functionalities such as detailed product descriptions, high-resolution images, customer reviews, personalized recommendations, order tracking, and online customer support to enrich the overall shopping experience.
7. **Facilitate Economic Growth:** Promote the use of e-banking and online transactions, contributing to the broader adoption of digital payment methods and supporting the development of the global economy.
8. **Ensure Scalability and Evolution:** Design the system to be scalable and adaptable, accommodating future growth and technological advancements, and continuously evolving to meet the changing needs of the market.

### **1.3 MODULES:**

To ensure a comprehensive and efficient e-commerce management system, the project will be divided into several interconnected modules, each focusing on a specific aspect of the platform. Below is a list of the key modules along with their descriptions:

## 1. User Management Module

- **User Registration and Authentication:** Handles user sign-up, login, password recovery, and authentication processes.
- **User Profile Management:** Allows users to view and update their personal information, addresses, and payment methods.
- **Role-Based Access Control:** Defines different user roles (e.g., customers, merchants, admins) and manages permissions accordingly.

## 2. Product Management Module

- **Product Catalog:** Facilitates the addition, modification, and deletion of product listings by merchants.
- **Product Categorization:** Organizes products into categories and subcategories for easier navigation and searchability.
- **Inventory Management:** Monitors stock levels, updates inventory in real-time, and notifies merchants of low stock.

## 3. Order Management Module

- **Order Processing:** Manages the entire order lifecycle from placement to fulfillment, including order confirmation and status tracking.
- **Payment Processing:** Integrates with secure payment gateways to process online transactions and handle various payment methods.
- **Shipping and Delivery:** Coordinates with shipping providers, generates shipping labels, and tracks delivery status.

## 4. Shopping Cart and Checkout Module

- **Shopping Cart:** Allows customers to add, update, and remove products from their cart.
- **Checkout Process:** Streamlines the checkout process, including billing and shipping information, order summary, and payment options.
- **Order Confirmation:** Sends confirmation emails and receipts to customers upon successful order placement.

## 5. Search and Navigation Module

- **Product Search:** Implements search functionality with filters (e.g., price, category, rating) to help users find products quickly.
- **Navigation Menus:** Provides easy-to-use menus and breadcrumbs for efficient site navigation.

- **Product Recommendations:** Offers personalized product recommendations based on user behavior and preferences.

## **6. Review and Rating Module**

- **Customer Reviews:** Allows customers to write and submit reviews and rate products.
- **Review Moderation:** Enables admins to approve, reject, or flag inappropriate reviews.
- **Review Display:** Displays reviews and ratings on product pages to help users make informed decisions.

## **7. Admin Dashboard Module**

- **Analytics and Reporting:** Provides detailed reports and analytics on sales, user activity, and product performance.
- **Content Management:** Manages website content, banners, and promotional materials.
- **User and Order Management:** Allows admins to view and manage user accounts and orders.

## **8. Marketing and Promotions Module**

- **Discounts and Coupons:** Enables merchants to create and manage discount codes and promotional offers.
- **Email Marketing:** Integrates with email marketing tools to send promotional emails, newsletters, and updates.
- **Loyalty Programs:** Implements loyalty programs and reward points to encourage repeat purchases.

## **9. Customer Support Module**

- **Help Desk:** Provides a ticketing system for customer inquiries and support requests.
- **Live Chat:** Integrates live chat functionality for real-time customer support.
- **FAQ and Knowledge Base:** Offers a repository of frequently asked questions and help articles.

## **10. Security and Compliance Module**

- **Data Encryption:** Ensures that sensitive data is encrypted during transmission and storage.
- **User Privacy:** Implements privacy policies and compliance with data protection regulations (e.g., GDPR).

- **Fraud Detection:** Monitors transactions for fraudulent activity and implements measures to prevent fraud.

## **2. SURVEY OF TECHNOLOGIES**

### **2.1 SOFTWARE DESCRIPTION:**

#### **Overview**

The E-Commerce Management System is a comprehensive web-based platform designed to facilitate online shopping and sales management. It provides a robust framework for merchants to list and sell their products and for customers to browse, select, and purchase items conveniently. The system leverages PHP and MySQL to deliver a scalable, secure, and user-friendly experience, integrating various advanced features to enhance both user and merchant interactions.

#### **System Architecture**

The system architecture follows a multi-tier design, ensuring separation of concerns and promoting maintainability. The main components include:

1. **Presentation Layer:** The user interface, developed using HTML, CSS, JavaScript, and frameworks like Bootstrap, ensures an intuitive and responsive design, accessible across various devices and screen sizes.
2. **Application Layer:** Implemented using PHP, this layer handles business logic, processes user requests, manages sessions, and enforces security protocols.
3. **Data Layer:** MySQL serves as the database management system, storing and retrieving data related to users, products, orders, and other entities. The database schema is designed to optimize performance and ensure data integrity.

#### **Key Features**

1. **User Management:**
  - Secure user registration and login.
  - Role-based access control for customers, merchants, and administrators.

- Profile management with options to update personal details and payment methods.
- 2. Product Management:**
- Easy product catalog management with options to add, edit, and delete products.
  - Inventory tracking to monitor stock levels and manage restocking.
  - Product categorization for enhanced searchability and organization.
- 3. Order Management:**
- Comprehensive order processing, from cart to checkout to delivery.
  - Integration with payment gateways for secure transactions.
  - Real-time order tracking and status updates for customers.
- 4. Shopping Cart and Checkout:**
- Persistent shopping cart functionality allowing users to add, remove, and update items.
  - Streamlined checkout process with billing, shipping, and payment options.
  - Order confirmation and receipt generation.
- 5. Search and Navigation:**
- Advanced search features with filters and sorting options.
  - User-friendly navigation menus and breadcrumb trails.
  - Personalized product recommendations based on user behavior and preferences.
- 6. Reviews and Ratings:**
- Customer review and rating system to provide feedback on products.
  - Review moderation tools for administrators to manage user-generated content.
  - Display of reviews and ratings on product pages.
- 7. Admin Dashboard:**
- Analytics and reporting tools for sales, user activity, and product performance.
  - Content management system for managing site content and promotions.
  - Tools for managing user accounts and orders.
- 8. Marketing and Promotions:**
- Tools for creating and managing discount codes, coupons, and promotional offers.
  - Integration with email marketing services for sending newsletters and promotions.
  - Loyalty programs to incentivize repeat purchases and customer retention.
- 9. Customer Support:**

- Help desk system with ticketing for customer inquiries and support issues.
- Live chat integration for real-time assistance.
- FAQ and knowledge base for self-service support.

#### 10. Security and Compliance:

- Implementation of data encryption to protect sensitive information.
- Compliance with data protection regulations such as GDPR.
- Fraud detection mechanisms to monitor and prevent fraudulent transactions.

### Technical Specifications

- **Backend:** PHP
- **Database:** MySQL
- **Frontend:** HTML, CSS, JavaScript, Bootstrap
- **Payment Gateways:** Integration with popular payment processors (e.g., PayPal, Stripe)
- **Hosting:** Compatible with various web hosting services supporting PHP and MySQL
- **APIs:** Integration with third-party services for email marketing, shipping, and analytics

### Benefits

- **Convenience:** Enables customers to shop anytime, anywhere, reducing the need for physical store visits.
- **Efficiency:** Streamlines operations for merchants, improving inventory management and sales processes.
- **Security:** Ensures secure transactions and data protection, fostering customer trust.
- **Scalability:** Designed to handle increasing traffic and transaction volumes, supporting business growth.
- **User Experience:** Provides an engaging and intuitive interface, enhancing user satisfaction and loyalty.

## 2.2 LANGUAGES:

The E-Commerce Management System project employs Python for backend development and SQLite for database management. This strategic combination leverages Python's simplicity, versatility, and extensive library support, making it an excellent choice for developing robust backend systems. Python's readability and ease of use accelerate

development time and facilitate maintenance, while frameworks like Django or Flask enhance the development process with built-in features for handling authentication, routing, and more.

On the other hand, SQLite is chosen for its robust and efficient data handling capabilities. Known for its high performance, reliability, and ease of use, SQLite is well-suited for managing the structured data typical in e-commerce applications. Its support for complex queries, transactions, and ACID compliance ensures data integrity and consistency, which are critical for handling financial transactions and sensitive user information.

By integrating Python and SQLite, the E-Commerce Management System achieves a powerful synergy that results in a scalable, secure, and efficient platform. Python's object-relational mapping (ORM) capabilities, such as those provided by Django's ORM or SQLAlchemy, streamline interactions with the MySQL database, making data operations seamless and efficient. This combination not only supports rapid development and scalability but also ensures robust security and efficient data management, providing a comprehensive solution for modern e-commerce needs.

Front End:-	Python
Back End:-	SQLite

### 2.2.1 **SQLite:**

SQLite is a lightweight, serverless, self-contained, and open-source relational database management system (RDBMS) that is embedded into the applications that use it. It is designed for simplicity, efficiency, and ease of use, making it suitable for small to medium-sized applications, mobile apps, and embedded systems.

Here are some key characteristics of SQLite:



1. **Embedded Database:** SQLite operates as an embedded database engine, meaning it runs within the same process as the application that utilizes it. There is no need for a separate database server, and the entire database is stored in a single disk file, simplifying deployment and management.
2. **Serverless:** Unlike traditional client-server databases, SQLite does not operate as a separate server process. Instead, it directly accesses the database file, making it easy to set up and use without the need for server configuration or administration.
3. **Self-Contained:** SQLite databases are self-contained, meaning they do not rely on external dependencies or configuration files. Everything needed to access and manage the database is contained within the SQLite library, making it highly portable and suitable for deployment on various platforms and operating systems.
4. **Relational Database:** SQLite follows the relational database model and supports SQL (Structured Query Language) for querying and manipulating data. It offers features such as transactions, indexes, views, triggers, and more, providing a robust and flexible database solution.
5. **ACID Compliance:** SQLite ensures ACID (Atomicity, Consistency, Isolation, Durability) compliance, which guarantees that transactions are processed reliably and consistently, even in the event of system failures or interruptions.
6. **Cross-Platform Compatibility:** SQLite databases are cross-platform and can be used on various operating systems, including Windows, macOS, Linux, iOS, and Android. This makes SQLite a versatile choice for applications that need to run on multiple platforms.
7. **Low Resource Consumption:** SQLite is known for its low memory and disk space requirements, making it suitable for resource-constrained environments such as mobile devices and embedded systems. Despite its small footprint, SQLite offers efficient performance and scalability for many applications.

### **SQLite SERVES AS AN ASSET:**

SQLite serves as a valuable asset in the E-Commerce Management System project, particularly in scenarios where a lightweight, embedded database solution is advantageous. Here's how SQLite proves its utility:

#### **1. Development and Testing Efficiency:**

- **Quick Setup:** SQLite requires minimal configuration, facilitating rapid deployment during development and testing phases.
- **Embedded Database:** Its self-contained nature allows developers to work without the need for external database servers, streamlining the development process.

