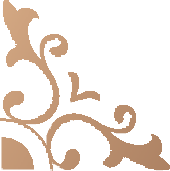
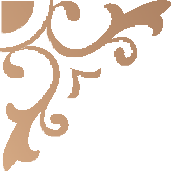
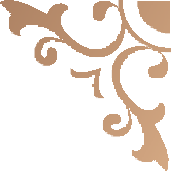
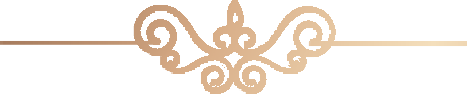
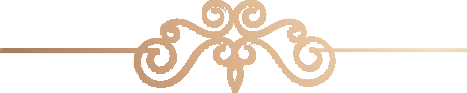
FACULTY OF COMPUTERS SCIENCE AND INFORMATION TECHNOLOGY DAMNHOUR UNIVERSITY



EgyMart || Misrify

GRADUATION PROJECT DOCUMENTATION

Ahmed Reda Algrgawy

Mostafa Hassan Negm

Zeyad Magdy Elkhamary

Ali Ayman El-Beltagy

Mohamed Hisham Siam

Sandreen Bassam Kotp

Reem Elsayed Ghareeb

Sara Abdelwahed Elkholy

SUPERVISOR: Dr/ Nora Shoaip

Graduation project

**2024-2025**

**EgyMart || Misrify**

Faculty of Computers and Informatics Damanhour University

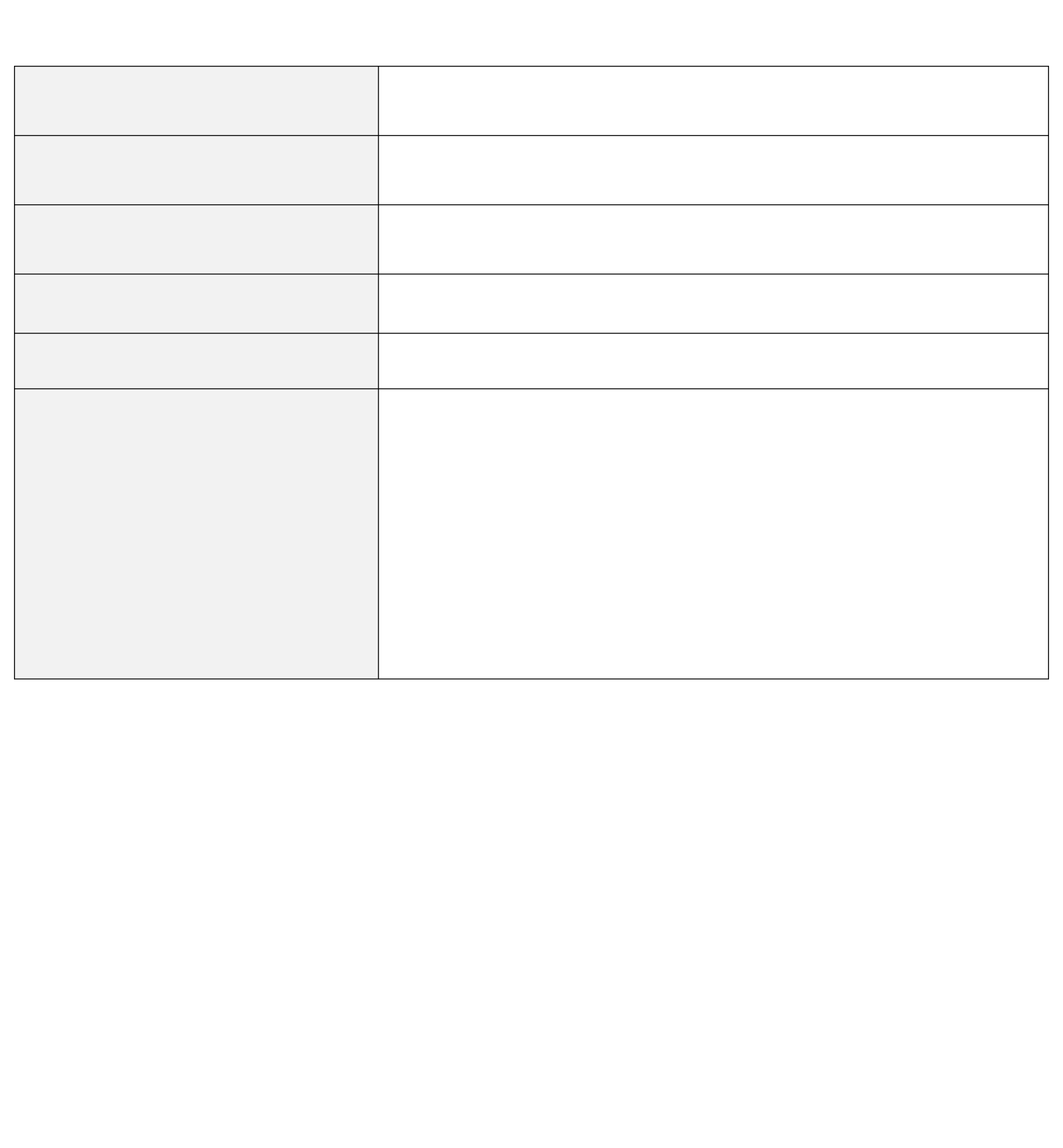
CS, IT, IS Department

Supervised by:

***Dr/ Nora Shoaip***

**Team Members:**

* Ahmed Reda Algrgawy
* Mostafa Hassan Negm
* Zeyad Magdy Elkhamary
* Ali Ayman El-Beltagy
* Mohamed Hisham Siam
* Sandreen Bassam Kotp
* Reem Elsayed Ghareeb
* Sara Abdelwahed Elkholy



**Graduation Project - English Abstract**

Project Code:

Project Title (in English): EgyMart || Misirify

Project Title (in Arabic): تعلمها

Scientific Department: CS, IT

Supervisor(s): Dr. Noura Shoaib - Eng. Emad Tawfeek

Project Team:

1. Ahmed Reda Algrgawy
2. Mostafa Hassan Negm
3. Zeyad Magdy Elkhamary
4. Ali Ayman El-Beltagy
5. Mohamed Hisham Siam
6. Sandreen Bassam Kotp
7. Reem Elsayed Ghareeb
8. Sara Abdelwahed Elkholy

**Project Abstract**

This project seeks to establish a comprehensive online marketplace exclusively dedicated to Egyptian brands and stores, addressing the challenges local businesses face in reaching wider audiences and promoting their products. In response to economic downturns and a growing demand for Egyptian alternatives during national campaigns supporting local products, the platform aims to empower Egyptian businesses, promote national pride, and provide consumers with high-quality, authentic products, The platform will feature core functionalities such as user-friendly registration, AI-powered search, secure payment options, and brand-specific management systems. It will provide a unique shopping experience through exclusive discounts, advanced filtering, and mobile-friendly design, targeting a diverse audience of Egyptian consumers, expatriates, and tourists. By leveraging AI and fostering community engagement, the marketplace aspires to drive economic growth, support local artisans, and celebrate Egyptian culture and craftsmanship, through this initiative, the project underscores its commitment to supporting local communities, promoting high-quality products, and creating a vibrant ecosystem where Egyptian brands and consumers thrive together.

**الملخص العربي لمشروع التخرج**

#### كود المشروع:

EgyMart - Misrify

**تعلمها**

#### عنوان المشروع باللغة العربية:

**عنوان المشروع باللغة الإنجليزية:**

#### القسم العلمي: علوم الحاسب – تكنولوجيا المعلومات

**اسم المشرف على المشروع: د. نورا شعيب – د. عماد توفيق**

**احمد رضا الجرجاوي**

**مصطفي حسن نجم**

**زياد مجدي الخماري**

**علي ايمن البلتاجي**

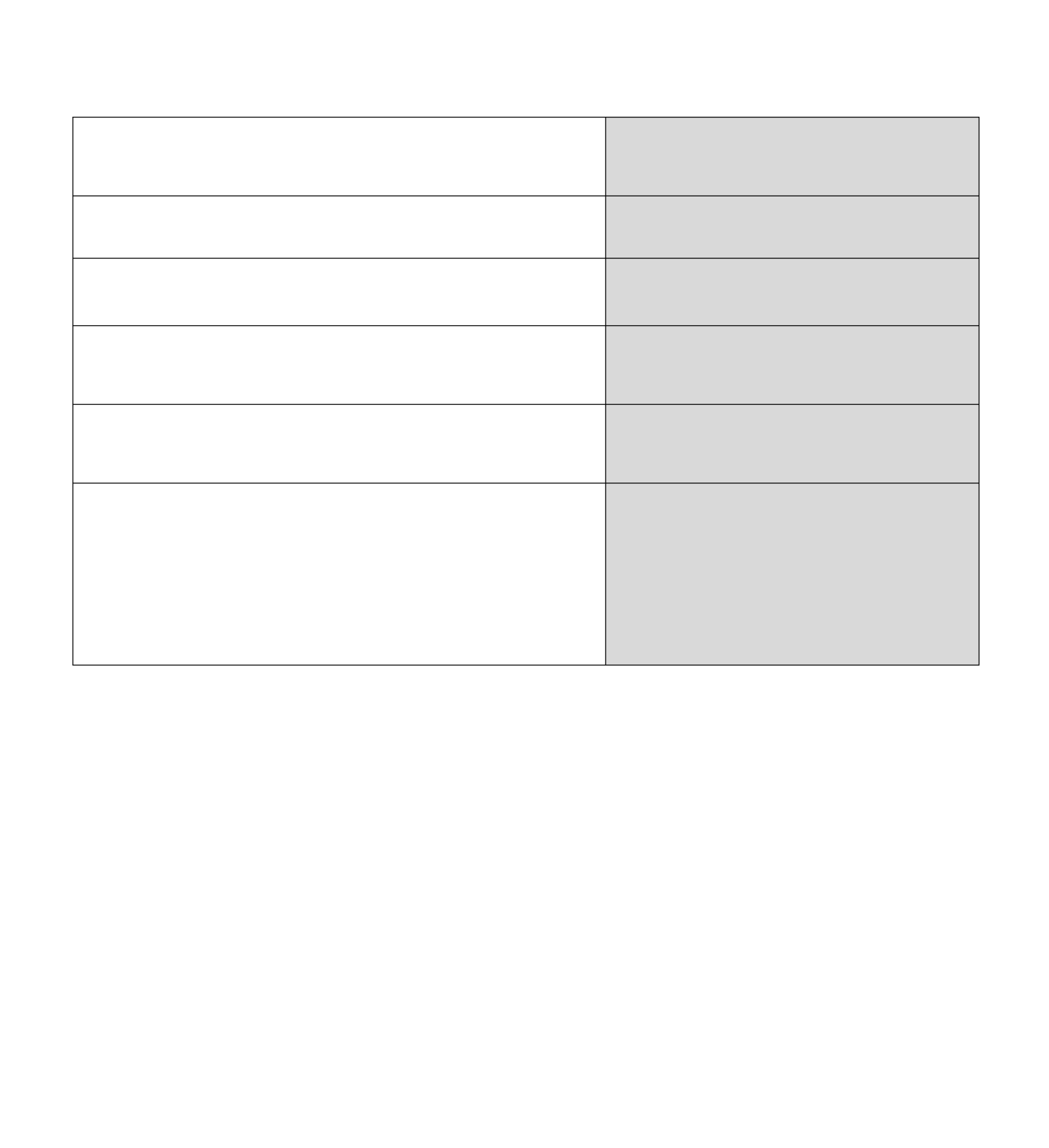
**محمد هشام صيام**

**ساندرين بسام قطب**

**ريم السيد غريب**

**ساره عبد الواحد الخولي**

**أسماء فريق العمل من الطلاب:**



**ملخص المشروع Abstract Project**

يسعى هذا المشروع إلى إنشاء سوق إلكتروني شامل مخصص حصريًا للعلامات التجارية والمتاجر المصرية، لمعالجة التحديات التي تواجهها الشركات المحلية في الوصول إلى جمهور أوسع والترويج لمنتجاتها. استجابةً للتحديات الاقتصادية وزيادة الطلب على البدائل المصرية خلال الحملات الوطنية لدعم المنتجات المحلية، يهدف المنصّة إلى تمكين الشركات المصرية، وتعزيز الفخر الوطني، وتوفير منتجات عالية الجودة وأصيلة للمستهلكين، ستتضمن المنصة وظائف أساسية مثل تسجيل سهل الاستخدام، وبحث مدعوم بالذكاء الاصطناعي، وخيارات دفع آمنة، وأنظمة إدارة مخصصة للعلامات التجارية. كما ستقدم تجربة تسوق فريدة من خلال خصومات حصرية، ومرشحات متقدمة، وتصميم متوافق مع الأجهزة المحمولة، مع استهداف جمهور متنوع من المستهلكين المصريين والمغتربين والسياح. وبالاعتماد على الذكاء الاصطناعي وتعزيز المشاركة المجتمعية، تطمح المنصة إلى تعزيز النمو الاقتصادي، ودعم الحرفيين المحليين، والاحتفاء بالثقافة والحرف المصرية، من خلال هذه المبادرة، يؤكد المشروع التزامه بدعم المجتمعات المحلية، وتعزيز المنتجات عالية الجودة، وخلق نظام حيوي مزدهر حيث تزدهر العلامات التجارية المصرية والمستهلكون معًا.