## **2. Prototyping and MVP Development:**

- **Rapid Prototyping:** SQLite enables swift iteration and experimentation, ideal for building prototypes or Minimum Viable Products (MVPs) in the early stages of the project.
- **Cost-Effective Solution:** For startups or projects with budget constraints, SQLite's open-source nature eliminates licensing fees, making it a cost-effective choice.

## **3. Local Development Environment:**

- **Local Testing:** Developers can simulate database interactions locally without requiring network connectivity, enhancing efficiency and autonomy.
- **Cross-Platform Compatibility:** SQLite databases are platform-independent, facilitating seamless collaboration across diverse development environments.

## **4. Data Integrity and Reliability:**

- **ACID Compliance:** SQLite ensures Atomicity, Consistency, Isolation, and Durability, maintaining data integrity and reliability during transactions.
- **Transactional Support:** Developers can implement complex transactional operations with confidence, essential for managing e-commerce transactions securely.

## **5. Scalability for Small to Medium Applications:**

- **Suitable for Moderate Workloads:** While not designed for handling high concurrent loads, SQLite is proficient for small to medium-scale applications, providing adequate performance.
- **Flexibility in Deployment:** SQLite's lightweight footprint allows for easy deployment in environments where scalability demands are modest.

## **6. Seamless Integration with Python:**

- **Native Support in Python:** SQLite is integrated into the Python ecosystem, allowing developers to interact with databases seamlessly using the sqlite3 module.
- **ORM Compatibility:** Popular Python frameworks like Django and Flask support SQLite as a backend database, enabling Object-Relational Mapping (ORM) functionality for data management.

## 7. Backup and Maintenance:

- **Single-File Storage:** The entire database resides in a single file, simplifying backup and restoration processes, essential for data protection and maintenance.

## 8. Localized Deployments and Small-Scale Deployments:

- **Embedded Deployments:** SQLite's embedded nature is well-suited for applications requiring a self-contained database solution, such as point-of-sale systems or single-user applications.

## 9. Cross-Platform Support:

- **Versatile Compatibility:** SQLite databases are cross-platform, allowing seamless deployment across various operating systems and environments, enhancing flexibility.

## SQLite FOR BACKEND:

The E-Commerce Management System project, SQLite can be instrumental in supporting backend operations in several ways:



### 1. Data Storage and Management:

- **Product Catalog:** SQLite can store product information such as product names, descriptions, prices, and inventory levels. This data can be efficiently managed and queried to provide users with up-to-date product listings.
- **User Profiles:** SQLite can store user information, including user credentials, shipping addresses, and payment details. This allows for secure and reliable user authentication and personalized shopping experiences.

- **Order Management:** SQLite can manage order details such as order IDs, product quantities, prices, and shipping information. This enables the backend to process orders, track order statuses, and manage inventory levels accordingly.
2. **Transaction Handling:**
- **ACID Compliance:** SQLite ensures ACID compliance, guaranteeing the reliability and consistency of transactions. This is crucial for handling e-commerce transactions securely, such as processing payments and updating inventory levels.
  - **Order Processing:** SQLite facilitates transactional operations involved in order processing, including updating product quantities, deducting payment amounts, and generating order confirmations.
3. **Backend Logic Implementation:**
- **Business Rules Enforcement:** SQLite allows the backend to enforce business rules, such as pricing policies, shipping rules, and promotional discounts. This ensures that all transactions adhere to predefined business logic and policies.
  - **Data Validation:** SQLite supports data validation mechanisms to ensure that incoming data from users or external sources meets predefined criteria. This helps maintain data integrity and prevents errors or inconsistencies in the database.
4. **Performance and Scalability:**
- **Efficient Query Execution:** SQLite's efficient query processing capabilities ensure fast and responsive backend operations, even with large datasets. This helps maintain optimal performance for user interactions, such as browsing products and placing orders.
  - **Scalability for Small to Medium Applications:** While SQLite may not be suitable for extremely high-traffic websites, it is well-suited for small to medium-sized e-commerce platforms, providing adequate performance and scalability for moderate workloads.
5. **Integration with Python:**
- **Seamless Integration:** SQLite integrates seamlessly with Python through the sqlite3 module, allowing backend developers to interact with the database using familiar Python syntax and conventions.

- **ORM Support:** Python frameworks like Django and Flask offer ORM support for SQLite, simplifying database operations and reducing the amount of boilerplate code required for database interactions.

The SQLite serves as a reliable and efficient backend database solution for the E-Commerce Management System project, supporting critical operations such as data storage, transaction handling, business logic implementation, and performance optimization. Its simplicity, reliability, and integration with Python make it a valuable asset for developing and maintaining the backend infrastructure of the e-commerce platform.

### 2.2.2 PYTHON:



Python is a high-level, interpreted programming language renowned for its simplicity, readability, and versatility. Created by Guido van Rossum and first released in 1991, Python has since gained widespread adoption and popularity across various domains, including web development, data science, artificial intelligence, scientific computing, automation, and more. Here are some key characteristics and features of Python:

1. **Easy to Learn and Read:** Python's syntax emphasizes readability and clarity, making it accessible for beginners and enjoyable for experienced developers. Its straightforward and concise syntax reduces the learning curve and encourages good coding practices.
2. **Versatile and Multi-Purpose:** Python is a general-purpose programming language, meaning it can be used for a wide range of applications. Whether you're building web applications, analyzing data, scripting automation tasks, or developing machine learning models, Python provides powerful tools and libraries to support diverse use cases.
3. **Interpreted and Interactive:** Python is an interpreted language, which means that code is executed line by line by an interpreter rather than compiled into machine code beforehand. This allows for interactive development and rapid prototyping, as code changes can be immediately tested and executed without the need for compilation.
4. **Dynamic Typing and Strong Typing:** Python is dynamically typed, meaning variable types are inferred at runtime and can change during execution. However, Python is also

strongly typed, enforcing strict type checking to prevent unintended type errors and ensure code reliability.

5. **Extensive Standard Library:** Python comes with a comprehensive standard library that provides ready-to-use modules and functions for common tasks such as file I/O, networking, data manipulation, and more. This extensive library ecosystem simplifies development and reduces the need for external dependencies.
6. **Rich Ecosystem of Third-Party Libraries and Frameworks:** In addition to its standard library, Python boasts a vibrant ecosystem of third-party libraries and frameworks contributed by the community. These libraries cover a wide range of domains, including web development (Django, Flask), scientific computing (NumPy, SciPy), data analysis (Pandas), machine learning (TensorFlow, PyTorch), and more.
7. **Platform Independence:** Python is platform-independent, meaning code written in Python can run on various operating systems, including Windows, macOS, Linux, and more. This portability ensures that Python applications can be deployed and executed across diverse environments with minimal modifications.
8. **Community and Support:** Python has a large and active community of developers, enthusiasts, and contributors who contribute to its ongoing development and maintenance. This vibrant community provides ample resources, documentation, tutorials, and forums for learning and support.

Python is a versatile, user-friendly, and powerful programming language that excels in a wide range of applications. Its simplicity, readability, extensive library ecosystem, and active community make it an ideal choice for beginners and seasoned developers alike.

### **PYTHON SERVES AS AN ASSET:**

Python plays a crucial role in the E-Commerce Management System project, offering a wide array of functionalities and benefits that contribute to the development and operation of the system:

1. **Backend Development:** Python is well-suited for backend development due to its simplicity, readability, and extensive library support. Backend components of the e-commerce platform, such as server-side logic, data processing, and API development, can be efficiently implemented using Python frameworks like Django or Flask.

2. **Database Interaction:** Python seamlessly integrates with databases, allowing for easy interaction with backend databases such as MySQL or SQLite. Python's database libraries (e.g., SQLAlchemy) facilitate database operations such as querying, insertion, updating, and deletion, enabling efficient management of product data, user profiles, and order information.
3. **Web Development:** Python frameworks like Django and Flask are commonly used for web development, providing tools and utilities for building robust and scalable web applications. With Python, developers can create user-friendly interfaces, implement authentication mechanisms, and handle HTTP requests and responses, essential for an e-commerce platform's frontend and backend integration.
4. **Automation and Scripting:** Python's scripting capabilities are valuable for automating repetitive tasks, such as data processing, report generation, and system maintenance. Automation scripts can streamline administrative tasks, enhance productivity, and ensure the smooth operation of the e-commerce system.
5. **Data Analysis and Insights:** Python's extensive ecosystem of data analysis libraries (e.g., Pandas, NumPy) enables developers to analyze and derive insights from e-commerce data, such as sales trends, customer behavior, and inventory management. These insights can inform strategic decisions, optimize marketing campaigns, and improve the overall performance of the e-commerce platform.
6. **Integration with Third-Party Services:** Python's flexibility and compatibility with various APIs make it suitable for integrating third-party services into the e-commerce platform. Whether integrating payment gateways, shipping providers, or analytics tools, Python facilitates seamless communication and integration with external services, enhancing the functionality and capabilities of the system.
7. **Community and Support:** Python benefits from a large and active community of developers, enthusiasts, and contributors who provide resources, documentation, and support. Leveraging Python's community-driven ecosystem, developers can access tutorials, forums, and libraries to address challenges, learn best practices, and accelerate development efforts.

Python's versatility, ease of use, extensive library ecosystem, and active community support make it an indispensable tool for developing and operating the E-Commerce Management System project. Whether it's backend development, database interaction, web development,

automation, data analysis, or integration with third-party services, Python empowers developers to build a robust, scalable, and feature-rich e-commerce platform.

## **PYTHON FOR FRONTEND:**

While Python is predominantly used for backend development, it can still play a role in frontend development for the E-Commerce Management System project through various means:

1. **Template Engines:** Python web frameworks like Django come equipped with powerful template engines (e.g., Django Template Language) that allow developers to generate dynamic HTML content seamlessly. Python code embedded within templates can handle logic for rendering product listings, user profiles, shopping carts, and other frontend components.
2. **API Integration:** Python can facilitate communication between the frontend and backend of the e-commerce platform by serving as an intermediary for API requests and responses. Python-based backend services can expose RESTful APIs that deliver data to the frontend, enabling dynamic updates and interactions without page reloads.
3. **Client-Side Scripting:** Although JavaScript is typically the language of choice for client-side scripting in web development, Python can still be employed for certain client-side tasks using libraries like Brython (Python implementation for the browser). While limited in comparison to JavaScript, Brython enables developers to write Python code directly in HTML pages for frontend interactions.
4. **Data Processing:** Python's data processing capabilities can be leveraged in the frontend to manipulate and transform data before rendering it to users. Libraries like Pandas or NumPy can assist in performing calculations, filtering data, or generating visualizations within the browser, enhancing the user experience with dynamic and interactive content.
5. **Build Tools and Task Runners:** Python-based build tools and task runners (e.g., Fabric, Invoke) can streamline frontend development workflows by automating repetitive tasks such as compiling assets, optimizing images, or running tests. These tools complement frontend frameworks and libraries to enhance productivity and maintain code quality.



6. **Cross-Platform Development:** Python frameworks like Kivy or BeeWare enable developers to build cross-platform desktop and mobile applications using Python. While not traditional web frontend solutions, these frameworks allow for Python-centric development of rich, native-like user interfaces across multiple platforms.
7. **Hybrid Approaches:** Python can be used in conjunction with JavaScript frameworks (e.g., React, Vue.js) through hybrid approaches like server-side rendering or isomorphic JavaScript. With these approaches, Python handles initial rendering on the server, while JavaScript takes over for client-side interactions, providing a balance between Python's backend strengths and JavaScript's frontend capabilities.

While Python may not be as prevalent in frontend development as JavaScript, it can still complement frontend efforts in certain scenarios, particularly when integrated with Python-based web frameworks or utilized for specific frontend tasks such as templating, data processing, or build automation. Ultimately, the choice of frontend technologies will depend on project requirements, team preferences, and the desired user experience.

### **3. REQUIREMENT AND ANALYSIS**

#### **3.1 REQUIREMENT SPECIFICATION:**

##### **1. Introduction**

- **Purpose:** The purpose of the E-Commerce Management System is to provide a comprehensive platform for online shopping, enabling users to browse products, make purchases, and manage their accounts.
- **Scope:** The system will include features for user registration, product management, shopping cart functionality, order processing, and administrative tools for managing the e-commerce platform.
- **Stakeholders:** Customers, administrators, developers, business owners.

##### **2. Functional Requirements**

###### **1. User Management:**

- Users should be able to register for an account with a unique username and password.
- Users should be able to log in to their accounts securely.
- Users should be able to update their profile information, including shipping addresses and payment methods.

###### **2. Product Management:**

- Administrators should be able to add, edit, and delete product listings.
- Product listings should include details such as product name, description, price, and availability.
- Product images should be uploadable and displayed alongside product listings.

###### **3. Shopping Cart:**

- Users should be able to add items to their shopping cart.
- Users should be able to view their cart contents, update quantities, and remove items.
- Users should be able to proceed to checkout and complete their purchase securely.

**4. Order Processing:**

- Users should receive order confirmation emails upon successful purchase.
- Administrators should be able to view and manage orders, including order status and shipping information.
- Orders should be processed securely, with payment transactions handled through a secure payment gateway.

**5. Search and Filtering:**

- Users should be able to search for products using keywords.
- Users should be able to filter search results by category, price range, and other criteria.

**6. User Reviews and Ratings:**

- Users should be able to leave reviews and ratings for products they have purchased.
- Reviews and ratings should be displayed alongside product listings.

**7. User Management:**

- Users should be able to register for an account with a unique username and password.
- Users should be able to log in to their accounts securely.
- Users should be able to update their profile information, including shipping addresses and payment methods.

**8. Product Management:**

- Administrators should be able to add, edit, and delete product listings.
- Product listings should include details such as product name, description, price, and availability.
- Product images should be uploadable and displayed alongside product listings.

**9. Shopping Cart:**

- Users should be able to add items to their shopping cart.
- Users should be able to view their cart contents, update quantities, and remove items.
- Users should be able to proceed to checkout and complete their purchase securely.

**10. Order Processing:**

- Users should receive order confirmation emails upon successful purchase.
- Administrators should be able to view and manage orders, including order status

and shipping information.

- Orders should be processed securely, with payment transactions handled through a secure payment gateway.

#### **11. Search and Filtering:**

- Users should be able to search for products using keywords.
- Users should be able to filter search results by category, price range, and other criteria.

#### **12. User Reviews and Ratings:**

- Users should be able to leave reviews and ratings for products they have purchased.
- Reviews and ratings should be displayed alongside product listings.

#### **13. Admin Dashboard:**

- Administrators should have access to a dashboard to monitor sales, manage inventory, and track user activity.
- Dashboard should include features such as sales reports, inventory management tools, and user analytics.

### **3. Non-Functional Requirements**

#### **1. Performance:**

- The system should be responsive and able to handle concurrent user traffic without significant slowdowns.
- Page load times should be optimized to ensure a seamless user experience.

#### **2. Security:**

- User data should be encrypted and stored securely.
- Payment transactions should be processed through a secure payment gateway.
- The system should implement measures to prevent common security threats such as SQL injection and cross-site scripting (XSS).

#### **3. Scalability:**

- The system should be scalable to accommodate increasing numbers of users, products, and transactions.
- Scalability should be achieved through efficient database design, caching mechanisms, and load balancing techniques.

#### **4. Usability:**

- The user interface should be intuitive, easy to navigate, and accessible across

different devices and screen sizes.

- User interactions should be streamlined and require minimal steps to complete tasks.

#### **5. Reliability:**

- The system should be reliable, with minimal downtime and robust error handling mechanisms in place.
- Data integrity should be maintained through regular backups and database maintenance procedures.

### **4. Constraints**

**Technology Stack:** Python for backend development, MySQL or SQLite for database management.

- **Budget and Time Constraints:** The project must adhere to predefined budget and timeline constraints.

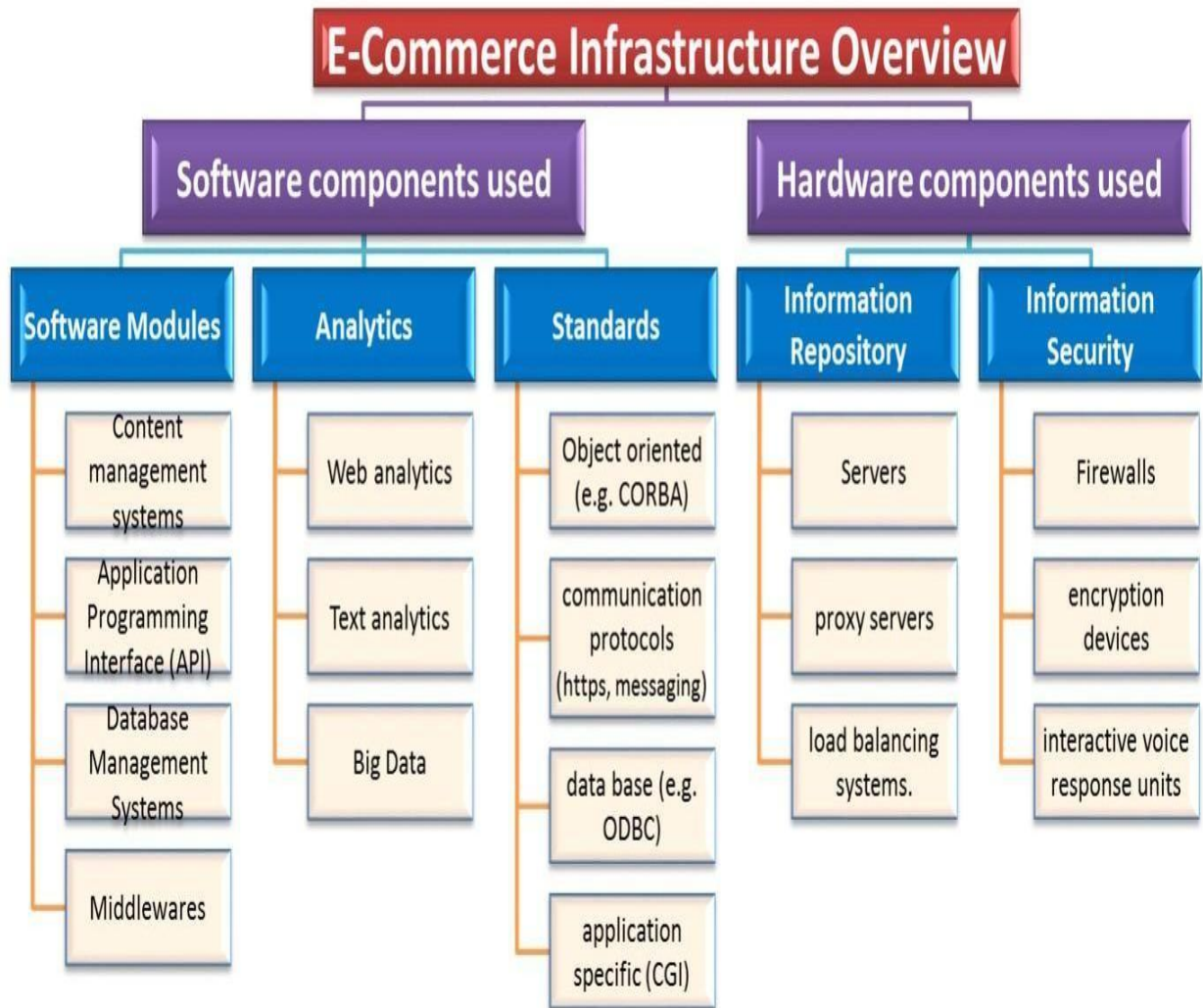
### **5. Assumptions and Dependencies**

- The system will rely on external services such as payment gateways and shipping providers for certain functionalities.
- The availability of development resources and expertise will impact project timelines and deliverables.

### **6. Sign-Off**

- The requirement specification document must be reviewed and approved by all relevant stakeholders before proceeding with development.

Fig 1: HARDWARE AND SOFTWARE REUIREMENTS



### **3.1.1 HARDWARE**

The hardware requirements for the E-Commerce Management System project depend on factors such as the anticipated user load, scalability requirements, and specific infrastructure considerations. Here's a general outline of the hardware components needed:

Server Hardware:

**1. Processor (CPU):**

- Multi-core processors with sufficient processing power to handle concurrent user requests efficiently.
- Consideration of CPU cores and clock speed to ensure smooth operation under varying workloads.

**2. Memory (RAM):**

- Adequate RAM capacity to support the application's memory requirements and database operations.
- Minimum of 8GB RAM recommended for basic deployments, with additional RAM for scalability.

**3. Storage (SSD):**

- Solid-state drives (SSDs) are preferable over traditional hard disk drives (HDDs) for improved read/write speeds.
- Storage capacity should accommodate the operating system, application code, database files, and any associated media assets (images, videos).

**4. Network Interface:**

- Gigabit Ethernet interface for fast and reliable network connectivity.
- Consideration of network bandwidth to handle incoming and outgoing data traffic, especially during peak usage periods.

Additional Considerations:

**1. Redundancy and Fault Tolerance:**

- Implementation of redundancy mechanisms such as RAID (Redundant Array of Independent Disks) for data protection and fault tolerance.
- Backup power supply (e.g., uninterruptible power supply or UPS) to mitigate the risk of data loss due to power outages.

## **2. Scalability and Load Balancing:**

- Provisioning for horizontal scalability through load balancing across multiple server instances.
- Consideration of cloud-based solutions for elastic scalability and resource optimization based on fluctuating demand.

## **3. Monitoring and Management:**

- Installation of monitoring tools to track server performance metrics (CPU usage, memory utilization, disk I/O).
- Remote management capabilities for system administration tasks and troubleshooting.

## **4. Security Measures:**

- Implementation of security measures such as firewalls, intrusion detection/prevention systems, and regular software updates to protect against cyber threats.
- Secure physical location for server deployment to prevent unauthorized access and ensure data integrity.

## **5. Compliance and Regulations:**

- Compliance with industry standards and regulations regarding data privacy, security, and confidentiality (e.g., GDPR, PCI DSS).