**نموذج تقييم ومناقشة مشروع التخرج**

#### EgyMart || Misrify

* Computer Science **○** Information Systems **○** Information technology **○** Multimedia

#### اسم المشروع باللغه الانجليزيه

#### القسم العلمي

**اجمالي**

**الدرجات**

**40 درجة**

**استيعاب وتمكن الطالب من المشروع والرد على استفسارات وأسئلة اللجنة**

**25 درجة**

**Presentation**

**) درجة الإلمام، الوضوح، اإلتقان(.**

**5 درجات**

**Documentation**

**) التنظيم العام، المحتوى، مستوى الكتابة، األسلوب، مراعاة الاخلاقيات (**

**5 درجات**

**هل المشروع يحقق الهدف المطلوب؟**

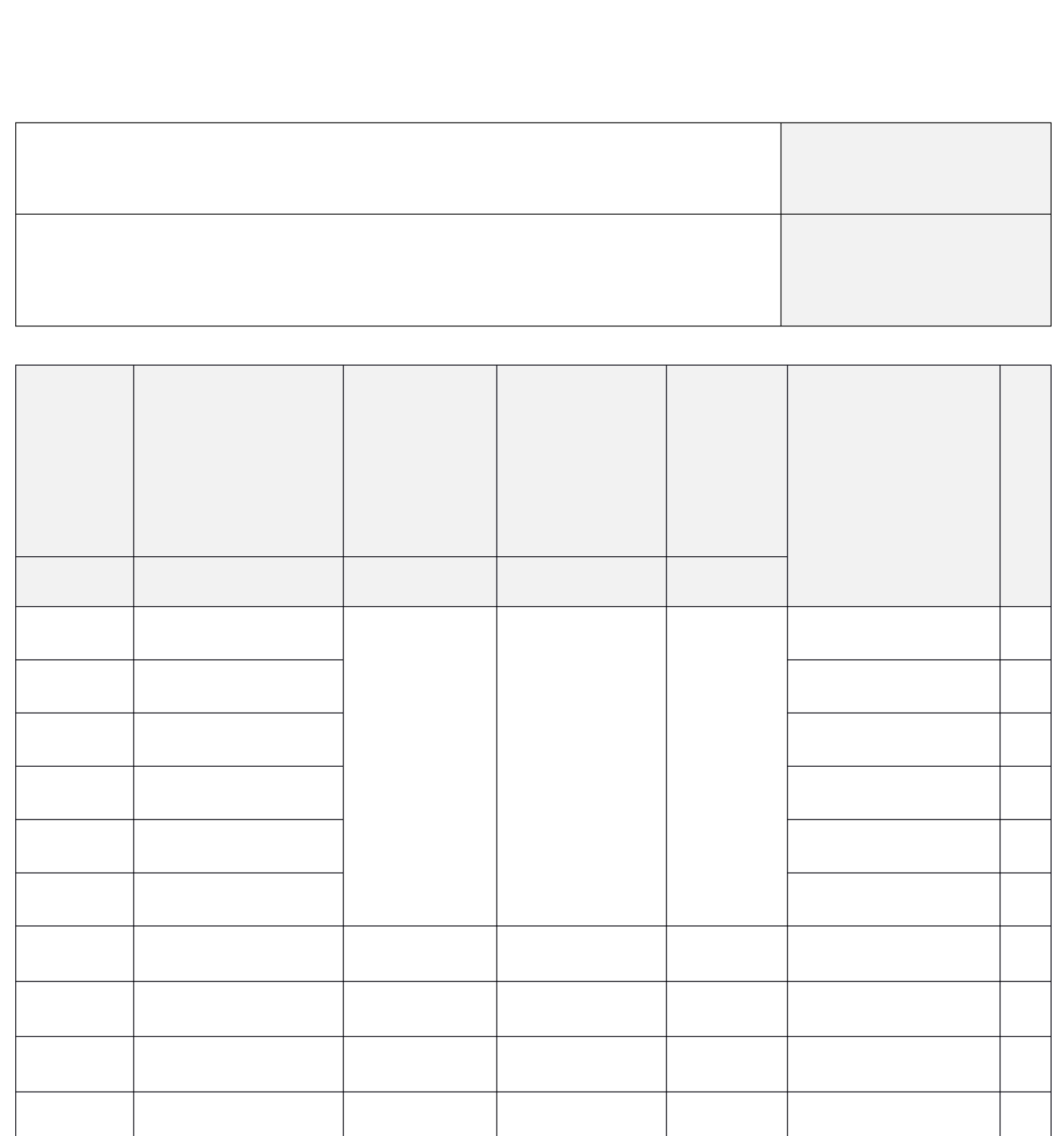
**5 درجات**

**م اسم الطالب**

1 أحمد عبدالعزيز الهامي 2 حسين محمد عبدالرازق 3 ندي عبدالناصر محمد 4 خديجه احمد السيد

5 محمد عاطف سعيد

6 محمد علي السمخراطي



7

8

9

10

4

نموذج وضع درجات أعمال السنة لمشروع التخرج (من قبل المشرف)

#### Opi Se

|  |  |  |  |
| --- | --- | --- | --- |
| * Computer Science **○** Information Systems **○** Information | | | |
| technology | * Multimedia |  |  |

**اسم المشروع باللغة**

#### االنجليزية

**القسم العلمي**

**اجمالي درجات أعمال السنة**

**60 درجة**

**قدرة الطالب على العمل ضمن فريق عمل**

**10 درجة**

**قدرة الطالب على تنفيذ المهام**

**المكلف بها**

**20 درجة**

**استيعاب وتمكن الطالب من األدوات التي استخدمت في**

**تطوير المشروع**

**10درجة**

**المشاركة في النقاشات واالجابة على األسئلة المطروحة اثناء اللقاءات الدورية**

**10 درجة**

**مدى التزام الطالب بحضور اللقاءات والمتابعة**

**المستمرة مـع المـشرف**

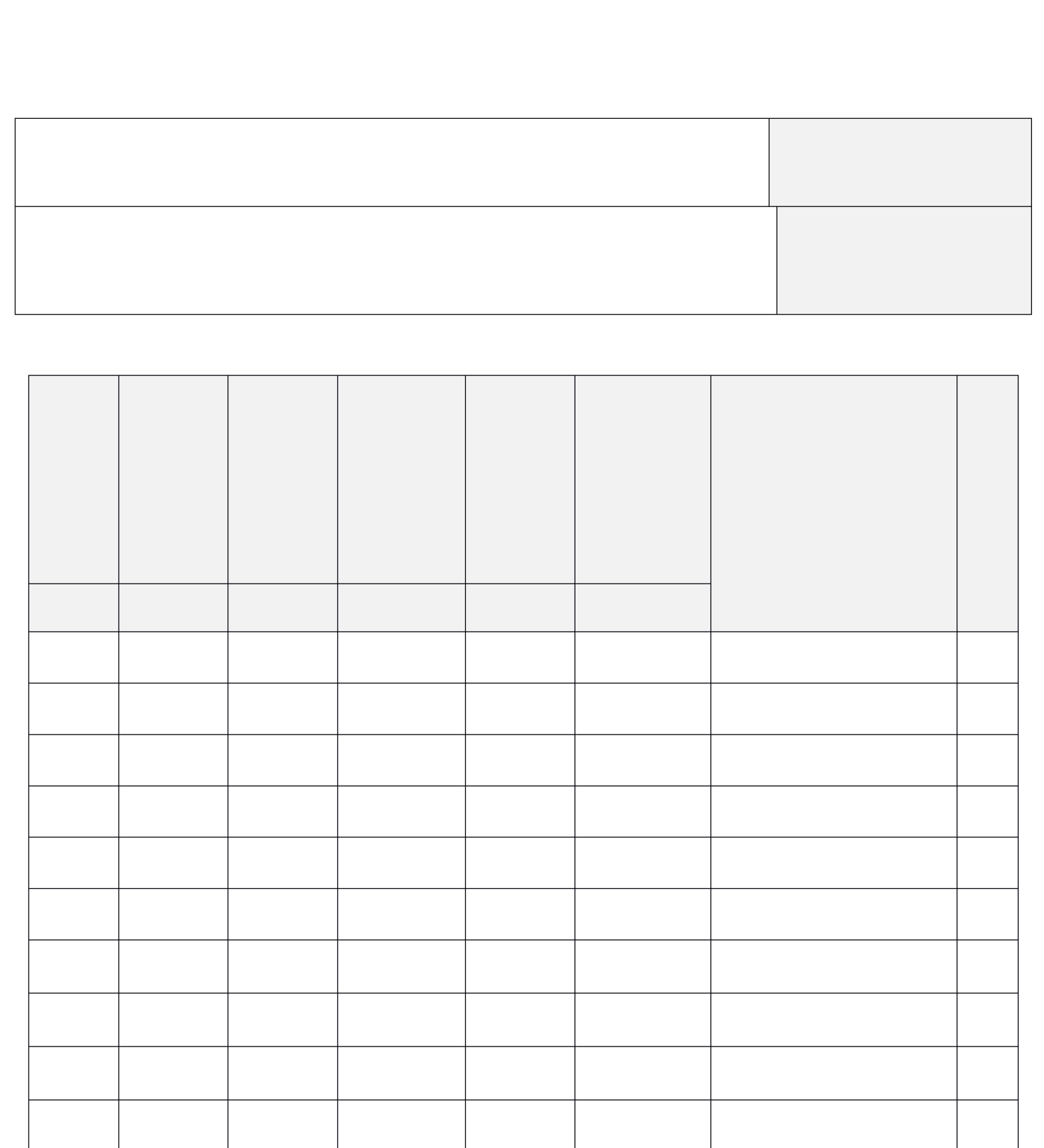
**10 درجة**

**م اسم الطالب**

1 أحمد عبدالعزيز الهامي 2 حسين محمد عبدالرازق 3 ندي عبدالناصر محمد 4 خديجه احمد السيد

5 محمد عاطف سعيد

6 محمد علي السمخراطي



7

8

9

10

5

Contents

[Acknowledgment 11](#_bookmark0)

[Abstraction 12](#_bookmark1)

[Chapter 1 13](#_bookmark2)

* 1. [Introduction 13](#_bookmark3)
     1. [Background 14](#_bookmark4)
     2. [Motivations and problem statement 14](#_bookmark5)
     3. [Objectives 15](#_bookmark6)
        1. [Finding Appropriate Partner 15](#_bookmark7)
        2. [Mental-health support 15](#_bookmark8)
        3. [Interactive learning 15](#_bookmark9)
        4. [Management tools 15](#_bookmark10)
        5. [Study tracking 15](#_bookmark11)
        6. [Gamification techniques 16](#_bookmark12)
        7. [Study competition 16](#_bookmark13)
        8. [Mutual respect 16](#_bookmark14)
        9. [Finding an Appropriate mentor 16](#_bookmark15)
     4. [Organization Profile 16](#_bookmark16)
        1. [Scope of experience includes 16](#_bookmark17)
        2. [This project depends on 17](#_bookmark18)
        3. [Services 17](#_bookmark19)
     5. [Purpose of the project 17](#_bookmark20)

[Chapter 2 18](#_bookmark21)

* 1. [Competitors 18](#_bookmark22)
     1. [Related Work and Competitors 19](#_bookmark23)
        1. [Traditional Study Platforms 19](#_bookmark24)
        2. [Peer Learning Networks 19](#_bookmark25)
        3. [Mental Health and Well-Being Focused Apps 20](#_bookmark26)
        4. [Time Management and Productivity Tools 20](#_bookmark27)
        5. [Gamification in Education 21](#_bookmark28)
        6. [Communication and Collaboration Platforms 21](#_bookmark29)
        7. [Mentorship Programs 22](#_bookmark30)
     2. [Social Media and Alternative Study Methods 22](#_bookmark31)
     3. [Social Media Platforms 23](#_bookmark32)
     4. [YouTube "Study with Me" Livestreams 24](#_bookmark33)

[Chapter 3 26](#_bookmark34)

* 1. [Website Models 26](#_bookmark35)
     1. [Recommendation System 27](#_bookmark36)
        1. [Introduction 27](#_bookmark37)
        2. [The Power of Recommendations 27](#_bookmark38)
        3. [Types of Recommendation Systems 28](#_bookmark39)
        4. [Hybrid Recommendation System 29](#_bookmark40)
        5. [Why Hybrid Recommender System? 30](#_bookmark41)
           1. [Content-based Filtering Drawbacks 30](#_bookmark42)

[3.1.5.2. Collaborative filtering Drawbacks 31](#_bookmark43)

* + - 1. [Model Design and Architecture 32](#_bookmark44)
         1. [Components of Content-Based Recommender 32](#_bookmark45)
      2. [Content Cleaner (NLP Component) 33](#_bookmark46)
      3. [Filtering Component 34](#_bookmark47)
      4. [Components of Neural Collaborative Filtering 34](#_bookmark48)

[3.1.7.1. Embeddings 34](#_bookmark49)

* + - 1. [Stochastic Gradient Descent (SGD) 35](#_bookmark50)
      2. [Filtering Component 37](#_bookmark51)
      3. [Hybrid-Recommender System (Put It All Together) 37](#_bookmark52)
      4. [Evaluation and Performance 38](#_bookmark53)

[3.1.9.1. Refinement Strategies 38](#_bookmark54)

* + - 1. [System Architecture 39](#_bookmark55)
    1. [Mental Health Model 42](#_bookmark56)
       1. [Introduction 42](#_bookmark57)
       2. [Mental Health Score Prediction 42](#_bookmark58)
          1. [Dataset 42](#_bookmark59)
       3. [System methodologies 42](#_bookmark60)
       4. [Data pre-processing 42](#_bookmark61)
       5. [Model Implementation 43](#_bookmark62)
       6. [Result 43](#_bookmark63)
       7. [Limitations 43](#_bookmark64)
       8. [Mental Health Screening 43](#_bookmark65)
       9. [Dataset 43](#_bookmark66)
       10. [Data Processing 46](#_bookmark67)

[3.2.3.4. Result 47](#_bookmark68)

* + 1. [User Dashboard 47](#_bookmark69)
       1. [Dashboard Functionality 47](#_bookmark70)
       2. [Data Source 48](#_bookmark71)
       3. [Benefits 48](#_bookmark72)
       4. [Disclaimer 48](#_bookmark73)

[Chapter 4 49](#_bookmark74)

* 1. [System Analysis 49](#_bookmark75)
     1. [Introduction 50](#_bookmark76)
     2. [System Requirements 50](#_bookmark77)
        1. [Functional Requirements 50](#_bookmark78)
        2. [Registration and Login 50](#_bookmark79)
        3. [Partner Matching 50](#_bookmark80)
        4. [Chat 51](#_bookmark81)
        5. [Meeting 51](#_bookmark82)

[4.2.1.6. Notes 51](#_bookmark83)

* + - 1. [Mental Health 51](#_bookmark84)
      2. [Quizzes and Assignments 51](#_bookmark85)
      3. [Non-Functional Requirements 52](#_bookmark86)
      4. [Usability: 52](#_bookmark87)
      5. [Reliability 52](#_bookmark88)
      6. [Maintainability 52](#_bookmark89)
      7. [Adaptability 52](#_bookmark90)
      8. [Portability 53](#_bookmark91)
      9. [Interoperability 53](#_bookmark92)
      10. [User Requirements 53](#_bookmark93)
  1. [Development Methodology 53](#_bookmark94)
     1. [Use Case Diagram 53](#_bookmark95)
     2. [Class Diagram 54](#_bookmark96)
     3. [Sequence Diagram 55](#_bookmark97)
     4. [Entity Relation Diagram (ERD) 57](#_bookmark98)

[Chapter 5 59](#_bookmark99)

* 1. [System Design 59](#_bookmark100)
     1. [Software Website Design 60](#_bookmark101)
     2. [UI in System Design 60](#_bookmark102)
     3. [User Interface 60](#_bookmark103)
     4. [Mobile App Screens 76](#_bookmark104)

[Chapter 6 81](#_bookmark105)

* 1. [Front-End Implementation 82](#_bookmark106)
     1. [Introduction 82](#_bookmark107)
     2. [Technology Stack 82](#_bookmark108)
        1. [Core Development Tools 82](#_bookmark109)
        2. [User Interface and Styling 83](#_bookmark110)
        3. [Form Management and Validation 83](#_bookmark111)
        4. [Real-Time Communication and Interaction 84](#_bookmark112)
        5. [Additional Libraries 84](#_bookmark113)
        6. [Deployment 84](#_bookmark114)
     3. [Project Structure 84](#_bookmark115)
        1. [Page-Based Structure 86](#_bookmark116)
     4. [Architectural Patterns 87](#_bookmark117)

[6.1.4.1 Presentational and Container Design Pattern 87](#_bookmark118)

[6.1.4.2. Facade Design Pattern 91](#_bookmark119)

* + 1. [State Management 93](#_bookmark120)
    2. [Routing 94](#_bookmark121)
    3. [Styling 94](#_bookmark122)
    4. [API Integration 94](#_bookmark123)

[6.1.8.1. useHttp custom hook 95](#_bookmark124)

* + 1. [Performance Optimization 95](#_bookmark125)
    2. [Deployment 95](#_bookmark126)
  1. [Back-End Implementation 96](#_bookmark127)
     1. [Backend Technologies 96](#_bookmark128)
        1. [JavaScript (JS) 96](#_bookmark129)
        2. [Node.js 96](#_bookmark130)
        3. [Express.js (Express Framework) 96](#_bookmark131)
        4. [Mongoose ORM (Object-Relational Mapper) 96](#_bookmark132)
     2. [Backend Architecture & Design 97](#_bookmark133)
        1. [Repository Pattern 97](#_bookmark134)
        2. [REST (Representational State Transfer) 98](#_bookmark135)
        3. [Socket.IO: Enabling Real-time Communication 100](#_bookmark136)
     3. [Backend components and resources 103](#_bookmark137)
        1. [Backend Server (Render web services) 103](#_bookmark138)
        2. [Database (MongoDB) 103](#_bookmark139)
        3. [Caching (AWS Redis) 105](#_bookmark140)
        4. [Media server (Cloudinary cloud service) 106](#_bookmark141)
        5. [Pushing-server (Firebase) 107](#_bookmark142)
     4. [Unit Testing 108](#_bookmark143)
        1. [Unit Testing tools 108](#_bookmark144)
     5. [Deployment Continuous Integration and Deployment (CI/CD) 111](#_bookmark145)
        1. [Continuous Integration (CI) 111](#_bookmark146)
        2. [Continuous Deployment (CD) 111](#_bookmark147)
  2. [Mobile App Implementation 112](#_bookmark148)
     1. [Tools Used 112](#_bookmark149)
     2. [Packages 112](#_bookmark150)
     3. [Project Architecture 113](#_bookmark151)
     4. [Core Folder 113](#_bookmark152)
     5. [Feature Structure 115](#_bookmark153)
     6. [Top Components 115](#_bookmark154)

[Chapter 7 119](#_bookmark155)

1. [Conclusion 119](#_bookmark156)
   1. [Conclusion 120](#_bookmark157)
   2. [Benefits 120](#_bookmark158)
   3. [Future works 121](#_bookmark159)
   4. [References 122](#_bookmark160)

# Acknowledgment

First and foremost, we would like to extend my deepest gratitude to my supervisor, ***Dr. Nora Shoip***, for her exceptional guidance, unwavering support, and invaluable advice throughout the course of this project. Her expertise and insights were a constant source of inspiration, and her encouragement helped us overcome challenges and stay motivated. Without her mentorship, this project would not have reached its current level of success.

I am equally grateful to ***Eng. Emad Tawfeek****,* our assistant supervisor, for his dedication, patience, and constructive feedback. His continuous assistance and attention to detail played a significant role in shaping this work and ensuring its completion to a high standard. We greatly appreciate his willingness to share his knowledge and time, which were crucial to our learning and growth throughout this journey.

We also want to thank our professors, colleagues, and friends who offered their support and encouragement along the way. Their belief in our abilities and their thoughtful words kept us going during the more challenging moments.

This project is the result of collective efforts, and we are deeply thankful to everyone who contributed, directly or indirectly, to its success.

# Abstraction

This project seeks to establish a comprehensive online marketplace exclusively dedicated to Egyptian brands and stores, addressing the challenges local businesses face in reaching wider audiences and promoting their products. In response to economic downturns and a growing demand for Egyptian alternatives during national campaigns supporting local products, the platform aims to empower Egyptian businesses, promote national pride, and provide consumers with high-quality, authentic products.

The platform will feature core functionalities such as user-friendly registration, AI-powered search, secure payment options, and brand-specific management systems. It will provide a unique shopping experience through exclusive discounts, advanced filtering, and mobile-friendly design, targeting a diverse audience of Egyptian consumers, expatriates, and tourists. By leveraging AI and fostering community engagement, the marketplace aspires to drive economic growth, support local artisans, and celebrate Egyptian culture and craftsmanship.

Through this initiative, the project underscores its commitment to supporting local communities, promoting high-quality products, and creating a vibrant ecosystem where Egyptian brands and consumers thrive together.

# Chapter 1



## Introduction

### 1.1 Motivation

The motivation behind this project stems from a profound desire to address the challenges faced by Egyptian brands and local businesses in gaining visibility and effectively reaching their target audience. Recent global events, such as the economic challenges posed by the COVID-19 pandemic, the Ukraine-Russia conflict, and regional tensions, have highlighted the need to bolster national economies and prioritize local production. In Egypt, these challenges were further compounded by campaigns urging consumers to support local products as an alternative to international brands, This movement brought to light the existence of many high-quality Egyptian products that remain underrepresented or entirely absent in both offline and online markets. These products often come from talented artisans and dedicated businesses that lack the resources to effectively market themselves on a broader scale. As a result, the potential for these products to contribute to the local economy and national identity remains largely untapped, With this project, we aim to bridge the gap between Egyptian businesses and consumers by creating an inclusive and vibrant online marketplace dedicated to showcasing the richness of Egyptian craftsmanship. This platform not only seeks to enhance the visibility of local brands but also to foster a renewed sense of pride in Egyptian culture and heritage, Additionally, by integrating modern technologies such as AI-driven search and personalized shopping experiences, the project aspires to create a cutting-edge platform that caters to the evolving needs of today’s consumers. By doing so, we aim to make shopping for Egyptian products not only accessible but also enjoyable, convenient, and rewarding, Ultimately, this project is driven by a vision to empower local businesses, stimulate economic growth, and celebrate the ingenuity and craftsmanship of Egyptian products. It is an initiative rooted in community engagement, national pride, and the desire to build a sustainable ecosystem where Egyptian brands and consumers can thrive together.

### 1.2 Project Description

This project aims to establish a comprehensive online marketplace exclusively dedicated to Egyptian brands and stores. The platform will serve as a centralized hub for local businesses to showcase their products and reach a wider customer base, while also offering consumers a convenient and unique shopping experience.

### 1.3 Problem Statement

We have all noticed in the recent period the decline in the economic level of Egypt, especially after the events of the Corona virus and the events of the Ukraine-Russia war, and the weakness of Egyptian products appeared even more during the Israeli aggression against Palestine. At that time, campaigns appeared to boycott non-Egyptian products that support the occupier entity, we the Egyptians began to search for Egyptian alternatives. We discovered that there were many powerful Egyptian products that were unknown to us and were completely marginalized, even though a large portion of these products were of high quality, we did not hear about them or see them in any offline nor online markets that’s because Egyptian brands and stores face challenges in reaching a wider customer base and effectively promoting their products.

### 1.4 Project Goals

* Create a thriving online marketplace: Establish a vibrant and sustainable platform that connects Egyptian brands and stores with consumers.
* Helps local businesses reach a wider customer base and increase their sales throw platform for local businesses to grow and thrive.
* Drive economic growth: Contribute to the local economy by stimulating consumer spending and supporting businesses.
* Foster community: Create a sense of community and connection among Egyptian brands, stores, and consumers.
* Promote Egyptian Brands To increase visibility and sales for local businesses by providing them with a dedicated platform.
* Offer a Unique Shopping Experience: To create a distinctive marketplace focused solely on Egyptian products, catering to the preferences of Egyptian consumers.
* Leverage AI for Enhanced Search: To implement an advanced search feature powered by artificial intelligence, allowing users to find products similar to images they upload.
* Provide a User-Friendly Interface: To design an intuitive and easy-to-navigate platform for both users and sellers, offer a user-friendly and enjoyable shopping experience for consumers.
* Enable Brand Partnerships: To offer brands the opportunity to have their own dedicated admin accounts, allowing them to manage their product listings and promotions effectively.
* Offer Exclusive Discounts: To provide users with exclusive discounts and promotions from participating stores.

### 1.5 Project Values

* **Supporting Local Communities:** Contributing to the economic development of local communities.
* **Celebrating Egyptian Culture:** Showcasing the beauty and diversity of Egyptian heritage.
* **Promoting Local Brands:** Empowering Egyptian businesses and fostering a sense of pride.
* **High-Quality Products:** Offering products made with the finest materials and attention to detail.
* **Supporting Local Artisans:** Collaborating with skilled artisans to create unique and authentic products.
* **Exceptional Service:** Providing excellent customer service and support.
* **Trust and Transparency:** Building trust through open communication and honesty.
* **Creating a Sense of Community:** Fostering a sense of belonging and connection among customers and merchants.
* **Students Support:** the platform will be helping the students by allowing them to buy their own products, startups ’products across the platform and also giving them special points according to there registration email.

### 1.6 Project Contribution

* Economic Development
  + **Job Creation:** Creating jobs in various sectors, including e-commerce, logistics, and customer service.
  + **Boosting Local Economy:** Stimulating local economic activity by increasing consumer spending and supporting local businesses.
  + **Export Opportunities:** Facilitating the export of Egyptian products to international markets.
* Social Impact
  + **Empowering Local Businesses:** Providing a platform for local businesses to reach a wider audience and grow their customer base.
  + **Preserving Traditional Craftsmanship:** Supporting local artisans and preserving traditional skills.
* Technological Advancement
  + **Digital Transformation:** Driving the digital transformation of the Egyptian retail industry.
  + **Innovation:** Encouraging innovation and the development of new technologies in the e-commerce sector.
  + **Improving Efficiency:** Improving the efficiency of supply chains and logistics operations.
* Global Impact
  + **Positioning Egypt as a Global Player:** Positioning Egypt as a global player in the e-commerce industry.
  + **Attracting Foreign Investment:** Attracting foreign investment in the Egyptian e-commerce sector.
  + **Increasing Egypt's Visibility:** Increasing Egypt's visibility on the international stage.

### 1.6 Project Features

#### 1.6.1 Main Features

* **User Management**
* **Role-Based Access Control (RBAC)**
* **Authentication & Authorization**
* **Profile Management**
* **Notification System**
* **Product Management**
* **Advanced Search and Filtering**
* **Shopping Cart**
* **Wishlist**
* **Checkout and Shipping Information**
* **Payment Options**
* **Reviews and Ratings**
* **Platform Security**
* **User Experience Enhancements**

#### Our Features

* **Powered-Ai Recommendation System according to users’ cart, whilst and orders.**
* **Support Students’ Businesses**
* **Helping students start their own businesses according to the instructions and guidelines of our university.**
* **Customer Support Chatbot**
* **Exchange Points for Discounts Promocodes and give any user with our university emails several points for free.**
* **Communication between Users and Merchants through comments on the products.**
* **Platform Analytics for Different Types of Users**
* **Give a wide range of attention to the Egyptians products and brands specially helping the small business to grow.**

#### 1.6.3 Feature Matrix

# Chapter 2



## Market Analysis and Strategic Development

### 2.1 Market Research

#### 2.1.1 Key Findings

* **Growing E-commerce Penetration:** Egypt's e-commerce market is rapidly expanding, driven by increased internet access and smartphone adoption.
* **Preference for Local Brands:** Egyptian consumers increasingly favor local brands for their authenticity, quality, and support for the domestic economy.
* **Challenges for Local Brands:** Local brands face challenges such as limited brand awareness, lack of online presence, and competition from international brands.
* **Opportunities for Growth:** E-commerce platforms offer significant opportunities for local brands to reach a wider audience, build brand recognition, and increase sales.

#### 2.1.2 Market Overview

* **E-commerce Landscape:** Analyze the size, growth rate, and key players in the Egyptian e-commerce market.
* **Consumer Behavior:** Understand consumer preferences, online shopping habits, and factors influencing purchase decisions.
* **Internet Penetration:** Assess the level of internet connectivity and smartphone usage in Egypt.
* **Payment Methods:** Evaluate the availability and popularity of different payment methods (e.g., credit cards, digital wallets, cash on delivery).

#### 2.1.3 Opportunities for Local Brands:

* **Leveraging E-commerce Platforms:** Explore opportunities to list products on our e-commerce platforms.
* **Search Engine Optimization (SEO):** Optimize websites for search engines to improve organic visibility.
* **Mobile Optimization:** Ensure websites and mobile apps are optimized for a seamless user experience on mobile devices.

#### 2.1.4 Challenges and Recommendations:

* **Challenges:** Identify key challenges faced by local brands in the e-commerce space, such as logistics, payment options, and competition.
* **Recommendations:** Provide actionable recommendations to overcome these challenges and capitalize on market opportunities.

### 2.2 Benchmarking

#### 2.2.1 Overview:

* **Our Project:**
  + An online marketplace dedicated to Egyptian brands, designed to enhance visibility and accessibility for local products.
  + Aims to connect local businesses directly with consumers through a comprehensive shopping platform.
* **صنع في مصر:**
  + A platform that promotes Egyptian products and brands, primarily focusing on raising awareness about locally made goods.
  + Emphasizes national pride and the quality of Egyptian craftsmanship.

#### 2.2.2 Goals:

* **Our Project:**
  + Establish a sustainable online marketplace.
  + Leverage AI for enhanced product search capabilities.
  + Provide a unique shopping experience tailored to consumer needs.
* **صنع في مصر:**
  + Promote local products and foster national pride among consumers.
  + Raise awareness about the availability and quality of Egyptian goods.

#### 2.2.3 Target Audience:

* **Our Project:**
  + Primarily targets young adults and working professionals (ages 18-45) across urban and suburban areas.
  + Also include students, expatriates, and tourists interested in local products.
* **صنع في مصر:**
  + Broadly targets all Egyptian consumers interested in supporting local products, appealing to a diverse demographic.