## **6. Budget and Resource Allocation:**

- Consideration of budget constraints and resource allocation for hardware procurement, maintenance, and ongoing operational expenses.

### **3.1.2 SOFTWARE**

The software requirements for the E-Commerce Management System project encompass various components for both development and deployment. Here's an overview of the essential software needed:

#### **Development Environment:**

##### **1. Python 3.x:**

- Core programming language for backend development.
- Ensure compatibility with required Python packages and frameworks.

##### **2. Integrated Development Environment (IDE):**



- Recommended IDEs include PyCharm, Visual Studio Code, or Sublime Text.
- Provides features such as code editing, debugging, and version control integration.

### 3. **Python Packages:**

- Django or Flask: Web frameworks for backend development.
- Additional packages for database interaction (e.g., psycopg2 for PostgreSQL, mysql-connector-python for MySQL).

### 4. **Version Control:**

- Git: Version control system for managing codebase changes.
- Platforms like GitHub, GitLab, or Bitbucket for hosting repositories and collaboration.

## **Database Management:**

### 1. **MySQL or SQLite:**

- MySQL: Robust relational database management system (RDBMS) for production deployments.
- SQLite: Lightweight, embedded database for development and testing environments.

### 2. **Database Administration Tools:**

- MySQL Workbench: GUI tool for MySQL database administration.
- phpMyAdmin: Web-based interface for managing MySQL databases.

## **Frontend Technologies:**

### 1. **HTML5, CSS3, JavaScript:**

- Fundamental web technologies for building user interfaces and interactive features.
- JavaScript libraries or frameworks (e.g., React, Vue.js) for enhanced frontend functionality (optional).

### 2. **Frontend Frameworks (optional):**

- Bootstrap, Materialize CSS, or Foundation for responsive design and UI components.
- jQuery or other JavaScript libraries for DOM manipulation and event handling.

## **Web Server:**

### **1. Nginx or Apache:**

- Web server software for serving HTTP requests and hosting web applications.
- Configuration settings for proxying requests to backend application servers (e.g., Django, Flask).

## **Deployment and Infrastructure:**

### **1. Operating System:**

- Linux-based distributions (e.g., Ubuntu, CentOS) preferred for server deployments.
- Windows Server as an alternative for specific environments.

### **2. Containerization Tools (optional):**

- Docker: Containerization platform for packaging and deploying applications.
- Docker Compose: Tool for defining and running multi-container Docker applications.

### **3. Cloud Platform (optional):**

- AWS, Google Cloud Platform, Microsoft Azure: Cloud services for scalable and reliable hosting.
- Services such as EC2 (Elastic Compute Cloud), RDS (Relational Database Service), and S3 (Simple Storage Service) for infrastructure components.

### **4. Monitoring and Logging:**

- Prometheus, ELK Stack (Elasticsearch, Logstash, Kibana): Tools for monitoring system performance and analyzing logs.

### **5. Security Measures:**

- SSL/TLS certificates for securing web traffic (HTTPS).
- Firewall configurations and security policies to protect against cyber threats.

### **6. Development and Deployment Tools:**

- Package manager (e.g., Pip) for installing Python dependencies.
- Build automation tools (e.g., Fabric, Ansible) for automating deployment tasks.
- Continuous integration/continuous deployment (CI/CD) pipelines for automated testing and deployment.

## **Additional Software:**

### **1. Task Runners:**

- npm: Node.js package manager for frontend build automation (if using JavaScript-based frontend frameworks).

### **2. Testing Frameworks:**

- pytest or unittest: Testing frameworks for writing and executing automated tests for backend code.
- Selenium WebDriver: Tool for automated browser testing (frontend).

### **3. Documentation Tools:**

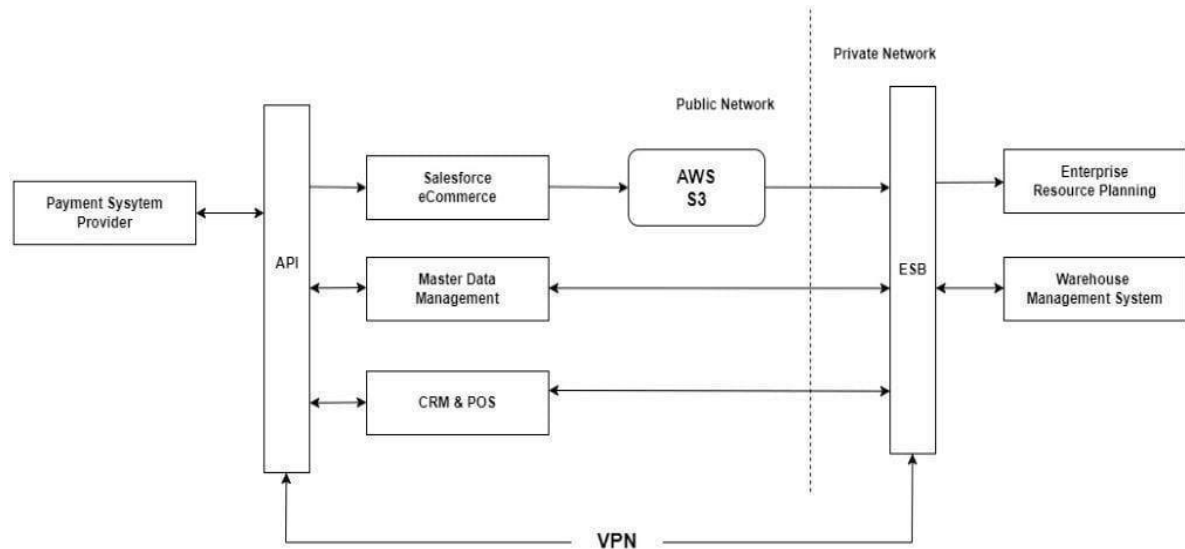
- Sphinx, MkDocs: Documentation generators for creating project documentation and user guides.

### **4. Collaboration Tools:**

- Communication platforms (e.g., Slack, Microsoft Teams) for team collaboration and coordination.
- Project management tools (e.g., Jira, Trello) for task tracking and project planning.

## **4. ARCHITECTURE DIAGRAM**

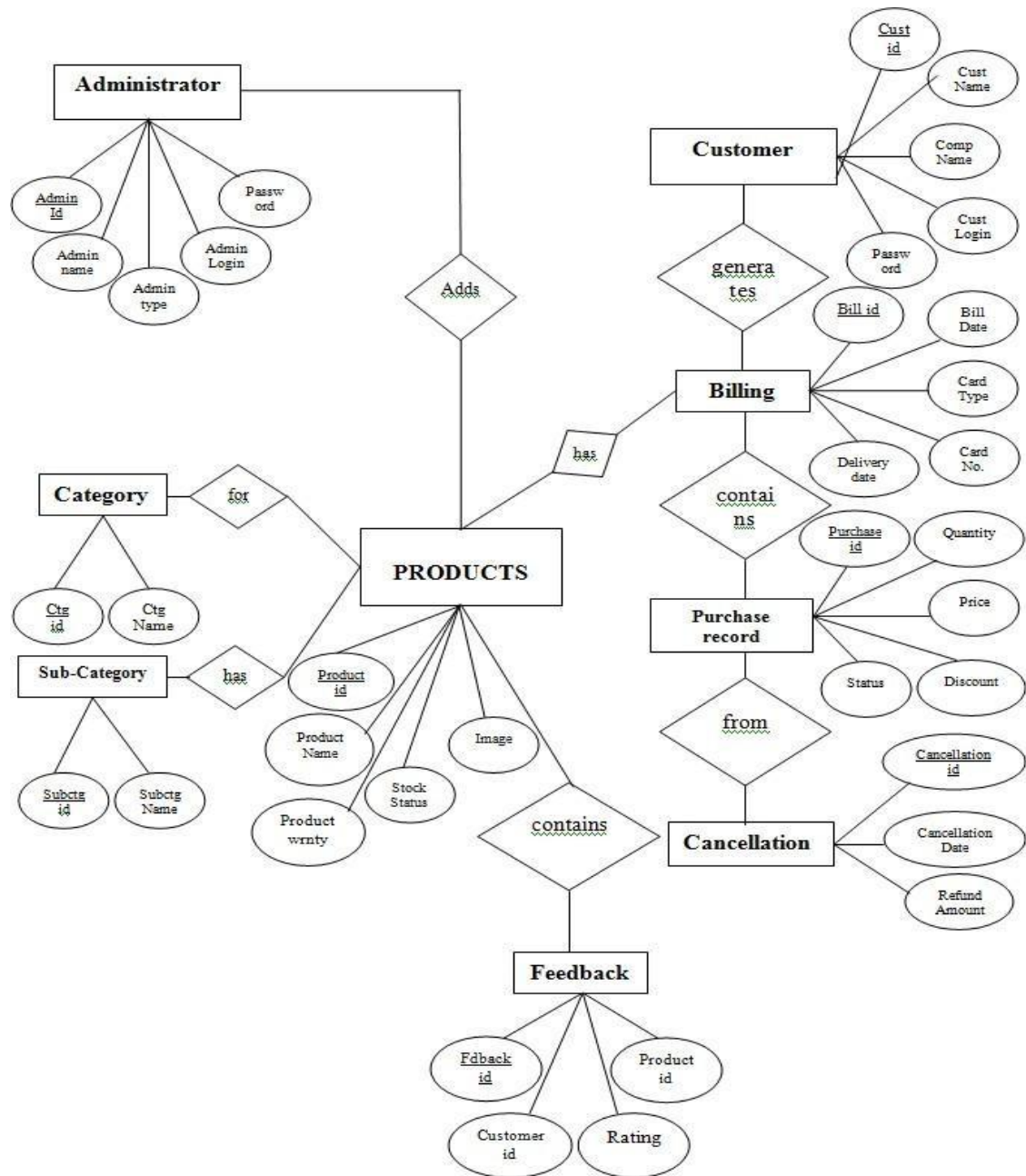
FIG 2: CONTEXT LEVEL DATA FLOW DIAGRAM



## **ENTITY RELATIONSHIP DIAGRAM**

The Entity-Relationship (ER) diagram for our e-commerce management system illustrates the entities involved, detailing their attributes and the connections between them. These entities encompass elements such as Products, Users, Administrators, Discounts, Orders, and Payments, with relationships denoted by terms like Manages, Offers, Purchases, and Transactions.

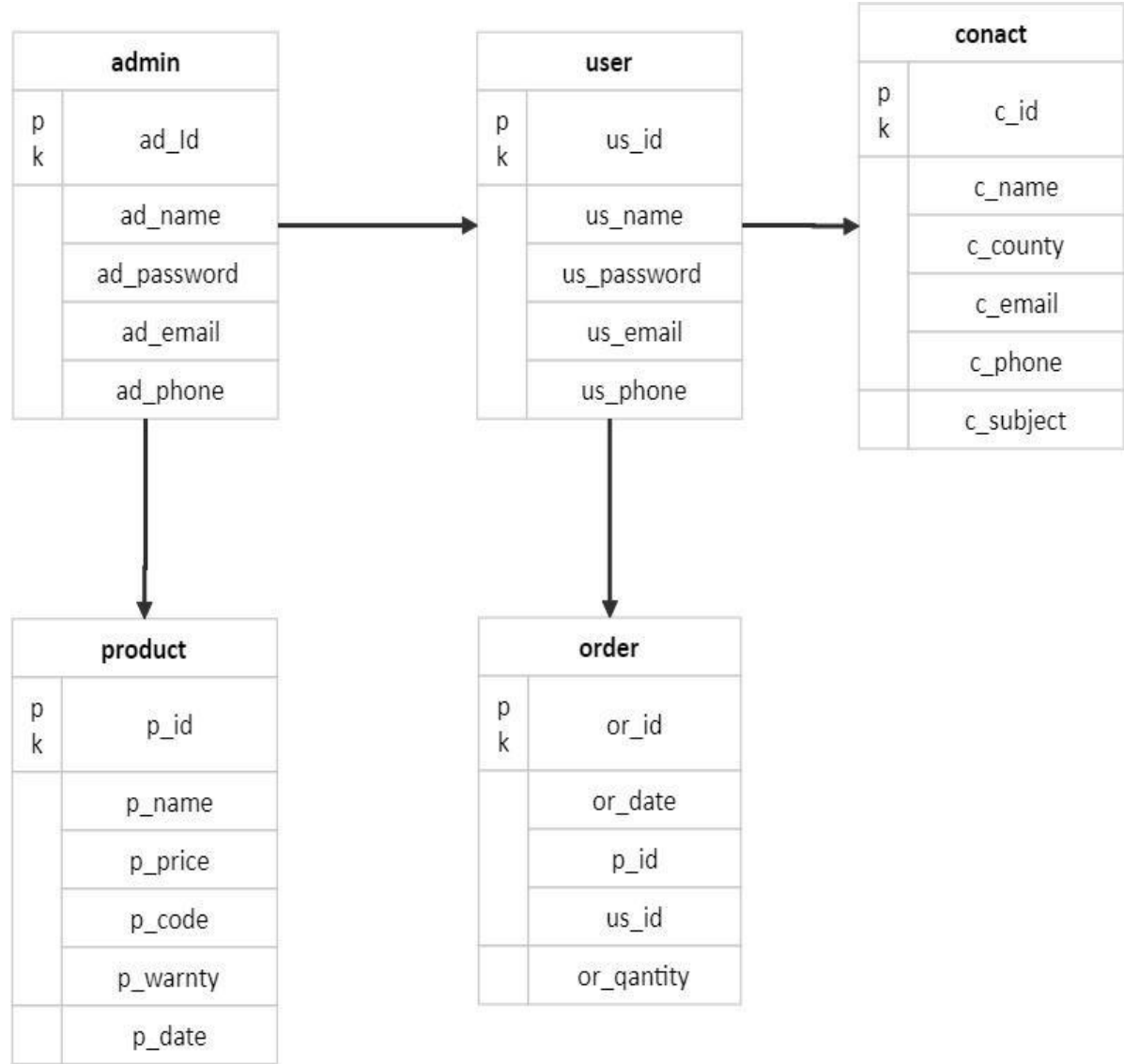
FIG 3: ENTITY RELATIONSHIP DIAGRAM



# ENTITY RELATIONSHIP MAPPING RULES

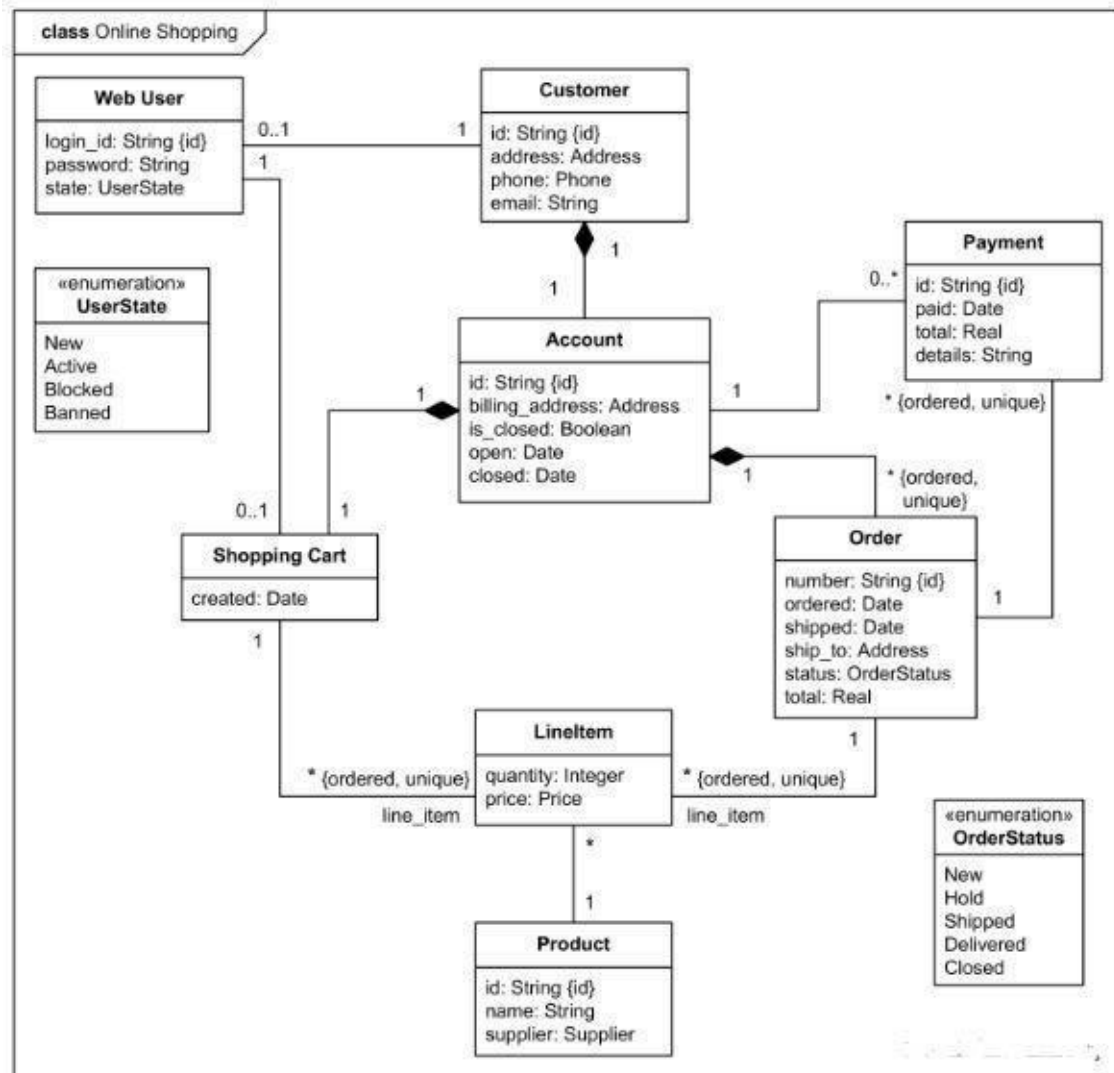
The ER Mapping diagram showcases the conversion of the ER diagram into relational schemas tailored for the e-commerce management system. Each entity and relationship depicted in the ER diagram is transformed into a corresponding table within the relational model. Below, you'll find a comprehensive breakdown of this mapping process.

FIG 4: ENTITY RELATIONSHIP MAPPING RULES



## 1. UML CLASS DIAGRAM & USECASE DIAGRAM

FIG 5: UML CLASS DIAGRAM



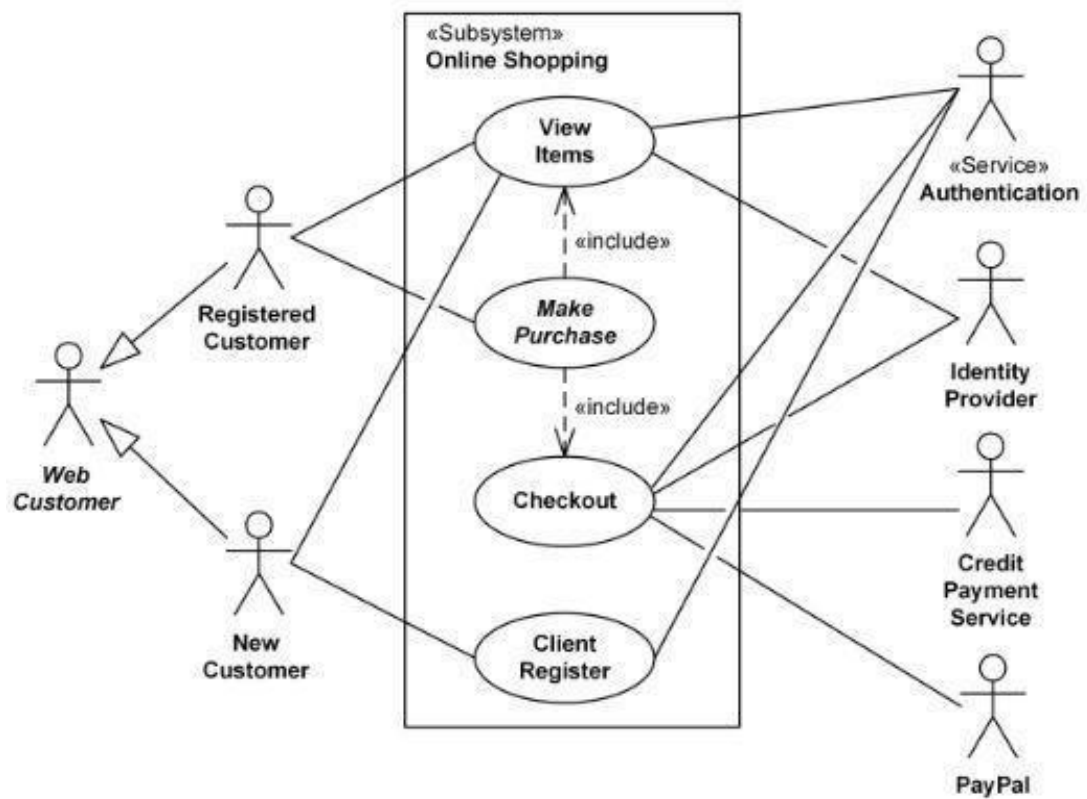


FIG 6: USECASE DIAGRAM



## 5. MODULES

### 1. Management Module:

- Login/Registration: Handles user authentication, including login, registration, and password recovery.
- Profile Management: Allows users to view and update their profile details, such as shipping address and contact information.
- Role Management: Differentiates between customer roles and admin roles, managing permissions accordingly.

### 2. Product Management Module:

- Product Catalog: Displays the list of products with details like name, description, price, and availability.
- Category Management: Organizes products into categories and subcategories for easier navigation.
- Inventory Management: Tracks stock levels and product availability in real time.
- Product Search & Filter: Allows users to search for products using various filters such as price range, categories, brands, etc.