#### 2.2.4 Features:

* **Our Project:**
  + Comprehensive e-commerce functionalities, including user registration, AI-powered Chatbot and Recommendation System, shopping cart, and brand partnership systems.
  + Focus on user experience with mobile-friendly design and social media integration.
  + Prices may be more competitive for smaller quantities, especially considering potential import duties and shipping costs.
* **صنع في مصر:**
  + Primarily focuses on promoting products and brands rather than full e-commerce experience.
  + May include features for browsing products but lacks extensive shopping functionalities.

#### 2.2.5 Community and Economic Impact:

* **Our Project:**
  + Directly aims to stimulate local economies by increasing sales for Egyptian brands and creating job opportunities.
  + Seeks to build a community of consumers and local businesses, fostering loyalty and engagement.
* **صنع في مصر:**
  + Focuses on raising awareness and supporting local artisans and businesses but may not provide direct sales facilitation, which limits immediate economic impact.

#### 2.2.6 Technological Integration:

* **Our Project:**
  + Plans to utilize advanced technology, such as AI for product discovery and a robust platform for managing product listings and customer interactions.
* **صنع في مصر:**
  + Likely employs a simpler technological framework, focusing more on promotion rather than advanced e-commerce capabilities.

#### 2.2.7 Additional Points About UI/UX For Our Project:

* **Responsive Design**: Ensure that your website adapts seamlessly to different screen sizes, making it easy to navigate on mobile devices.
* **Touch-Optimized Elements:** Design elements that are easy to tap and interact with touchscreens.
* **Hamburger Menus:** Consider using hamburger menus to condense navigation options on smaller screens.
* **Account Dashboard:** Provide a personalized dashboard where users can access their order history, saved items, and account settings.
* **Search Functionality:** Provide a robust search bar that allows users to quickly find products or information.
* **Products Review:** Ensure that the user is able to review and see other users reviews on specific products and interact with them through comments.

### 2.3 Data Gathering

#### 2.3.1 Overview:

* **Purpose:** The goal of this data collection is to gather information on Egyptian local brands for an online shopping platform. The objective is to compile a database of brands in categories like clothing, makeup, and skincare, including product offerings, price ranges, digital presence, and other key details.
* **Data Sources:** Data will be sourced from:
  + Official websites of the local brands
  + Social media pages (Instagram, Facebook, etc.)
* **Data Collection Methods:**
  + **Online Research:** Manual research will be conducted to gather brand information from their websites and social media accounts.
* **Data Quality Assurance:** To ensure data accuracy, all collected data will be verified by cross-referencing multiple sources (websites and social media). Team members will regularly review data entries to ensure consistency. If there are discrepancies, follow-up research will be conducted.

#### 2.3.2 Data Collection Procedures:

##### 2.3.2.1 Timeline:

* + Week 1-2: Initial research and data gathering from websites and social media.
  + Week 3: Data verification and cross-referencing.
  + Week 4: Final data review and compilation into the database.

##### 2.3.2.2 Roles and Responsibilities:

* + **Zeyad (Data Gathering Lead)**:
    - Leads the data collection strategy.
    - Manages communication with local brands to gather product details.
    - Ensure all required data fields are collected.
  + **Sara, Sandreen, Mostafa (Data Scrapers)**:
    - Develop web scrapers using Python and Node.js to automate data collection.
    - Extract product information from brand websites and other online sources.
    - Ensure scraping scripts are efficient and maintainable.
  + **Ali, Mohamed (Data Verifiers)**:
    - Cross-check scraped data with primary sources to ensure accuracy.
    - Use automated scripts and manual methods to validate data integrity.
    - Collaborate with the Database Managers to address discrepancies.
  + **Ahmed, Reem (Database Managers)**:
    - Design and maintain MongoDB database schemas.
    - Oversee data entry, backups, and migrations.
    - Optimize database performance and ensure data security.

#### 2.3.3 Data Management Plan:

##### 2.3.3.1 Data Storage:

* + **Database:** The main repository for all brand information, hosted on a cloud server.
  + **Local Storage:** Backup files will be stored on local drives for safety and quick access.
  + **Cloud Storage**: Backup Images using Cloudinary.

##### 2.3.3.2 Data Security:

* + Access to the cloud server will be restricted through login credentials.
  + Regular backups will be taken to avoid data loss.
  + Sensitive information will be encrypted where necessary.

##### 2.3.3.3 Data Access:

* + Only authorized team members will have full access to the database.
  + Other team members will have restricted access depending on their roles (e.g., data verification, analysis).

# Chapter 3



## Requirements Analysis

### 3.1 Functional Requirements

#### 3.1.1 Types of Users Across the Platform

* **Customer:** The primary user role, representing individuals who purchase products on the platform.
* **Merchant:** Businesses that sell and manage (creating, adding, deleting) their products through the platform.
* **Admin:** The highest level of administrative authority, with full control over the platform and its settings, will be responsible for managing the overall platform, including user accounts, products, and orders.

#### 3.1.2 Users Roles and Permissions

* **Admin:**
  + Create, edit, and delete, toggle roles for user’s accounts (Merchant and customer).
  + Manage all products, categories, and brands.
  + Create, edit, and delete products.
  + Approval of the requests created by Merchant to add new products.
  + Manage system settings and configurations.
  + Process payments and refunds.
  + Monitor platform performance and analytics.
  + Handle customer support inquiries.
  + Access and modify system code.
  + Oversee platform security and compliance.
* **Merchant:**
  + Add, edit, and delete only their own products.
  + Set product prices, quantities, and availability.
  + View sales analytics and performance metrics.
  + Communicate with customers by reviews.
  + Access merchant-specific settings and tools.
  + Cannot manage user accounts or system settings.
* **Customer:**
  + Create and manage an account.
  + Browse and search for products.
  + Filter Products based on attributes (e.g., price, category, brand, color, size).
  + Add products to their cart.
  + Remove or clear items from their cart.
  + Add, remove Products to his Wishlist.
  + View both Wishlist and Cart.
  + Checkout and place orders.
  + View order history and tracking.
  + View and Write reviews for products.

#### 3.1.3 Authentication and Authorization

* **User Registration:**
  + Allow users to create accounts with unique email addresses and passwords.
  + Implement strong password requirements to enhance security.
  + Provide options for social login (e.g., Google, Facebook) for convenience.
* **Login Process:**
  + Verify user credentials upon login to ensure authorized access.
  + Implement robust password hashing and salting to protect against unauthorized access.
  + Consider using multi-factor authentication (MFA) for enhanced security, requiring additional verification factors (e.g., SMS codes, security tokens).
* Profile Management:
  + Permit users to update their profile information, change passwords, and manage preferences.
* **Password Recovery:**
  + Provide a password recovery or reset mechanism to allow users to regain access to their accounts in case of forgotten passwords.
  + Implement security measures to prevent unauthorized password resets.
* **Role-Based Access Control (RBAC):**
  + Assign different roles to users based on their privileges (e.g., customer, merchant, admin).
  + Define permissions for each role to control access to different features and resources.
  + Ensure that users can only access features and data that they are authorized to see.
* **Token-Based Authentication:**
  + Issue unique tokens to authenticated users to identify them in subsequent requests.
  + Use secure token generation and storage mechanisms to prevent unauthorized access.
  + Set expiration times for tokens to mitigate security risks.
* **Data Encryption:**
  + Encrypt sensitive data (e.g., credit card information, personal data) at rest and in transit to protect against unauthorized access.
  + Use strong encryption algorithms and protocols.
* **Input Validation:**
  + Validate user input to prevent malicious attacks like SQL injection and cross-site scripting (XSS).
  + Sanitize user input before processing it.

#### 3.1.4 Product Management

* **Admin:**
  + Add, edit, and delete products over the platform.
  + Create, edit, and delete categories and brands.
  + Upload product images.
  + Track product stock levels in real-time to ensure accurate availability information.
* **Merchant:**
  + Add, edit, and delete products for their specific brand.
  + Assign products to appropriate categories.
  + Manage product attributes, prices, quantities, and availability.
  + Create product bundles, offering discounts or incentives for purchasing multiple products together.
  + Upload product images.
  + Adjust stock levels manually (e.g., for returns, damaged goods).
* **Customer:**
  + Browse and search for products.
  + View product details, including images, descriptions, and prices.
  + Filter products by category, brand, price, and other attributes.
  + Navigate across recommended and featured products.

#### 3.1.5 Product Search and Filtering

* **Keyword Search:**
  + Implement a robust search function that allows customers to find products by keyword, description, or brand.
  + Consider using natural language processing for improved search results.
* **Filtering Options:**
  + Provide filters for various product attributes (e.g., price, category, brand, color, size).
  + Allow customers to refine their search based on their preferences.

#### 3.1.6 Chatbot

* + The chatbot shall provide product recommendations based on user browsing and purchase history.
  + The chatbot shall analyze the user's Wishlist and recommend similar products.
  + The recommendation system shall dynamically update suggestions based on real-time user interactions.

#### 3.1.7 Shopping Cart

* **Product Addition:**
  + Allow customers to add products to their shopping cart with ease.
  + The Platform will provide a clear indication of added products, such as a confirmation message or updated cart total.
* **Product Removal:**
  + Enable customers to remove products from their shopping cart if they change their minds.
  + The Platform will provide a clear confirmation message upon removal.
* **Quantity Adjustment:**
  + Allow customers to adjust the quantity of products in their shopping cart.
  + Ensure that the cart total is updated accordingly.
* **Cart Persistence:**
  + Maintain the shopping cart across sessions, even if the customer closes their browser or logs out.
  + The Platform will use session cookies to save cart data, even if they log out or close their browser.
* **Cart Summary:**
  + Display a clear summary of the shopping cart, including product names, quantities, prices, and total cost.
  + Provide options to view more details about individual products.
* **Wishlist Functionality:**
  + Allow customers to save products to a Wishlist for later purchase.
  + Provide a seamless transition from Wishlist to cart.

#### 3.1.8 Checkout and Payment

* **Shipping Information:**
  + Collect customer's shipping information, including address, phone number, and preferred delivery method.
  + Validate shipping address and provide real-time shipping estimates.
* **Points Exchange for Promo Codes**
  + Users will be able to exchange their points with different promocodes through their account dashboard.
* **Payment Options:**
  + Offer a variety of payment methods, including credit cards, debit cards, digital wallets (e.g., PayPal, Apple Pay), and cash on delivery.
* **Order Confirmation:**
  + Display an order confirmation page that summarizes the order details, including products, quantities, shipping information, and payment method.
  + Provide an option for customers to print or save the order confirmation.
* **Order Tracking:**
  + Provide a link to track the order status on the order confirmation page or in the customer account.
* **Secure Payment Integration:**
  + Integrate with secure payment gateways to process transactions securely.
  + Ensure that all sensitive data is encrypted and transmitted securely.
* **Error - Success Handling:**
  + Provide clear error messages if there are issues with payment or shipping information.
  + Provide a successful message or page after payment completion.

#### 3.1.9 Reviews and Ratings

* **Customer Reviews:**
  + Allow customers to submit reviews for products they have purchased.
  + Require customers to log in or create an account to submit reviews.
  + Provide a rating system (e.g., stars) for customers to rate products.
  + Allow merchants to publicly reply to customer reviews, fostering engagement and building trust.
* **Review Moderation:**
  + Implement a review moderation process to prevent spam, inappropriate content, or fake reviews.
  + Review and approve or reject reviews before they are published.
  + Product pages should display average ratings and customer reviews.
* **Product Pages:**
  + Displaying average product ratings and the number of reviews on product pages.
  + Show individual reviews with timestamps and customer names (if allowed).

#### 3.1.10 Other Features

* **Profile Information:**
  + Allow customers to view and edit their personal information, including name, email address, phone number, and shipping address.
  + Provide a secure password change option.
* **Customer Wishlist:**
  + Enable customers to add products to their Wishlist.
  + Allow customers to remove products from their Wishlist or move them to their shopping cart.
* **Customer Support** 
  + Provide Customer Support via ai chatbot conversation.
  + Provide multiple channels for customers to contact support (email, phone, forms).
  + Offer a comprehensive FAQ section to address common customer inquiries.
* **Notification System**
  + Users will be receiving notifications about new products , successful payments , order updates ,etc.

### 3.2 Non-Functional Requirements

#### 3.2.1 Performance

* **Response time:** We used the latest technologies available to provide fast performance with reduced response time which makes users enjoy the best performance experience. (e.g., loading the homepage, searching for a product).
* **Scalability:** Of course, a big idea needs scalability to include a larger number of users, and we provide this in our application by scalability database.
* **Peak Traffic:** The platform should be able to handle increased traffic during peak times, such as holidays or sales events.

#### 3.2.2 Security

* **Data protection:** we used mechanisms like (e.g., using multi-factor authentication, Using secure token generation) for protecting user's and admin's sensitive data (e.g., passwords, payment information)
* **Secure Payment Processing:** The payment gateway should comply with relevant security standards (e.g., PCI DSS).
* **Hack prevention:** validation and sanitize the inputs before process to make sure that it’s not cyberattacks like SQL injection or XSS.

#### 3.2.3 Accessibility

* **Compliance with Standards:** Adhere to accessibility guidelines (e.g., WCAG) to ensure the platform can be used by people with disabilities.
* **Alternative Text for Images:** Provide descriptive text for images to assist visually impaired users.

#### 3.2.4 Availability

* **Uptime:** Our system is designed to be 99.99% ready to use at all times, allowing for the convenience of use at any time.
* **Recovery plans:** There is feedback can the users use to report if there is any problem and there are plans to follow to guarantee their rights.

#### 3.2.5 Usability

* **User interface:** Designed to be smooth and easy to use and surely, make users understand how to use and reach to what they want by best ways.
* **Responsive Design:** The website will adapt to different screen sizes (desktop, tablet, mobile) for optimal user experience.
* **User-Friendly Checkout:** The checkout process should be straightforward and secure.

#### 3.2.6 Reliability

* **Stability:** Our system is stable to provide users deal with it safely.
* **Data accuracy:** Surely, it’s important to care about the quantity of products that will be shown to users to know if it’s available or not.
* **Data Backup:** Regularly back up critical data to ensure recoverability in case of data loss.

#### 3.2.7 Integration

* **Third-Party Integrations:** Integrate with relevant third-party services (e.g., payment gateways, shipping providers, marketing tools).
* **RESTful APIs:** Provide well-documented RESTful APIs for developers to interact with the platform's functionality.
* **API Security:** Implement robust security measures (e.g., OAuth, API keys) to protect sensitive data.
* **API Documentation:** Provide clear and comprehensive API documentation to guide developers in using the platform's APIs.