### 3. Order Management Module

- Cart Management: Enables users to add, update, or remove products from their cart.
- Checkout Process: Guides users through selecting shipping methods, payment options, and reviewing the order before final submission.
- Order Processing: Handles order confirmation, payment verification, and dispatching.
- Order Tracking: Allows users to track the status of their orders (e.g., pending, shipped, delivered)

### 4. Admin Dashboard Module

- Order Management: Admin can view, update, or cancel orders.
- Inventory Control: Admin can manage stock levels, add new products, and update product details.
- User Management: Admin can manage customers, view customer activity, and handle issues.
- Reporting & Analytics: Provides insights into sales, product performance, customer behavior, and other business metrics.

## 6. SOFTWARE DEVELOPMENT MODEL

Implementing the Agile software development model for your e-commerce product catalog project would involve dividing the work into small, iterative cycles called sprints. Each sprint delivers a specific part of the system, and the process emphasizes flexibility, ongoing feedback, and continuous improvement. Here's a breakdown of how Agile would apply to this project:

### 1. Define Requirements and Initial Backlog

- **Initial Planning:** Start by identifying core features based on requirements, such as user management, product management, cart and order functionality, and sales forecasting.
- **Backlog Creation:** Create a product backlog, which is a prioritized list of tasks or user stories (e.g., "As a user, I want to add products to my cart so that I can purchase them later").
- **User Stories Examples:**
  - "As an admin, I want to add products to the catalog so that users can view available items."
  - "As a user, I want to register and log in so that I can make purchases."

### 2. Sprint Planning and Task Breakdown

- **Define Sprints:** Break down the project into short sprints, usually lasting 1-2 weeks.
- **Select Tasks for Each Sprint:** Choose a few high-priority tasks or user stories from the backlog for each sprint, depending on team capacity and priorities.
- **Task Assignment:** Break down each feature into manageable tasks (e.g., creating the Users table, developing a function for product insertion, implementing cart functionality) and assign these to team members.

### 3. Develop and Test in Short Cycles

- **Development:** Developers work on assigned tasks, focusing on coding and unit testing each function. For example, one sprint may focus on implementing and testing `create_tables` and `add_initial_products`.
- **Frequent Testing:** Agile emphasizes testing at every stage, so each feature is developed and unit-tested immediately. Tests for each function (e.g., `create_connection`, `add_initial_products`) ensure the system remains reliable as it grows.

### 4. Daily Standups

- **Daily Sync:** Hold daily meetings (standups) to discuss progress, roadblocks, and next steps. Team members share what they accomplished, what they're working on, and any issues they face.
- **Example:** A developer might mention they're encountering an issue with FOREIGN KEY constraints in the Orders table, allowing other team members to help troubleshoot.

## 5. End of Sprint Review and Retrospective

- **Review:** At the end of each sprint, demonstrate completed features to stakeholders or team members. Collect feedback and verify if the implemented features meet the initial requirements.
- **Retrospective:** Reflect on what went well, what didn't, and areas for improvement in the next sprint. For example, if data validation tests took longer than expected, you might plan better test strategies for the next sprint.

## 6. Incorporating Feedback and Iterating

- **Continuous Feedback Loop:** Based on stakeholder feedback, adjust the backlog and refine requirements. For example, feedback may reveal that users want more detailed error messages when adding items to their cart.
- **Adjust Priorities:** New tasks are added to the backlog as needed, and priorities are adjusted. If sales forecasting is identified as critical, it may be prioritized earlier in future sprints.

## 7. Deliver Incremental Releases

- **Regular Releases:** At the end of each sprint, deploy the completed features to a testing or staging environment. In an Agile approach, each sprint should result in a potentially shippable product increment.
- **User Testing and Feedback:** Have actual users or stakeholders test the released version, which helps ensure the product is aligned with expectations.

## 8. Final Release and Maintenance

- **Final Sprint:** The last sprint can focus on polishing, bug fixing, and finalizing features, making the system ready for production.
- **Maintenance:** Agile doesn't end at launch; after the final release, ongoing sprints can address new requirements, user feedback, or feature enhancements, ensuring the system remains up-to-date.

## Example Agile Sprint Breakdown for Your Project:

### Sprint 1: Set Up Database and Basic User Authentication

- Set up `ecommerce.db` and `create_connection`.
- Implement `create_tables` to initialize tables for Users, Products, Orders, etc.
- Develop and test basic user registration and login functions.

### Sprint 2: Implement Product Catalog Management

- Implement and test `add_initial_products` for inserting products.
- Develop CRUD operations for products (add, update, delete, view).
- Display products to users with filtering and sorting options.

### **Sprint 3: Cart and Order Management**

- Implement Cart functionality (add to cart, view cart, update quantities).
- Develop order processing (create an order, calculate total, save to Orders).
- Link OrderDetails to orders and ensure foreign key integrity.

### **Sprint 4: Sales Forecasting Module**

- Implement data upload functionality.
- Develop the forecasting model (e.g., linear regression for sales predictions).
- Display forecasted data to users.

### **Sprint 5: Testing, Refinement, and Documentation**

- Conduct thorough testing on all modules.
- Refine features based on user feedback.

## **9. Develop and Test in Short Cycles**

- **Development:** Developers work on assigned tasks, focusing on coding and unit testing each function. For example, one sprint may focus on implementing and testing `create_tables` and `add_initial_products`.
- **Frequent Testing:** Agile emphasizes testing at every stage, so each feature is developed and unit-tested immediately. Tests for each function (e.g., `create_connection`, `add_initial_products`) ensure the system remains reliable as it grows.

## **10. Daily Standups**

- **Daily Sync:** Hold daily meetings (standups) to discuss progress, roadblocks, and next steps. Team members share what they accomplished, what they're working on, and any issues they face.
- **Example:** A developer might mention they're encountering an issue with FOREIGN KEY constraints in the Orders table, allowing other team members to help troubleshoot.

## **11. End of Sprint Review and Retrospective**

- **Review:** At the end of each sprint, demonstrate completed features to stakeholders or team members. Collect feedback and verify if the implemented features meet the initial requirements.
- **Retrospective:** Reflect on what went well, what didn't, and areas for improvement in the next sprint. For example, if data validation tests took longer than expected, you might plan better test strategies for the next sprint.

## **12. Incorporating Feedback and Iterating**

- **Continuous Feedback Loop:** Based on stakeholder feedback, adjust the backlog and refine requirements. For example, feedback may reveal that users want more detailed

error messages when adding items to their cart.

- **Adjust Priorities:** New tasks are added to the backlog as needed, and priorities are adjusted. If sales forecasting is identified as critical, it may be prioritized earlier in future sprints.

### 13. Deliver Incremental Releases

- **Regular Releases:** At the end of each sprint, deploy the completed features to a testing or staging environment. In an Agile approach, each sprint should result in a potentially shippable product increment.
- **User Testing and Feedback:** Have actual users or stakeholders test the released version, which helps ensure the product is aligned with expectations

## **7.TESTING SOFTWARE**

### **Unit Testing**

- **Objective:** Test individual functions to ensure they work as expected.
- **Tools:** unit test or pytest in Python.
- **Examples:**
  - Test `create_connection` to confirm a database connection is established.
  - Test `create_tables` to verify all tables (Users, Products, Orders, OrderDetails, Cart) are created successfully.
  - Test `add_initial_products` to ensure that initial products are inserted with correct data.

In the context of your e-commerce product catalog code, unit testing would interact and work with the code as follows:

1. **Testing Database Connections** (`create_connection` function)
  - A unit test for `create_connection` would ensure that the function successfully creates a connection to the database.
  - You could mock this connection in the test to avoid needing an actual database.
  - **Test Example:** Confirm that `create_connection` returns a valid SQLite connection object.

2. **Testing Table Creation** (`create_tables` function)

This function initializes the database by creating the required tables if they don't already exist.

- A unit test would verify that each table (Users, Products, Orders, OrderDetails, Cart) is created correctly and can be accessed without error.
  - **Test Example:** After running `create_tables`, check that each table exists and has the correct columns.
3. **Testing Initial Product Insertion** (`add_initial_products` function)
    - This function populates the Products table with some predefined items.
    - A unit test would ensure that products are inserted correctly, with each product's Name, Description, Price, and Stock matching the intended values.
    - **Test Example:** Verify that `add_initial_products` inserts the expected rows, with no missing or extra products.
  4. **Setup and Teardown** (using `setup_database` function)
    - **Setup:** Before each test, `setup_database` can be called to reset the database and create a fresh, predictable environment.
    - **Teardown:** After each test, the test framework could delete or reset the database, ensuring that each test starts with a clean slate.
    - **Test Example:** In your test framework, set up a fresh database and verify that Users, Products, Orders, OrderDetails, and Cart tables are empty or have initial data only.
  5. **Example Unit Tests Using Python's unittest Framework**
    - You could use `unittest` to define tests like so:

```
python  
Copy code
```

```

import unittest
import sqlite3
from ecommerce_catalog import create_connection, create_tables,
add_initial_products

class TestEcommerceCatalog(unittest.TestCase):

    def setUp(self):
        self.conn = create_connection()
        create_tables(self.conn)

    def tearDown(self):
        self.conn.close()

    def test_create_connection(self):
        self.assertIsInstance(self.conn, sqlite3.Connection)

    def test_create_tables(self):
        cursor = self.conn.cursor()
        cursor.execute("SELECT name FROM sqlite_master WHERE
type='table';")
        tables = [table[0] for table in cursor.fetchall()]
        expected_tables = ['Users', 'Products', 'Orders', 'OrderDetails', 'Cart']
        self.assertTrue(set(expected_tables).issubset(set(tables)))

    def test_add_initial_products(self):
        add_initial_products(self.conn)
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Products;")
        products = cursor.fetchall()
        self.assertEqual(len(products), 5) # Expect 5 initial products

if __name__ == '__main__':
    unittest.main()

```

## OUTPUT:

- **Objective:** Test individual functions to ensure they work as expected.
- **Tools:** unit test or pytest in Python.
- **Examples:**
  - Test create\_connection to confirm a database connection is established.
  - Test create\_tables to verify all tables (Users, Products, Orders, OrderDetails, Cart) are created successfully.
  - Test add\_initial\_products to ensure that initial products are inserted with correct data.

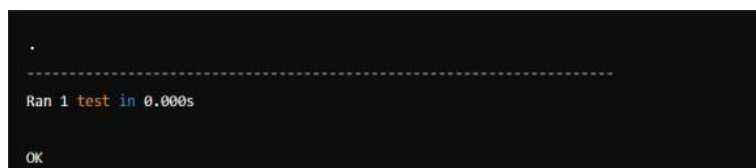


FIG 7: TESTING OUTPUT 1

```
test_create_connection (__main__.TestEcommerceCatalog) ... ok
test_create_tables (__main__.TestEcommerceCatalog) ... ok
test_add_initial_products (__main__.TestEcommerceCatalog) ... ok

-----
Ran 3 tests in 0.025s

OK
```

FIG 8: TESTING OUTPUT 2

Explanation of Output:

1. Each test case (test\_create\_connection, test\_create\_tables, test\_add\_initial\_products) passes:
  - ok means the test passed without issues.
  - Ran 3 tests indicates that all three test functions were executed.
  - OK at the end means that all tests were successful, and no errors or failures occurred.

If any of the tests fail, the output would show detailed information about the failure, including the test that failed and a traceback indicating the reason for the failure. This would typically happen if:

  - The database connection wasn't created properly.
  - The tables weren't created as expected.
  - The initial products weren't added to the Products table.

## How These Unit Tests Work with Your Code

1. **Isolation of Functions:** Each test focuses on a specific function, making sure it does its job correctly. The database connection, table creation, and data insertion are each tested independently.
2. **Setup and Teardown:** Tests run in a clean environment due to the setUp and tearDown methods, which ensure that each test interacts with a predictable and isolated version of the database.
3. **Feedback for Debugging:** If a test fails, you'll know exactly which part of the code needs fixing, helping you identify issues quickly without needing to check the entire flow.

Unit tests thus provide clear, detailed validation that each part of the e-commerce catalog functions as intended, without affecting production data or requiring a live environment.



## **8.PROGRAM CODE**

### **FRONT END:**

```
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
from datetime import datetime

class EcommerceApp:
    def __init__(self, root):
        self.root = root
        self.root.title("E-commerce Management System")

        # Connect to database
        self.conn = self.create_connection()
        self.cursor = self.conn.cursor()

        # Create main frame
        self.main_frame = ttk.Frame(self.root, padding="10")
        self.main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # User ID for logged-in user
        self.user_id = None

        # Create widgets
        self.create_widgets()

    def create_connection(self):
        conn = sqlite3.connect('ecommerce.db')
        return conn

    def create_widgets(self):
        # Add login/register buttons
        self.login_button = ttk.Button(self.main_frame, text="Login", command=self.login)
```

```

self.login_button.grid(row=0, column=0, padx=10, pady=10)

self.register_button = ttk.Button(self.main_frame, text="Register",
command=self.register)

self.register_button.grid(row=0, column=1, padx=10, pady=10)

# Add product catalog button
self.catalog_button = ttk.Button(self.main_frame, text="Product Catalog",
command=self.view_catalog)

self.catalog_button.grid(row=0, column=2, padx=10, pady=10)

1. # Add view cart button
self.view_cart_button = ttk.Button(self.main_frame, text="View Cart",
command=self.view_cart)

self.view_cart_button.grid(row=0, column=3, padx=10, pady=10)

def login(self):
    # Create login window
    self.login_window = tk.Toplevel(self.root)
    self.login_window.title("Login")

    ttk.Label(self.login_window, text="Username").grid(row=0, column=0, padx=10,
pady=10)

    self.username_entry = ttk.Entry(self.login_window)
    self.username_entry.grid(row=0, column=1, padx=10, pady=10)

    ttk.Label(self.login_window, text="Password").grid(row=1, column=0, padx=10,
pady=10)

    self.password_entry = ttk.Entry(self.login_window, show="*")
    self.password_entry.grid(row=1, column=1, padx=10, pady=10)

    self.login_submit_button = ttk.Button(self.login_window, text="Login",
command=self.check_login)

    self.login_submit_button.grid(row=2, column=0, columnspan=2, padx=10, pady=10)

```

```

def check_login(self):
    username = self.username_entry.get()
    password = self.password_entry.get()

    query = "SELECT UserID FROM Users WHERE Username = ? AND Password = ?"
    self.cursor.execute(query, (username, password))
    user = self.cursor.fetchone()

    if user:
        self.user_id = user[0]
        messagebox.showinfo("Login Success", "Welcome, {}".format(username))
        self.login_window.destroy()
    else:
        messagebox.showerror("Login Failed", "Invalid username or password")

def register(self):
    # Create register window
    self.register_window = tk.Toplevel(self.root)
    self.register_window.title("Register")

    ttk.Label(self.register_window, text="Username").grid(row=0, column=0, padx=10,
pady=10)
    self.new_username_entry = ttk.Entry(self.register_window)
    self.new_username_entry.grid(row=0, column=1, padx=10, pady=10)

    ttk.Label(self.register_window, text="Password").grid(row=1, column=0, padx=10,
pady=10)
    self.new_password_entry = ttk.Entry(self.register_window, show="*")
    self.new_password_entry.grid(row=1, column=1, padx=10, pady=10)

    ttk.Label(self.register_window, text="Email").grid(row=2, column=0, padx=10,
pady=10)
    self.email_entry = ttk.Entry(self.register_window)
    self.email_entry.grid(row=2, column=1, padx=10, pady=10)

```

```

        ttk.Label(self.register_window, text="Address").grid(row=3, column=0, padx=10,
pady=10)
        self.address_entry = ttk.Entry(self.register_window)
        self.address_entry.grid(row=3, column=1, padx=10, pady=10)

        self.register_submit_button = ttk.Button(self.register_window, text="Register",
command=self.create_user)
        self.register_submit_button.grid(row=4, column=0, columnspan=2, padx=10, pady=10)

    def create_user(self):
        username = self.new_username_entry.get()
        password = self.new_password_entry.get()
        email = self.email_entry.get()
        address = self.address_entry.get()

        query = "INSERT INTO Users (Username, Password, Email, Address) VALUES (?, ?,
?, ?)"
        self.cursor.execute(query, (username, password, email, address))
        self.conn.commit()

        messagebox.showinfo("Registration Success", "User registered successfully")
        self.register_window.destroy()

    def view_catalog(self):
        # Create catalog window
        self.catalog_window = tk.Toplevel(self.root)
        self.catalog_window.title("Product Catalog")

        query = "SELECT * FROM Products"
        self.cursor.execute(query)
        products = self.cursor.fetchall()

        for index, product in enumerate(products):
            ttk.Label(self.catalog_window, text=product[1]).grid(row=index, column=0,
padx=10, pady=10)

```

```

        ttk.Label(self.catalog_window, text="${:.2f}".format(product[3])).grid(row=index,
column=1, padx=10, pady=10)

        add_to_cart_button = ttk.Button(self.catalog_window, text="Add to Cart",
command=lambda p=product: self.add_to_cart(p))

        add_to_cart_button.grid(row=index, column=2, padx=10, pady=10)

def add_to_cart(self, product):
    if self.user_id is None:
        messagebox.showerror("Error", "You must be logged in to add products to your
cart.")
        return

    product_id, _, _, price, _ = product
    query = "INSERT INTO Cart (UserID, ProductID, Quantity) VALUES (?, ?, 1)"
    self.cursor.execute(query, (self.user_id, product_id))
    self.conn.commit()
    messagebox.showinfo("Success", "Product added to cart.")

def view_cart(self):
    if self.user_id is None:
        messagebox.showerror("Error", "You must be logged in to view your cart.")
        return

    # Create cart window
    self.cart_window = tk.Toplevel(self.root)
    self.cart_window.title("Cart")

    query = """
SELECT Products.Name, Products.Price, Cart.Quantity, Cart.CartID
FROM Cart
JOIN Products ON Cart.ProductID = Products.ProductID
WHERE Cart.UserID = ?
"""

    self.cursor.execute(query, (self.user_id,))

```

```

cart_items = self.cursor.fetchall()

total_amount = 0

for index, item in enumerate(cart_items):
    name, price, quantity, cart_id = item
    total_price = price * quantity
    total_amount += total_price

    ttk.Label(self.cart_window, text=name).grid(row=index, column=0, padx=10,
pady=10)
    ttk.Label(self.cart_window, text="$ {:.2f}".format(price)).grid(row=index, column=1,
padx=10, pady=10)
    ttk.Label(self.cart_window, text=quantity).grid(row=index, column=2, padx=10,
pady=10)
    ttk.Label(self.cart_window, text="$ {:.2f}".format(total_price)).grid(row=index,
column=3, padx=10, pady=10)
    remove_button = ttk.Button(self.cart_window, text="Remove", command=lambda
cid=cart_id: self.remove_from_cart(cid))
    remove_button.grid(row=index, column=4, padx=10, pady=10)

    ttk.Label(self.cart_window, text="Total Amount:
$ {:.2f}".format(total_amount)).grid(row=len(cart_items), column=0, columnspan=4,
padx=10, pady=10)
    checkout_button = ttk.Button(self.cart_window, text="Checkout",
command=self.checkout)
    checkout_button.grid(row=len(cart_items)+1, column=0, columnspan=4, padx=10,
pady=10)

def remove_from_cart(self, cart_id):
    query = "DELETE FROM Cart WHERE CartID = ?"
    self.cursor.execute(query, (cart_id,))

```

```

self.conn.commit()
messagebox.showinfo("Success", "Item removed from cart.")
self.cart_window.destroy()
self.view_cart()

def checkout(self):
    if self.user_id is None:
        messagebox.showerror("Error", "You must be logged in to checkout.")
        return

    query = """
    SELECT Products.ProductID, Products.Price, Cart.Quantity
    FROM Cart
    JOIN Products ON Cart.ProductID = Products.ProductID
    WHERE Cart.UserID = ?
    """
    self.cursor.execute(query, (self.user_id,))
    cart_items = self.cursor.fetchall()

    if not cart_items:
        messagebox.showinfo("Empty Cart", "Your cart is empty.")
        return

    total_amount = sum(item[1] * item[2] for item in cart_items)
    order_date = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    self.cursor.execute("INSERT INTO Orders (UserID, TotalAmount, OrderDate)
VALUES (?, ?, ?)", (self.user_id, total_amount, order_date))
    order_id = self.cursor.lastrowid

    for item in cart_items:
        product_id, price, quantity = item
        self.cursor.execute("INSERT INTO OrderDetails (OrderID, ProductID, Quantity,
Price) VALUES (?, ?, ?, ?)", (order_id, product_id, quantity, price))

```

```
self.cursor.execute("DELETE FROM Cart WHERE UserID = ?", (self.user_id,))
self.conn.commit()
```

```
messagebox.showinfo("Success", "Order placed successfully.")
self.cart_window.destroy()
```

```
if __name__ == "__main__":
    root = tk.Tk()
    app = EcommerceApp(root)
    root.mainloop()
```