#### 3.2.8 Maintainability

* **Ease of modification:** merchants can modify anything about their products easily and quickly and admins can do this but have more accessibility.
* **Code Quality:** Ensure clean, well-structured code for easier maintenance and updates.
* **Documentation:** Complete documentation of all processes, system components and what it does.
* **Modularity:** Design the platform with modular components for easier updates and scalability.

### 3.3 Technical Requirements

#### 3.3.1 Project Management and Documentation

* Project planning and organization: Notion
* Task management: GitHub Projects
* Documenting: Microsoft Word - Excel
* Development Environment: Visual Studio Code
* Version Control: Git
* Repository: GitHub
* Design Tool: Figma

#### 3.3.2 Frontend Development

* Framework: ReactJS
* CSS Framework: Tailwind CSS & Bootstrap
* UI Component Library: Daisy UI

#### 3.3.3 Backend Development

* Server-Side Framework: Node.js
* Web Framework: Express.js
* Database: MongoDB
* Authentication: JSON Web Tokens (JWT)
* API Development: RESTful APIs
* Testing Tool: Postman

#### 3.3.4 Mobile App Development

* Framework: Flutter
* Programming Language: Dart

#### 3.3.5 AI Technologies

* Natural Language Processing (NLP) Libraires for our specific goal.
* using collaborative filtering, content-based filtering, or hybrid methods for our Recommendation System.
* Leverage Machine Learning (ML) for user behavior analysis and recommendation improvements over time.

### 3.4 User Stories

#### 3.4.1 Customer-Centric User Stories

* **As a customer,** I want to create and manage my account, including updating my personal information and preferences.
* **As a customer,** I want to have a secure login process with options for social login and password recovery.
* **As a customer,** I want to be able to easily browse and search for products on the platform.
* **As a customer**, I want to be able to browse through different categories, brand and filter products for my specific needs.
* **As a customer,** I want to add products to my cart and proceed to check out with seamless experience.
* **As a customer,** I want to be able to save products to a Wish list for later purchase.
* **As a customer,** I want to get Recommended products based on my Wishlist and purchase history
* **As a customer,** I want to receive clear and timely communication about my order status, including shipping updates and delivery confirmation.
* **As a customer,** I want to be able to read product reviews and ratings from other customers.
* **As a customer,** I want to receive notifications about new products and discounts
* **As a customer,** I want to receive personalized product recommendations based on my purchase history and browsing behavior.
* **As a customer,** I want to get discounts based on my collected points
* **As a customer,** I want to have a variety of payment options available, including credit cards, debit cards, and digital wallets.
* **As a customer,** I want to have easy access to customer support through multiple channels, such as email, phone, and live chat.
* **As a customer,** I want to receive prompt and helpful responses to my inquiries and issues.
* **As a customer,** I want to have access to a comprehensive FAQ section with answers to common questions.
* **As a customer,** I want to be able to track my order history and view previous purchases.

#### 3.4.2 Merchant-Centric User Stories

* **As a merchant,** I want to easily add, edit, and delete products on the platform.
* **As a merchant,** I want to be able to create and manage product variations (e.g., size, color) within a single listing.
* **As a merchant,** I want to be able to upload high-quality product images and videos.
* **As a merchant,** I want to have the ability to set product prices, quantities, and availability.
* **As a merchant,** I want to be able to track product stock levels in real-time and receive notifications when stock is low.
* **As a merchant,** I want to be able to create and manage promotions and discounts for my products.
* **As a merchant,** I want to have access to detailed sales analytics and performance metrics to track the success of my products.
* **As a merchant,** I want to be able to receive feedback from customers through product reviews and ratings.
* **As a merchant,** I want to be able to track order status and communicate with customers about shipping updates and my sales analytics.
* **As a merchant,** I want to be able to connect with payment gateways to accept various payment methods.
* **As a merchant,** I want to be able to communicate with customers through messaging or live chat.

#### 3.4.3 Admin-Centric User Stories

* **As an admin,** I want to be able to manage user accounts, including creating, editing, and deleting them.
* **As an admin,** I want to be able to assign different roles and permissions to users based on their needs.
* **As an admin,** I want to be able to manage product categories and brands.
* **As an admin,** I want to be able to set up and customize the platform's appearance and branding.
* **As an admin,** I want to be able to manage system settings, including payment gateways, shipping options, and tax rates.
* **As an admin,** I want to be able to review and approve or reject product listings submitted by merchants.
* **As an admin,** I want to be able to manage product reviews and ratings, ensuring they comply with platform guidelines.
* **As an admin,** I want to be able to handle customer disputes and resolve issues between merchants and customers.
* **As an admin,** I want to have access to detailed analytics and reporting on platform performance, user behavior, and sales.
* **As an admin,** I want to be able to implement and manage security measures to protect the platform and user data.
* **As an admin,** I want to have tools to manage and respond to customer support inquiries.

### 3.5 User Flow

#### 3.5.1 Customer visits the platform's homepage.

* **Step 1: Sign Up**
  + User clicks on the "Sign Up" or "Create Account" button.
  + Users enter their email address, password, and other required information (e.g., name, address).
  + User agrees to the terms of service and privacy policy.
  + User clicks "Create Account."
* **Step 2: Account Confirmation**
  + Platform sends a confirmation email with code to the user's registered email address.
  + Users enter the code on the confirmation page to verify their account.
  + User is redirected to login page.
* **Step 3: Login**
  + Users can log in to their account using their email address and password.
  + Platform offers a password recovery option for forgotten passwords.
* **Step 4: Profile Management**
  + Users can update their personal information (e.g., name, address, phone number).
  + Users can change their password.
  + User can access his cart and Wishlist.
  + Uses can read, delete and clear their notifications
  + Users can exchange their points into coupons or promocodes
* **Step 5: Browse Products**
  + Users can browse through product categories, brands, and use search filters.
  + Users can view product details, including images, descriptions, and prices.
  + Users can read product reviews and ratings.
  + Users can find recommended products based on their preferences
* **Step 6: Mange Wishlist**
  + Users can add products to their Wishlist.
  + Users can view their Wishlist.
  + User can remove and clear items form his Wishlist.
  + Users can add products to their cart through Wishlist.
* **Step 7: Mange Cart**
  + Users can add products to their shopping cart.
  + Users can view their cart contents and adjust quantities.
  + Users can remove items from their cart.
  + User can clear his cart items.
* **Step 8: Checkout**
  + User proceeds to check out.
  + User enters shipping information and selects a payment method.
  + Users review their order and confirm the purchase.
  + Users can track their order status.
  + Users can provide coupons or promocode to get discounts
* **Step 9: Customer Support**
  + Users can contact customer support for inquiries or issues.
  + Platform offers multiple channels for support (email, phone).
* **Step 10: Reviews and Ratings**
  + Users can write reviews and ratings for purchased products.
  + Users can view other customers’ reviews.
* **Step11: Chatbot Assistance**
  + User accesses the chatbot through a floating widget or a dedicated chat interface.
  + Users ask the chatbot for product suggestions based on preferences (e.g., category, brand, budget).
  + Users can add suggested products directly to the cart or Wishlist from the chat interface.

#### 3.5.2 Merchant logs in to the platform.

* **Step 1: Access Merchant Dashboard**
  + Merchants navigate their merchant dashboard.
  + Merchant views his own brand products list
* **Step 2: Product Listing**
  + Merchant clicks on the "Add Product" button.
  + Merchant enters product details:
    - Product name
    - Description
    - Price
    - Quantity
    - Availability
    - Category
    - Brand
    - Merchant uploads product images.
    - other variations (e.g., size, color).
* **Step 3: Edit Product**
  + Merchants can edit details for the product.
  + Merchants can add discounts for the product.
* **Step 5: Product Listing Review**
  + Platform reviews the product listed to be Approved (Pending)
  + If approved, the product is published.
  + Merchants can view real-time stock levels.
* **Step 8: Manage Orders**
  + Merchants can view and manage orders related to their products.
  + Merchants can track order status and communicate with customers.
* **Step 9: View Analytics**
  + Merchants can access sales data, including total sales, revenue, and customer demographics.
  + Merchants can read product reviews and ratings from customers.

#### 3.5.3 Admin logs in to the platform.

* **Step 1: Access Admin Dashboard**
  + Admin navigates to the admin dashboard.
* **Step 2: User Management**
  + Admin can create, edit, and delete user accounts.
  + Admin can assign different roles and permissions to users (e.g., customer, merchant, admin).
* **Step 3: Product Management**
  + Admin can manage product categories and brands.
  + Admin can create products with details.
  + Admin can review and approve or reject product listings submitted by merchants.
* **Step 4: Platform Settings and Analytics**
  + Admin can customize the platform's appearance and branding (e.g., logo, colors, theme).
  + Admin can manage system settings, including payment gateways, shipping options, and tax rates.
  + Admin can access detailed analytics on platform performance, user behavior, and sales.
* **Step: Security and Compliance**
  + Admin can implement security measures (e.g., password policies, data encryption).
  + Admin can ensure compliance with relevant regulations (e.g., GDPR).
* **Step 7: Customer Support**
  + Admin can view and manage customer support inquiries.
  + Admin can respond to customer inquiries and resolve issues.
  + Admin can handle disputes between merchants and customers.

# Chapter 4



## Project Plan and Feasibility Study

### 4.1 Development Methodology

#### 4.1.1. Introduction

In our e-commerce project, we are adopting the **Agile Development Methodology**, specifically the **Scrum** framework. This approach allows us to be flexible, respond to change quickly, and continuously improve our product through iterative development. Our goal is to ensure that we deliver high-quality features that meet user needs while fostering collaboration within our team.

#### 4.1.2. Roles and Responsibilities

To implement Scrum effectively, our team will embrace defined roles that promote collaboration and accountability:

* **Scrum Master**:
  + Our Scrum Master will facilitate all Scrum processes, ensuring that the team adheres to Agile principles. This individual will remove any impediments that may hinder progress, coordinate meetings, and protect the team from external disruptions.
* **Product Owner**:
  + The Product Owner will act as the main point of contact between stakeholders and the development team. They will manage the Product Backlog, prioritize tasks based on business value, and ensure that user feedback is integrated into the development process.
* **Development Team**:
  + Composed of cross-functional members, our Development Team will include developers, designers, and testers. This team is responsible for delivering potentially shippable increments at the end of each sprint. Members will collaborate closely, self-organizing their tasks to achieve sprint goals.

#### 4.1.3. Scrum Artifacts and Their Use

Our team will utilize key Scrum artifacts to maintain transparency and focus:

* **Product Backlog**:
  + We will continuously refine and prioritize our Product Backlog to ensure it reflects the most critical features and improvements. The Product Owner will gather input from stakeholders to keep this document up to date.
* **Sprint Backlog**:
  + Before each sprint, we will hold a Sprint Planning meeting where the team selects items from the Product Backlog. The Sprint Backlog will serve as our roadmap for the sprint, outlining tasks that need to be completed to meet our Sprint Goal.
* **Increment**:
  + At the end of each sprint, we will produce a potentially shippable increment of the product. This increment will undergo testing to ensure it meets our quality standards and the Definition of Done.

#### 4.1.4. Scrum Events and Their Implementation

Our team will participate in several structured Scrum events to facilitate communication and progress tracking:

* **Sprint Planning**:
  + We will conduct a Sprint Planning meeting at the beginning of each sprint, lasting approximately 2-4 hours. During this meeting, we will define the Sprint Goal and select high-priority items from the Product Backlog for development.
* **Sprint Review**:
  + At the end of each sprint, we will conduct a Sprint Review meeting. Here, we will demonstrate the completed work to stakeholders, gather their feedback, and discuss any adjustments needed for the Product Backlog. This meeting helps ensure we are aligned with user expectations.
* **Sprint Retrospective**:
  + Following the Sprint Review, we will hold a Sprint Retrospective. This meeting will allow us to reflect on the past sprint, discussing what went well, what didn’t, and how we can improve our processes moving forward.

#### 4.1.5. Practical Steps for Our Team

To effectively implement the Scrum framework, our team will follow these practical steps:

1. **Kickoff Meeting**:
   * We will initiate our project with a kickoff meeting, introducing team members and discussing our goals, roles, and the Scrum process.
2. **Initial Backlog Creation**:
   * Our Product Owner will work with stakeholders to create an initial Product Backlog. This will include user stories, features, and enhancements prioritized by business value.
3. **Sprint Cycles**:
   * We will operate in two-week sprints, allowing frequent iterations and feedback. Each sprint will begin with Sprint Planning and conclude with Sprint Review and Retrospective meetings.
4. **Communication Channels**:
   * We will be using Discord & WhatsApp for ongoing communication, ensuring that team members can easily share updates and ask questions.
5. **Documentation and Tracking**:
   * We will use Notion and Microsoft Word and GitHub Project to manage our Product Backlog and track progress. All team members will be responsible for updating their tasks and keeping the backlog current.

### 4.2 Timeline:

* Total Duration: 7 Months (28 - 30 weeks)
* Total Sprints: 15 sprints (2 weeks each)

#### 4.2.1 Phase 1: Research and Documentation (1 month)

**Duration:** Month 1

* **Week 1:** Conduct Market Research
  + Identify potential competitors and analyze their strengths and weaknesses.
  + Gather data on target customers’ preferences and behaviors.
  + Document market trends and relevant statistics.
* **Week 2:** Competitor Analysis
  + Identify key features and functionalities of competitors’ platforms.
  + Conduct a feature gap analysis to determine unique selling points.
  + Document insights from competitor analysis.
* **Week 3:** Platform Architecture Design
  + Draft the initial architecture plan for the platform.
  + Determine essential modules, including admin and merchant panels, product catalog, and user management.
  + Decide on technologies to be used for front-end and back-end.
* **Week 4:** Platform Design
  + Develop wireframes and layout designs for all user panels.
  + Collaborate with stakeholders to finalize the look and feel of the platform.
  + Document design guidelines and user flow.

#### 4.2.2 Phase 2: Platform Implementation (4 months)

**Duration:** Months 2 - 5

* **Month 2: Platform Core Development**

**Weeks 1 - 2:** Setting Up E-commerce Core

* + - Database Setup: Design and initialize databases to store products, user data, orders, and transactions.
    - API Development: Begin developing core APIs for the following:
    - Product Display: List and categorize products for users.
    - User Profiles: Store user account information and preferences.
    - Cart Management: Enable users to add, remove, and update items in the cart.
    - Basic Frontend Setup: Implement initial front-end interface and navigation for the core pages, such as homepage, product listings, and user login.