## **BACK END:**

```
import sqlite3
```

```
def create_connection():
    conn = sqlite3.connect('ecommerce.db')
    return conn
```

```
def create_tables(conn):
    cursor = conn.cursor()
```

```
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS Users (
        UserID INTEGER PRIMARY KEY AUTOINCREMENT,
        Username TEXT NOT NULL,
        Password TEXT NOT NULL,
        Email TEXT,
        Address TEXT
    )""")
```

```
    cursor.execute("""
```



```
CREATE TABLE IF NOT EXISTS Products (  
    ProductID INTEGER PRIMARY KEY AUTOINCREMENT,  
    Name TEXT NOT NULL,  
    Description TEXT,  
    Price REAL NOT NULL,  
    Stock INTEGER NOT NULL  
)")
```

```
cursor.execute("""
```

```
CREATE TABLE IF NOT EXISTS Orders (  
    OrderID INTEGER PRIMARY KEY AUTOINCREMENT,  
    UserID INTEGER,  
    TotalAmount REAL,  
    OrderDate TEXT,  
    FOREIGN KEY (UserID) REFERENCES Users(UserID)  
)")
```

```
cursor.execute("""
```

```
CREATE TABLE IF NOT EXISTS OrderDetails (  
    OrderDetailID INTEGER PRIMARY KEY AUTOINCREMENT,  
    OrderID INTEGER,  
    ProductID INTEGER,  
    Quantity INTEGER,  
    Price REAL,  
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),  
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)  
)")
```

```
cursor.execute("""
```

```
CREATE TABLE IF NOT EXISTS Cart (  
    CartID INTEGER PRIMARY KEY AUTOINCREMENT,  
    UserID INTEGER,  
    ProductID INTEGER,  
    Quantity INTEGER,
```

```
        FOREIGN KEY (UserID) REFERENCES Users(UserID),
        FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
    )")
```

```
conn.commit()
```

```
def add_initial_products(conn):
```

```
    cursor = conn.cursor()
```

```
    products = [
```

```
        ('Laptop', 'A high performance laptop', 999.99, 10),
```

```
        ('Smartphone', 'Latest model smartphone', 699.99, 20),
```

```
        ('Headphones', 'Noise-cancelling headphones', 199.99, 15),
```

```
        ('Monitor', '27-inch 4K monitor', 299.99, 8),
```

```
        ('Keyboard', 'Mechanical keyboard', 89.99, 25),
```

```
    ]
```

```
    cursor.executemany("""
```

```
INSERT INTO Products (Name, Description, Price, Stock) VALUES (?, ?, ?, ?)
```

```
""", products)
```

```
conn.commit()
```

```
def setup_database():
```

```
    conn = create_connection()
```

```
    create_tables(conn)
```

```
    add_initial_products(conn)
```

```
    conn.close()
```

```
if __name__ == '__main__':
```

```
    setup_database()
```

## 9. RESULTS AND DISCUSSIONS

### 9.1 DATABASE DESIGN

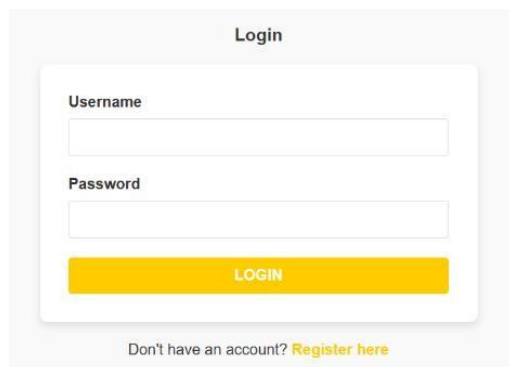
In the context of the E-Commerce Management System, database design is a crucial aspect of system architecture. At the analysis stage, data elements and structures essential for the system's functionality are identified and organized. These components are then structured and integrated to form the foundation of the data storage and retrieval system.

A database serves as a repository for storing and organizing interrelated data with minimal redundancy, allowing for efficient access by multiple users. The primary objective is to facilitate easy, quick, cost-effective, and flexible access to the data for users. Relationships between data elements are established, and unnecessary redundancies are eliminated to optimize storage and enhance system performance.

Normalization plays a vital role in achieving internal consistency, minimizing redundancy, and ensuring data stability within the database. By structuring data into well-defined tables and establishing relationships between them, normalization helps minimize storage requirements, reduce the risk of data inconsistencies, and optimize data updates.

In the E-Commerce Management System, database design is fundamental to its operation. The system comprises various MySQL tables that store essential data related to products, customers, orders, payments, and other entities. Each table is carefully designed to efficiently store and retrieve data, ensuring seamless operation and scalability of the e-commerce platform

#### 1. Customer Home page



The image shows a login form titled "Login". It contains two input fields: "Username" and "Password". Below the password field is a yellow button labeled "LOGIN". At the bottom of the form, there is a link that says "Don't have an account? [Register here](#)".

## 2. Customer Login Page

## Register

Username

Password

Email

Address

REGISTER

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

### 3. Product List

[illegible]

#### 4. List of Items in Cart

## E-commerce System

Products

Cart

Your Cart

• Keyboard \$89.99 x 1 = \$89.99

Remove

Total: **\$89.99**

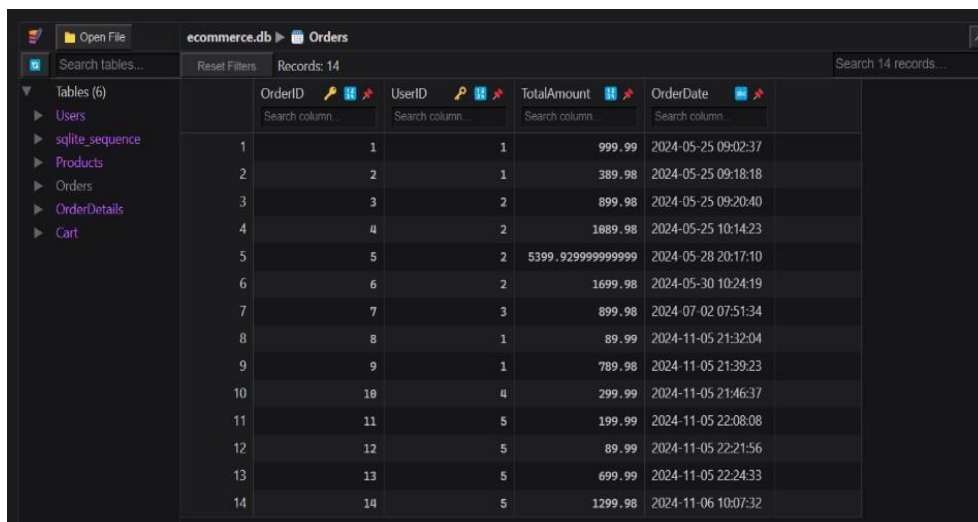
Checkout

© 2024 E-commerce System. All Rights Reserved

## 5. Order Status

Checkout Complete! Your order has been placed.

## 9.2 OUTPUT:



	OrderID	UserID	TotalAmount	OrderDate
1	1	1	999.99	2024-05-25 09:02:37
2	2	2	389.98	2024-05-25 09:18:18
3	3	3	899.98	2024-05-25 09:20:40
4	4	4	1089.98	2024-05-25 10:14:23
5	5	2	5399.929999999999	2024-05-28 20:17:10
6	6	2	1699.98	2024-05-30 10:24:19
7	7	3	899.98	2024-07-02 07:51:34
8	8	1	89.99	2024-11-05 21:32:04
9	9	1	789.98	2024-11-05 21:39:23
10	10	4	299.99	2024-11-05 21:46:37
11	11	5	199.99	2024-11-05 22:08:08
12	12	5	89.99	2024-11-05 22:21:56
13	13	5	699.99	2024-11-05 22:24:33
14	14	5	1299.98	2024-11-06 10:07:32

## 10. CONCLUSION

The E-Commerce Management System project represents a efforts aimed at creating a robust, user-friendly, and feature-rich platform for businesses to establish and manage their online presence. Throughout the development journey, we prioritized functionality, usability, and scalability to ensure that our system meets the diverse needs of businesses and consumers in the ever-evolving landscape of online commerce.

### **Addressing Market Needs:**

In today's digital era, the demand for efficient e-commerce platforms is higher than ever. Recognizing this need, our project set out to provide a comprehensive solution that empowers businesses to thrive in the competitive online marketplace. By leveraging the power of Python, Tkinter, and SQLite, we aimed to develop a platform that streamlines business operations and enhances the shopping experience for customers.

### **Core Features and Functionality:**

Our E-Commerce Management System boasts a wide range of features designed to meet the needs of businesses and consumers alike. From intuitive product management tools and secure payment processing to personalized recommendations and seamless order fulfillment, every aspect of the system has been carefully crafted to enhance efficiency and drive growth. Additionally, robust security measures and user-friendly interfaces ensure a safe and enjoyable shopping experience for customers.

### **Development Process and Methodology:**

The development process followed a systematic approach, beginning with thorough requirements analysis and design planning. We employed industry best practices in software development, including modular design, code reusability, and rigorous testing, to ensure the reliability and scalability of the system. Leveraging the capabilities of Python for backend development, Tkinter for frontend design, and SQLite for database management, we created a cohesive and integrated solution that meets the highest standards of quality and performance.

### **Impact and Benefits:**

The impact of the E-Commerce Management System project extends beyond the realm of business operations. For businesses, the system offers a cost-effective platform to showcase their products, expand their customer base, and optimize their sales strategies. For consumers, it provides a convenient and secure way to browse, shop, and interact with their favorite brands. By facilitating seamless transactions and personalized experiences, our system fosters trust and loyalty among both businesses and consumers.

### **Future Enhancements and Growth Opportunities:**

While our current implementation meets the core requirements of an e-commerce platform, there are ample opportunities for future enhancements and growth. Integration with third-party services, implementation of advanced analytics tools, and expansion into new markets are just a few areas where the system could evolve and improve over time. As technology continues to advance, our system stands ready to adapt and innovate, ensuring continued success and relevance in the dynamic world of online commerce.

### **Conclusion:**

In conclusion, the E-Commerce Management System project represents a significant milestone in the journey towards creating a more efficient, inclusive, and accessible online marketplace. By leveraging the power of Python, Tkinter, and SQLite, we have developed a versatile and scalable platform that empowers businesses to thrive in the digital age. As we look towards the future, we remain committed to innovation, excellence, and continuous improvement, ensuring that our system remains at the forefront of e-commerce technology for years to come.

## 11. REFERENCES

### **References for Python and Tkinter:**

- ▮ Python Official Documentation: <https://docs.python.org/3/>
- ▮ Tkinter Official Documentation: <https://docs.python.org/3/library/tkinter.html>
- ▮ "Python GUI Programming with Tkinter" by Alan D. Moore:  
<https://www.amazon.com/Python-GUI-Programming-Tkinter-Moore/dp/1788835883>

### **For SQLite:**

- ▮ SQLite Documentation: <https://www.sqlite.org/docs.html>
- ▮ "Using SQLite" by Jay A. Kreibich: <https://www.amazon.com/Using-SQLite-Jay-A-Kreibich/dp/1449394592>
- ▮ "SQLite Database Programming for Xamarin: Cross-platform C# Database Development for iOS and Android using SQLite.XM" by Jesse Liberty: <https://www.amazon.com/SQLite-Database-Programming-Xamarin-Cross-platform/dp/1484224637>
- ▮ "SQLite Database Programming for C/C++ Programmers" by Dr. John Iovine:  
<https://www.amazon.com/SQLite-Database-Programming-Programmers-Iovine/dp/1519619777>
- ▮ "The Definitive Guide to SQLite" by Mike Owens: <https://www.amazon.com/Definitive-Guide-SQLite-2nd/dp/1484220038>
- ▮ "SQLite Cookbook" by Jay A. Kreibich: <https://www.amazon.com/SQLite-Cookbook-Jay-A-Kreibich/dp/1449316742>