**Weeks** 3 - 4: Core Functionalities and User Management

* + - Cart and Checkout Process: Build out the cart functionality for adding/removing items, implement a preliminary checkout flow, including summary pages.
    - User Authentication and Roles: Set up login and registration with role-based access (e.g., users, admins, merchants), implement basic profile management where users can view and update their personal details.
    - Establish sessions and data caching for a smooth user experience and performance optimization.
* **Month 3: Merchant - Admin Panel** 
  + **Weeks 1 - 2:** Implement Admin-Merchant Panel
    - Develop base functionalities for admin and merchant dashboards, including login, user management, and product upload options.
    - Add features for product management, order tracking, and sales reporting.
    - Implement user role management for secure admin access.
  + **Weeks 3 – 4: Admin-Merchant Core Features** 
    - Allow merchants to manage their product listings, inventory, and view sales.
    - Set up merchant-specific order management and reporting.
* **Month 4 Weeks 1 - 2: Shipping and Feature Integration** 
  + Shipping Feature Integration
    - Integrate options for shipping carriers, tracking, and order status updates.
    - Ensure correct calculation of shipping costs based on weight, size, and destination.
  + Payment Integration
    - Integrate payment gateway(s) and ensure secure transaction processing.
    - Set up refund and transaction failure handling mechanisms.
* **Month 4 - 5: AI Technology Selection and Integration**
  + **Month 4: Weeks 3 – 4:** Research and Evaluation of AI Tools
    - Research AI image search technologies compatible with the platform.
    - Select and test a suitable AI tool for recommendation.
    - evaluate model accuracy, response time, and customization options.
  + **Month 5 Weeks 1 - 2:** Planning and Architecture for Integration
    - Plan how the AI image search will interface with existing modules, especially the product catalog and user panels.
    - Plan how the recommendation chatbot will be integration with the database and user preferences.
    - Define endpoints and data flows for image search, recommended products requests and responses.
    - Establish Security Measures: Set up authentication for API calls to secure AI technology integrations.
  + **Month 5 Weeks 3 - 4:** AI Integration and Testing
    - Connect the AI tool’s API with the platform’s backend to enable image search functionality.
    - Implement server-side scripts to handle image uploads, send them to the AI service, and retrieve results.
    - Design a user-friendly interface where users can upload or take photos for image-based search.
    - Create a results page that displays AI-generated product matches with relevant filters and sorting options.
    - Set up the system to process AI results, translate image search responses into product recommendations or listings, and display results.

#### 4.2.3 Phase 3: Testing & Improvements (2 weeks)

**Duration:** Month 6, Weeks 1 - 2

* **Week 1:** Testing
  + Conduct functional testing across all modules (e-commerce core, admin panel, merchant panel, payment, and shipping).
  + Perform usability testing with sample users to get feedback on user experience.
  + Test the AI-powered image search for accuracy and responsiveness.
* **Week 2:** Security and Improvements
  + Conduct security testing to ensure data protection and compliance with regulations.
  + Fix bugs or issues identified during testing.
  + Make improvements based on feedback from testing.

#### 4.2.4 Phase 4: Final Testing and Deployment (2 weeks)

**Duration:** Month 6, Weeks 3 - 4

* **Week 3:** Final Testing
  + Perform load and stress testing to assess performance under high traffic.
  + Finalize any last-minute adjustments or fixes.
* **Week 4:** Deployment
  + Set up production environment and ensure deployment readiness.
  + Deploy the platform to a live server.
  + Monitor platform post-launch for stability and functionality.

#### 4.2.5 Gantt Chart:

A multicolored chart with text

Description automatically generated

#### 4.2.6 Milestones:

A diagram of a program

Description automatically generated with medium confidence

### 4.3. Resource Allocation

#### 4.3.1. Team Roles and Responsibilities

* Team Lead: Ahmed Algrgawy
  + Coordinates between teams, facilitates communication, and oversee project timelines.
  + Ensures alignment with Agile practices, conducts team meetings (such as stand-ups), and helps remove blockers.
  + Tracks project progress and reports to stakeholders.
  + Develops and maintains comprehensive project documentation, including user manuals and technical guides.
  + Works with each team to ensure documentation reflects current features and practices.
  + Prepares documentation that supports Agile processes and assists onboarding and training for new team members.
* Web Front-End Developer: Sandy Kotp – Zeyad Elkhamary
  + Develops the web interface, focusing on functionality, responsiveness, and user experience.
  + Works closely with the UI/UX team to implement designs.
  + Ensures cross-browser compatibility and optimizes the application for speed and scalability.
* Mobile Developer: Sara Elkholy – Mostafa Negm
  + Develops the mobile app using Flutter, ensuring compatibility across iOS and Android.
  + Works with back-end and UI/UX teams to integrate functionality and ensure design consistency.
  + Focuses on mobile-specific optimizations, performance, and user experience.
* UI/UX Designer: Mohamed Siam
  + Designs the user interface and overall experience for both web and mobile platforms.
  + Conducts user research, creates wireframes, prototypes, and ensures a cohesive design language.
  + Works with both front-end and mobile developers to translate designs into the final product.
* Back-End Developer: Reem Ghareeb – Ahmed Algrgawy
  + Develops and maintains server-side logic, databases, and application functionality.
  + Works with the front-end and mobile developers to establish and maintain APIs.
  + Focuses on system architecture, data management, and application performance.
* Security Specialist: Ali El-Beltagy
  + Implements and manages security protocols to protect data and system integrity.
  + Conducts vulnerability assessments and works with developers to fix security gaps.
  + Ensures compliance with data protection regulations and best practices.
* AI/ML Engineer: Ahmed Algrgawy
  + Design and train the recommendation system (e.g., collaborative filtering, content-based, hybrid models).
  + Develop and fine-tune NLP models for understanding user queries and generating responses.
  + Preprocess data and manage pipelines for model training and updates.
  + Evaluate model performance and optimize algorithms.

#### 4.3.2. Documentation Types

* **Project Planning and Organization**:
  + **Notion**: For organizing project timelines, roadmaps, and meeting notes.
* **Task Management**:
  + **GitHub Projects**: For tracking issues, bugs, and development tasks.
* **Technical Documentation**:
  + **Microsoft Word**: For API documentation, architectural diagrams, and technical specs.
* **Data Documentation**:
  + **Microsoft Excel**: For data mapping, tracking product details, and inventory management.
* **Design Documentation**:
  + **Figma**: For wireframes, UI prototypes, and design guidelines

#### 4.3.3. Tools and Software

* **Frontend Development**:
  + **Framework**: ReactJS
  + **CSS Frameworks**: Tailwind CSS, Bootstrap
  + **UI Components**: Daisy UI
  + **IDE**: Visual Studio Code
  + **Version Control**: Git & GitHub
* **Backend Development**:
  + **Languages & Frameworks**: Node.js, Express.js
  + **Database**: MongoDB
  + **Authentication**: JSON Web Tokens (JWT)
  + **API Testing**: Postman
* **Mobile App Development**:
  + **Framework**: Flutter
  + **Language**: Dart
* **Design and Prototyping**:
  + **Design Tool**: Figma
* **AI and Machine Learning Development:**
  + **Frameworks and Libraries:** **TensorFlow,** Hugging Face Transformers, scikit-learn
  + **Data Processing and Visualization:** Pandas and NumPy and Matplotlib
* **Project Management & Communication**:
  + **Task Management**: GitHub Projects
  + **Documentation**: Notion, Microsoft Word, Excel

#### 4.3.4. Monitoring and Evaluation

* **Code Quality**:
  + **ESLint & Prettier**: For maintaining coding standards.
  + **GitHub Actions**: For continuous integration and automated testing.
* **Data Accuracy**:
  + **Automated Data Validation**: Scripts to validate data against source.
  + **Manual Review**: Regular cross-checks by the Data Verifiers team.
* **Performance Monitoring**:
  + **Frontend Performance**: Lighthouse for auditing web performance.
  + **Backend Monitoring**: PM2 for Node.js process management.
  + **Database Monitoring**: MongoDB Atlas for performance metrics and alerts.
* **User Feedback & Testing**:
  + **User Testing**: Conduct usability testing sessions with target users.
  + **Bug Tracking**: GitHub Issues for reporting and tracking bugs.

### 4.4 Financial Feasibility

#### 4.4.1 Development Costs:

* **Hosting and Cloud Services**
  + **Cloud Servers:** AWS, Google Cloud, or DigitalOcean for hosting back-end and databases.
    - **Cost:** $50–$300/month (basic plan for initial traffic).
  + **Database Hosting:** Managed SQL/NoSQL databases.
    - **Cost:** Included with cloud hosting or $50–$200/month.
* **Version Control and Collaboration**
  + GitHub, GitLab, or Bitbucket (team plan).
    - **Cost:** $0–$25/user/month (free plans available for students).
* **AI and Machine Learning Tools**
  + Services like TensorFlow, scikit-learn (free), or APIs for advanced AI (e.g., OpenAI, Google AI).
    - **Cost:** $0–$100/month (depending on API usage).
* **Cybersecurity Tools**
  + Penetration testing tools (e.g., Burp Suite, Nessus).
    - **Cost:** $0–$500 (student discounts available).
  + SSL Certificates for secure connections.
    - **Cost:** $50–$100 annually.
* **Design Tools**
  + Figma, Adobe XD, or Canva for UI/UX design.
    - **Cost:** $0–$20/user/month (educational discounts available).
* **Other Tools**
  + Testing frameworks (e.g., Selenium, Jest).
    - **Cost:** Free or up to $50/month for premium versions.

#### 4.4.2 Ongoing Operating Costs:

* **Cloud Hosting and Database Storage**: Estimated based on anticipated traffic and storage needs for images, user data, and product information.
* **Maintenance**: Budget for platform updates, performance optimization, and bug fixes.
* **Customer Support**: Costs for either internal support teams or outsourced customer service.

#### 4.4.3 Revenue Model:

* + **Commission on Sales**: A percentage from each transaction.
  + **Subscription for Brands**: Monthly or annual subscription plans for premium features, including analytics, promotions, and product highlights.
  + **Ad Revenue**: Opportunities for brands to advertise within the app, increasing visibility to a targeted audience.

#### 4.4.4 Break-Even Analysis:

* + Calculate monthly revenue targets to cover operating costs, aiming for a break-even within the first year.
  + Financial goals: A balance between user engagement growth and brand partnerships that drive recurring revenue.

#### 4.4.5 Funding Requirements:

* Low Estimate (Basic Needs): $500–$2,000
  + Assuming you leverage free tools, college-provided resources, and minimal hardware upgrades.
* Moderate Estimate (Additional Features): $3,000–$6,000
  + If you require premium tools, new hardware, or additional cloud and AI resources.
* High Estimate (Comprehensive Setup): $7,000–$10,000+
  + If scaling infrastructure, conducting professional-grade testing, or marketing heavily.

### 4.5 Risk Assessment:

#### 4.5.1. Identify Potential Risks:

##### 4.5.1.1 Technical Risks:

* Website security breaches (hacking, data breaches, …etc.).
* System failures (server downtime, software bugs, …etc.).
* Payment gateway issues (fraudulent transactions, processing errors, …etc.).
* Network vulnerabilities (DDoS attacks, malware infections, …etc.).
* UI and front-end issues with web applications or mobile.

##### 4.5.1.2 Operational Risks:

* Supply chain disruptions (stockouts, delayed deliveries, …etc.).
* Inventory management errors (overstock, understock, …etc.).
* Logistics challenges (shipping errors, loss of packages, …etc.).
* Order processing mistakes (incorrect orders, delayed shipments).

##### 4.5.1.3 Financial Risks:

* Payment processing fees.
* Chargebacks and refunds.
* Economic downturns.
* Currency fluctuations.

##### 4.5.1.4 Legal and Regulatory Risks:

* Non-compliance with data privacy regulations (GDPR, CCPA).
* Intellectual property infringement.
* Tax and customs issues.
* Consumer protection laws.
* Copyright and Merchandise Sales.

#### 4.5.2. Assess Risk Likelihood and Impact:

##### 4.5.2.1 Likelihood:

* **Technical Risks**: there is High probability for this at the beginning of the coding and development stage, but it will be reduced after reviewing and updating.
* **Operational Risks:** it will be low probability or less because these risks happen in every project and we try to make it chance to happen.
* **Financial Risks:** there are a lot of rumors about it but it is a medium probability to happen and we have plans to deal with.
* **Legal and Regulatory Risks:** it has medium probability for these risks because many reasons like users don’t follow our terms and instructions.

##### 4.5.2.2 Impact:

* **Technical Risks**: it will be minor consequences because it will not affect usage of the users very badly.
* **Operational Risks:** it will have moderate consequences for the users and can make users uncomfortable using our project.
* **Financial Risks:** it will have severe consequences and make users scared for their money to lose.
* **Legal and Regulatory Risks:** it will be severe consequences Because it makes us pay a lot of compensation to users in many cases if we did the mistake.

#### 4.5.3 Prioritize Risks:

* Of course, the most influential will always have priority, then the least influential will come.
* The priority will be Financial Risks and Legal and Regulatory Risks because of their influence.
* Secondly, priority will be given to Operational Risks then Technical Risks they less priority but must care about them for the comfort of the user.

#### 4.5.4. Develop Mitigation Strategies:

##### 4.5.4.1 Technical Controls:

* + Strong passwords and encryption.
  + Regular security audits and vulnerability assessments.
  + Firewalls and intrusion detection systems.
  + Secure payment gateways and fraud prevention tools.
  + Reviewing, updating and developing the system.

##### 4.5.4.2 Operational Controls:

* + Robust inventory management systems.
  + Reliable logistics partners.
  + Efficient order processing procedures.
  + Regular system backups and disaster recovery plans.

##### 4.5.4.3 Financial Controls:

* + Diversified payment options.
  + Insurance coverage.
  + Financial forecasting and budgeting.

##### 4.6.4.4 Legal and Regulatory Controls:

* + Stay updated on relevant laws and regulations.
  + Consult with legal experts.
  + Implement data privacy policies and procedures.
  + Make sure the users know the terms and privacy policies.

# Chapter 5



## System Design and Diagrams

### Use Case Diagram

#### 5.1.1 Customer Use case

A diagram of a customer

Description automatically generated

#### 5.1.2 Admin Use case

A diagram of a product

Description automatically generated

#### 5.1.3 Merchant Use case

A diagram of a product

Description automatically generated

### Use Case Table

#### Customer Use Cases Tables

1. **Register (shared)**

|  |  |
| --- | --- |
| **Use case ID** | **UC-01** |
| **Use Case name** | Register |
| **Actor** | Customer , Merchant |
| **Precondition** | User is not already registered. |
| **Trigger** | User decides to create an account. |
| **Main Flow** | 1. User provides necessary details ( name, email, password). 2. System validates inputs. 3. Account is created and confirmation is sent. |
| **Postcondition** | User has an active account. |
| **Exception** | Invalid or incomplete details prevent registration. |

1. **Login (shared)**

|  |  |
| --- | --- |
| **Use case ID** | **UC-02** |
| **Use Case name** | Login |
| **Actor** | Customer ,Merchant |
| **Precondition** | User is already registered. |
| **Trigger** | User attempts to log in. |
| **Main Flow** | 1. User enters email and password. 2. System validates credentials.   3. User is granted access. |
| **Postcondition** | User is logged into their account. |
| **Exception** | Invalid credentials result in login failure.. |

1. **Browse / Search Products**

|  |  |
| --- | --- |
| **Use case ID** | **UC-03** |
| **Use Case name** | Browse / Search Products |
| **Actor** | Customer |
| **Precondition** | Customer is on the platform. |
| **Trigger** | Customer searches for or browses products. |
| **Main Flow** | 1. Customer enters a search query or navigates categories.   2. System displays relevant results. |
| **Postcondition** | Customer sees a list of products. |
| **Exception** | No matching products found. |

1. **Filter Products**

|  |  |
| --- | --- |
| **Use case ID** | **UC-04** |
| **Use Case name** | Filter Products |
| **Actor** | Customer |
| **Precondition** | Customer is viewing a list of products.. |
| **Trigger** | Customer applies filters (size, price, brand). |
| **Main Flow** | 1. Customer selects filter criteria.  2. System updates the product list based on filters. |
| **Postcondition** | Filtered products are displayed. |
| **Exception** | No products match the selected filters. |

1. **View Product Details**

|  |  |
| --- | --- |
| **Use case ID** | **UC-05** |
| **Use Case name** | View Product Details |
| **Actor** | Customer |
| **Precondition** | Customer is browsing or searching products. |
| **Trigger** | Customer clicks on a product. |
| **Main Flow** | 1. Customer selects a product.  2. System displays product details  ( images, description, price). |
| **Postcondition** | Customer views product details. |
| **Exception** | Product details fail to load due to system error.. |

1. **Navigate Recommended Products**

|  |  |
| --- | --- |
| **Use case ID** | **UC-06** |
| **Use Case name** | Navigate Recommended Products |
| **Actor** | Customer |
| **Precondition** | Customer is logged in. |
| **Trigger** | Customer views recommendations. |
| **Main Flow** | 1. System displays recommended products.  2. Customer navigates through the list. |
| **Postcondition** | Customer explores recommended products. |
| **Exception** | Recommendations fail to load. |

1. **Manage Cart**

|  |  |
| --- | --- |
| **Use case ID** | **UC-04** |
| **Use Case name** | Manage Cart |
| **Actor** | Customer |
| **Precondition** | Customer is logged in. |
| **Trigger** | Customerdecides to manage their cart. |
| **Main Flow** | 1. Customer adds/removes items or clears the cart.  2. System updates the cart. |
| **Postcondition** | Cart reflects the desired state. |
| **Exception** | System fails to update the cart. |

1. **Manage / View Wishlist**

|  |  |
| --- | --- |
| **Use case ID** | **UC-08** |
| **Use Case name** | Manage / View Wishlist |
| **Actor** | Customer |
| **Precondition** | Customer is logged in. |
| **Trigger** | Customer views or updates wishlist. |
| **Main Flow** | 1. Customer adds/removes items to/from the wishlist.  2. System updates the wishlist. |
| **Postcondition** | Wishlist reflects the desired state. |
| **Exception** | Wishlist fails to update. |

1. **Checkout & Place Orders**

|  |  |
| --- | --- |
| **Use case ID** | **UC-09** |
| **Use Case name** | Checkout & Place Orders |
| **Actor** | Customer |
| **Precondition** | Cart contains items. |
| **Trigger** | Customer proceeds to checkout. |
| **Main Flow** | 1. Customer enters payment and shipping details.  2. System processes the order.  3. Confirmation is displayed. |
| **Postcondition** | Order is placed successfully. |
| **Exception** | Payment fails or order processing error occurs. |

1. **View / Write Reviews**

|  |  |
| --- | --- |
| **Use case ID** | **UC-10** |
| **Use Case name** | View / Write Reviews |
| **Actor** | Customer |
| **Precondition** | Customer has purchased a product. |
| **Trigger** | Customer views or writes a review. |
| **Main Flow** | 1. Customer navigates to the review section.  2. Customer writes or reads reviews. |
| **Postcondition** | Reviews are displayed or updated. |
| **Exception** | Review fails to save or load. |

1. **View Order History**

|  |  |
| --- | --- |
| **Use case ID** | **UC-11** |
| **Use Case name** | View Order History |
| **Actor** | Customer |
| **Precondition** | Customer has placed previous orders. |
| **Trigger** | Customer navigates to order history. |
| **Main Flow** | 1. Customer selects the order history option.  2. System displays previous orders. |
| **Postcondition** | Order history is displayed. |
| **Exception** | Order history fails to load.. |

1. **Manage Notifications**

|  |  |
| --- | --- |
| **Use case ID** | **UC-12** |
| **Use Case name** | Manage Notifications |
| **Actor** | Customer |
| **Precondition** | Customer opened the platform |
| **Trigger** | Customer navigates to notification area. |
| **Main Flow** | 1. Customer read the notification.  2. customer delete the notification |
| **Postcondition** | none |
| **Exception** | There is no notifications |

1. **Chat with Chatbot**

|  |  |
| --- | --- |
| **Use case ID** | **UC-13** |
| **Use Case name** | Chat with chatbot |
| **Actor** | Customer |
| **Precondition** | Customer opened the platform |
| **Trigger** | Customer navigates to chatbot chat. |
| **Main Flow** | 1. Customer send a prompt  2. customer receive a response |
| **Postcondition** | Logged in |
| **Exception** | Chatbot does not response |

#### 5.2.2 Merchant Use Cases Tables

1. **Track Stock Levels**

|  |  |
| --- | --- |
| **Use case ID** | **UC-14** |
| **Use Case name** | Track Stock Levels |
| **Actor** | Merchant |
| **Precondition** | Merchant has products listed in the system. |
| **Trigger** | Merchant checks product stock levels. |
| **Main Flow** | 1. Merchant navigates to the stock section.  2. System displays stock levels for all products. |
| **Postcondition** | Merchant views current stock levels. |
| **Exception** | System fails to retrieve stock data. |

1. **Manage Their Own Products**

|  |  |
| --- | --- |
| **Use case ID** | **UC-15** |
| **Use Case name** | Manage Their Own Products |
| **Actor** | Merchant |
| **Precondition** | Merchant has access to their product list. |
| **Trigger** | Merchant decides to manage products. |
| **Main Flow** | 1. Merchant views the product list.  2. Merchant adds, removes, edits, or uploads images for products. |
| **Postcondition** | Product list is updated as per the merchant's actions. |
| **Exception** | System fails to update product data |

**15.1- Add Products**

|  |  |
| --- | --- |
| **Use case ID** | **UC-15.1** |
| **Use Case name** | Add Products |
| **Actor** | Merchant |
| **Precondition** | Merchant is logged in. |
| **Trigger** | Merchant decides to add a new product. |
| **Main Flow** | 1. Merchant enters product details (e.g., name, description, price).  2. System saves the new product. |
| **Postcondition** | Product is added to the system. |
| **Exception** | Validation errors prevent the product from being added. |

**15.2- Remove Products**

|  |  |
| --- | --- |
| **Use case ID** | **UC-15.2** |
| **Use Case name** | Remove Products |
| **Actor** | Merchant |
| **Precondition** | Merchant is logged in and has products listed. |
| **Trigger** | Merchant chooses to remove a product. |
| **Main Flow** | 1. Merchant selects a product to remove.  2. System removes the product from the list. |
| **Postcondition** | Product is removed from the system. |
| **Exception** | Product fails to delete due to system error. |

**15.3- Edit Products**

|  |  |
| --- | --- |
| **Use case ID** | **UC-15.3** |
| **Use Case name** | Edit Products |
| **Actor** | Merchant |
| **Precondition** | Merchant is logged in and has products listed. |
| **Trigger** | Merchant chooses to edit product. |
| **Main Flow** | 1. Merchant selects a product to edit.  2. Merchant updates product details.  3. System saves the changes. |
| **Postcondition** | Product details are updated. |
| **Exception** | System fails to save the changes. |

**15.4- Upload Product Images**

|  |  |
| --- | --- |
| **Use case ID** | **UC-15.4** |
| **Use Case name** | Upload Product Images |
| **Actor** | Merchant |
| **Precondition** | Merchant is logged in and has products listed. |
| **Trigger** | Merchant uploads images for a product. |
| **Main Flow** | 1. Merchant selects a product.  2. Merchant uploads an image file.  3. System attaches the image to the product. |
| **Postcondition** | Product is updated with new images. |
| **Exception** | Image fails to upload due to size or format errors. |

1. **Communicating with Customers**

|  |  |
| --- | --- |
| **Use case ID** | **UC-16** |
| **Use Case name** | Communicate with Customers |
| **Actor** | Merchant |
| **Precondition** | Merchant has active customers. |
| **Trigger** | Merchant initiates or responds to communication. |
| **Main Flow** | 1. Merchant views customer queries.  2. Merchant sends responses or promotional messages. |
| **Postcondition** | Communication is successfully exchanged. |
| **Exception** | Messages fail to send due to system issues. |

1. **Create Product Bundles**

|  |  |
| --- | --- |
| **Use case ID** | **UC-17** |
| **Use Case name** | Create Product Bundles |
| **Actor** | Merchant |
| **Precondition** | Merchant has multiple products listed. |
| **Trigger** | Merchant decides to create a bundle. |
| **Main Flow** | 1. Merchant selects products for the bundle.  2. Merchant sets a bundle price.  3. System saves the bundle. |
| **Postcondition** | Product bundle is available in the system. |
| **Exception** | Bundle fails to save due to system errors. |

1. **Assign Products to Categories**

|  |  |
| --- | --- |
| **Use case ID** | **UC-18** |
| **Use Case name** | Assign Products to Categories |
| **Actor** | Merchant |
| **Precondition** | Merchant is logged in and has products listed. |
| **Trigger** | Merchant decides to assign products to categories. |
| **Main Flow** | 1. Merchant selects a product.  2. Merchant assigns a category.  3. System updates the product's category. |
| **Postcondition** | Product is assigned to the selected category. |
| **Exception** | Category assignment fails due to system issues. |

**19- View Sales Analytics**

|  |  |
| --- | --- |
| **Use case ID** | **UC-19** |
| **Use Case name** | View Sales Analytics |
| **Actor** | Merchant |
| **Precondition** | Merchant has made sales. |
| **Trigger** | Merchant views analytics. |
| **Main Flow** | 1. Merchant navigates to the analytics section.  2. System displays sales data (e.g., revenue, top products). |
| **Postcondition** | Merchant reviews sales analytics. |
| **Exception** | Analytics data fails to load. |

**20-Set Product Availability & Prices**

|  |  |
| --- | --- |
| **Use case ID** | **UC-20** |
| **Use Case name** | Upload Product Images |
| **Actor** | Merchant |
| **Precondition** | Merchant is logged in and has products listed. |
| **Trigger** | Merchant uploads images for a product. |
| **Main Flow** | 1. Merchant selects a product.  2. Merchant uploads an image file.  3. System attaches the image to the product. |
| **Postcondition** | Product is updated with new images. |
| **Exception** | Image fails to upload due to size or format errors. |

#### Admin Use Case Tables

**21- Handle Customer Support**

|  |  |
| --- | --- |
| **Use case ID** | **UC-21** |
| **Use Case name** | Handle Customer Support |
| **Actor** | Admin |
| **Precondition** | Customer support tickets or inquiries must exist. |
| **Trigger** | Admin accesses the customer support module. |
| **Main Flow** | 1. View unresolved tickets.  2. Respond to customer queries or escalate issues.  3. Mark tickets as resolved. |
| **Postcondition** | Customer inquiries are resolved and recorded. |
| **Exception** | If no response is provided by the customer, close the ticket as unresolved. |

**22- Monitor Platform Performance**

|  |  |
| --- | --- |
| **Use case ID** | **UC-22** |
| **Use Case name** | Monitor Platform Performance |
| **Actor** | Admin |
| **Precondition** | Admin must be logged in with analytics access. |
| **Trigger** | Admin navigates to the performance dashboard. |
| **Main Flow** | 1. View real-time platform metrics (uptime, traffic).  2. Generate performance reports. |
| **Postcondition** | Admin reviews insights for platform optimization. |
| **Exception** | If metrics fail to load, display an error message. |

**23- Process Payments and Refunds**

|  |  |
| --- | --- |
| **Use case ID** | **UC-23** |
| **Use Case name** | Process Payments and Refunds |
| **Actor** | Admin |
| **Precondition** | Transactions or refund requests must exist. |
| **Trigger** | Admin accesses the payment module. |
| **Main Flow** | 1. Review transaction details.  2. Approve or reject refund requests. |
| **Postcondition** | Payments or refunds are processed successfully. |
| **Exception** | If payment gateway fails, notify relevant parties. |

**24- Manage System Settings**

|  |  |
| --- | --- |
| **Use case ID** | **UC-24** |
| **Use Case name** | Manage System Settings |
| **Actor** | Admin |
| **Precondition** | Admin must have superuser permissions. |
| **Trigger** | Admin navigates to the system settings page. |
| **Main Flow** | 1. Modify global configurations (platform policies, themes).  2. Save changes. |
| **Postcondition** | Updated settings are applied to the platform. |
| **Exception** | If changes fail to save, show an error message. |

**25- Oversee Platform Security**

|  |  |
| --- | --- |
| **Use case ID** | **UC-25** |
| **Use Case name** | Oversee Platform Security |
| **Actor** | Admin |
| **Precondition** | Admin must have security privileges. |
| **Trigger** | Admin selects the security module. |
| **Main Flow** | 1. Review security logs.  2. Identify and block malicious activities.  3. Implement security patches. |
| **Postcondition** | Platform security is maintained. |
| **Exception** | If security logs fail to load, escalate to the IT team. |

**26- Create Categories and Brands**

|  |  |
| --- | --- |
| **Use case ID** | **UC-26** |
| **Use Case name** | Create Categories and Brands |
| **Actor** | Admin |
| **Precondition** | Admin must be logged in. |
| **Trigger** | Admin navigates to the category or brand creation page. |
| **Main Flow** | 1. Input category or brand details.  2. Save new entries. |
| **Postcondition** | New categories or brands are added to the platform. |
| **Exception** | If duplicate entries are detected, reject the request. |

**27- Approve Merchant Product Requests**

|  |  |
| --- | --- |
| **Use case ID** | **UC-27** |
| **Use Case name** | Approve Merchant Product Requests |
| **Actor** | Admin |
| **Precondition** | Pending merchant product submissions must exist. |
| **Trigger** | Admin accesses the product approval section. |
| **Main Flow** | 1. Review product details.  2. Approve or reject the product. |
| **Postcondition** | Product status is updated (approved/rejected). |
| **Exception** | If details are incomplete, request corrections from the merchant. |

**28- Manage Products**

|  |  |
| --- | --- |
| **Use case ID** | **UC-28** |
| **Use Case name** | Manage Products |
| **Actor** | Admin |
| **Precondition** | Product listings must exist. |
| **Trigger** | Admin navigates to the product management module. |
| **Main Flow** | 1. View all products.  2. Add, edit, delete, or upload images. |
| **Postcondition** | Product listings are updated. |
| **Exception** | If invalid data is submitted, reject the request. |

**29- Track Stock Levels**

|  |  |
| --- | --- |
| **Use case ID** | **UC-29** |
| **Use Case name** | Track Stock Levels |
| **Actor** | Admin |
| **Precondition** | Products must exist in the system. |
| **Trigger** | Admin accesses the stock management module. |
| **Main Flow** | 1. View stock levels across merchants.  2. Notify merchants of low stock. |
| **Postcondition** | Stock levels are monitored, and merchants are notified. |
| **Exception** | If stock data fails to load, show an error message. |

**30- Access and Modify System Code**

|  |  |
| --- | --- |
| **Use case ID** | **UC-30** |
| **Use Case name** | Access and Modify System Code |
| **Actor** | Admin |
| **Precondition** | Admin must have backend access permissions. |
| **Trigger** | Admin navigates to the system code interface. |
| **Main Flow** | 1. Access the codebase.  2. Edit or update code settings.  3. Test and deploy changes. |
| **Postcondition** | System configurations or features are updated. |
| **Exception** | If changes fail, roll back to the previous version. |

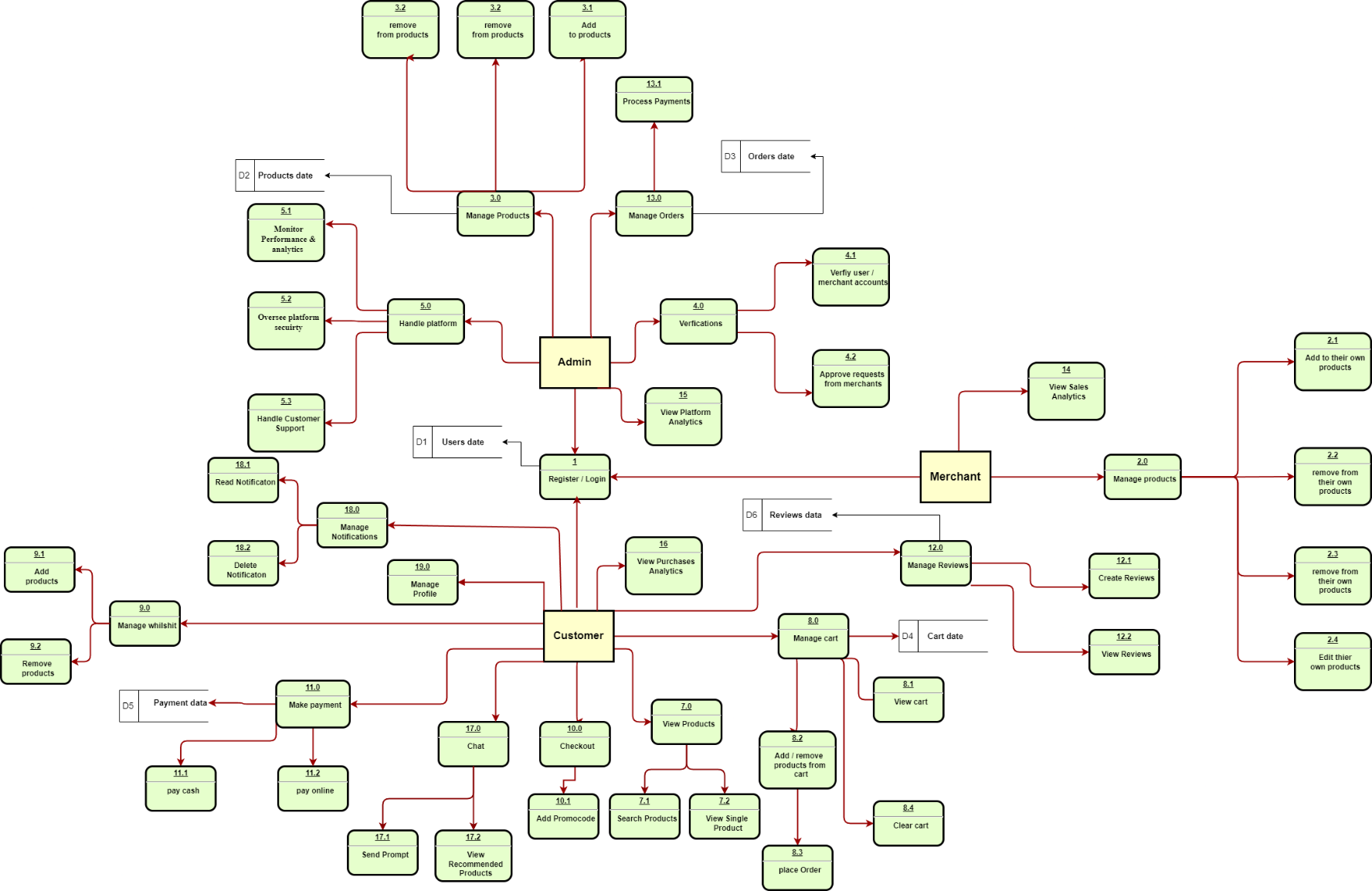
### A computer screen shot of a diagram5.3 Class Diagram

### 5.4 Context Diagram

A diagram of a diagram

Description automatically generated

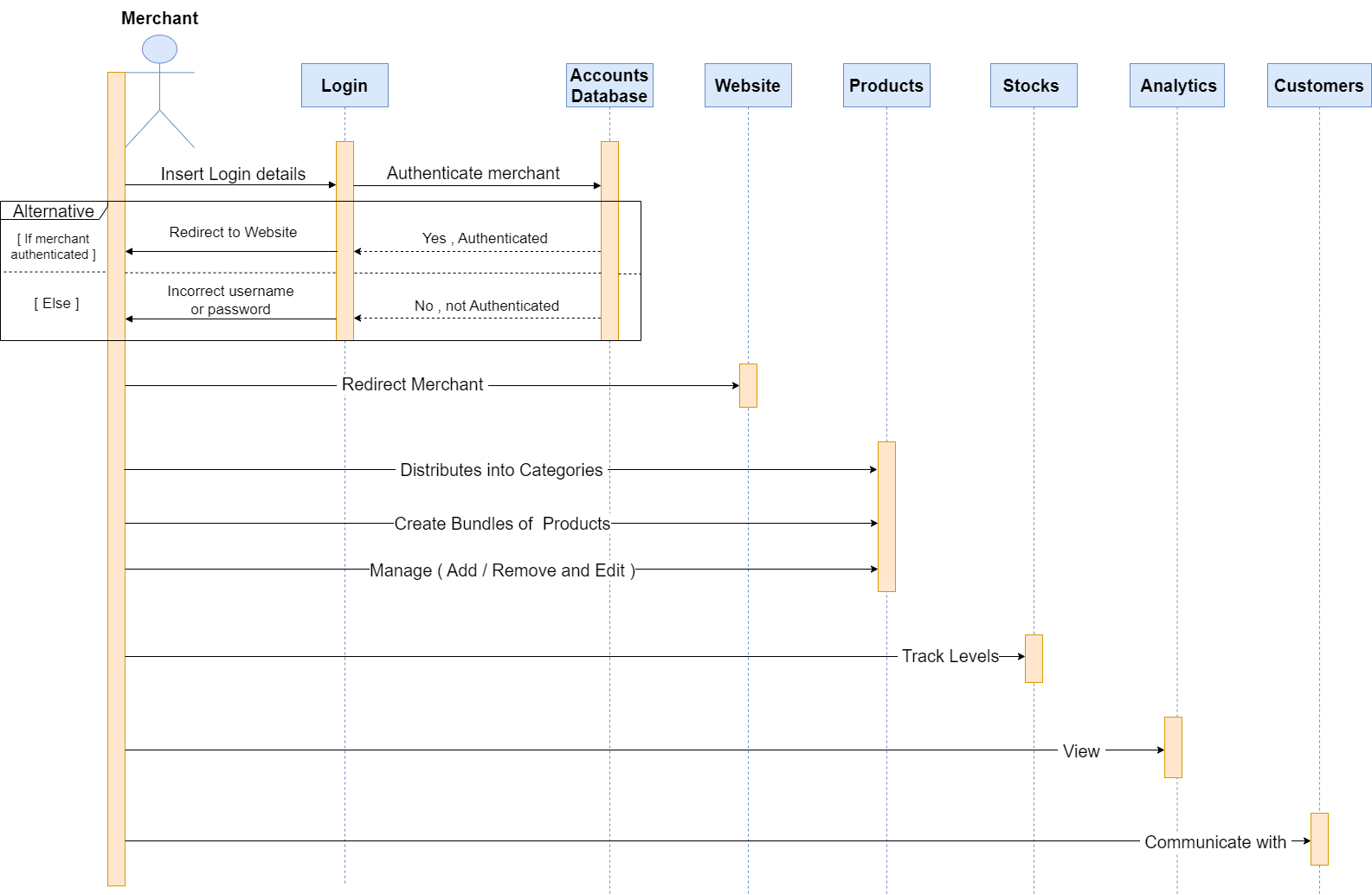
### 5.5 DFD Diagram



### 5.6 Sequence Diagram

#### A diagram of a project Description automatically generated5.6.1 User Sequence

#### 5.6.1 Merchant Sequence

****

#### 5.6.3 Admin Sequence

**A screenshot of a computer

Description automatically generated**

### 5.7 ERD Diagram

### 5.8 Normalized Tables



### 5.9 Data Dictionary

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Collection Name** | **Field Name** | **Data Type** | **Constraints** | **Default Value** | **Example Value** | **Description** |
| User | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | Name | String | Required | N/A | "John Doe" | User's full name. |
|  | Email | String | Unique, Required | N/A | "john.doe@example.com" | User's email address, used for login and communication. |
|  | Password | String | Required | N/A | "hashedpassword123" | User's password (hashed). |
|  | Number | String | Optional | N/A | "+1234567890" | User's phone number. |
|  | Img\_Url | String | Optional | N/A | "http://example.com/img.jpg" | URL to the user's profile image. |
|  | Adress | String | Optional | N/A | "123 Main St, City, Country" | User's physical address. |
|  | Role | String | Optional | "User" | "Admin" | Role of the user, e.g., "Admin", "User". |
|  | Gender | String | Optional | N/A | "Male" | User's gender. |
|  | Points | Int | Optional | 0 | 150 | User's points for rewards or loyalty programs. |
|  | PurchaseHisrory | Array | Optional | N/A | ["Product1", "Product2"] | Array of product IDs or names purchased by the user. |
|  | Recommendtion | Array | Optional | N/A | ["ProductA", "ProductB"] | Array of recommended product IDs or names for the user. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| Coupon | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | UserId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | Code | String | Unique, Required | N/A | "ABC123" | Unique code for the promotion or discount. |
|  | ExpirationData | Date | Required | N/A | "2024-12-31T23:59:59Z" | Date when the discount or offer expires. |
|  | DiscountAmount | Float | Required | N/A | 25.5 | Discount amount, usually in the relevant currency. |
|  | IsActive | Bool | Optional | TRUE | TRUE | Whether the discount is active or not. |
|  | Used\_Points | Int | Optional | 0 | 100 | Number of points used for the discount, if applicable. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| Brand | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | MerchantId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | Name | String | Required | N/A | "Egy-Brand" | Brand name. |
|  | Description | String | Optional | N/A | "This Brand Target Shoes for different ages" | Brand Description |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| Cateogry | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | Name | String | Required | N/A | "shoes" | Category name. |
|  | Description | String | Optional | N/A | "Shoes Products Only" | Category Description |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| Product | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | CategoryId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | BrandId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d2 | Unique identifier for the document in the database. |
|  | Name | String | Required | Brand Name' | "Wide Leg Pants" | Product name. |
|  | Description | String | Optional | N/A | "Wide Leg Pants with different sizes and colores" | Product Description |
|  | Quantity | Int | Required | N/A | 50 | The quantity of the product available in stock. |
|  | Price | Float | Required | N/A | 19.99 | Price of the product, usually in the relevant currency. |
|  | Colors | Array | Optional | N/A | ["Red", "Blue", "Green"] | Array of colors available for the product. |
|  | Img\_Url | String | Optional | N/A | "http://example.com/img.jpg" | URL to the product image. |
|  | Sizes | Array | Optional | N/A | ["S", "M", "L"] | Array of available sizes for the product. |
|  | isDiscount | Bool | Optional | FALSE | TRUE | Whether the product has a discount or not. |
|  | DiscountAmount | Float | Optional | 0 | 5 | The discount amount for the product, if applicable. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| Reviews | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | UserId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | ProdcutId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d2 | Unique identifier for the document in the database. |
|  | Rating | Float | Required | N/A | 4.5 | Rating given to the product, usually on a scale from 1 to 5. |
|  | Text | String | Optional | N/A | "Great product, would buy again!" | Text of the review, describing the user's experience. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the review was created. |
|  | Comments | Array | Optional | N/A | ["Excellent!", "Highly recommended."] | Array of comments or replies to the review. |
| Payments | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | orderId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | UserId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | Method | String | Required | N/A | "Credit Card" | Payment method used by the customer (e.g., "Credit Card", "PayPal"). |
|  | Status | String | Required | N/A | "Completed" | Status of the payment (e.g., "Pending", "Completed"). |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the payment was made. |
| Order | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | UserId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | CouponId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d2 | Unique identifier for the document in the database. |
|  | ShippingAddress | String | Optional | N/A | "123 Main St, City, Country" | Shipping address where the order will be delivered. |
|  | ShippingMethod | String | Optional | N/A | "Standard Shipping" | Shipping method selected for the order (e.g., "Standard Shipping", "Express"). |
|  | TrackCode | String | Optional | N/A | "XYZ12345" | Tracking code for monitoring the shipment. |
|  | Status | String | Required | N/A | "Shipped" | Current status of the order (e.g., "Pending", "Shipped", "Delivered"). |
|  | TotalPrice | Float | Required | N/A | 199.99 | Total price of the order, including shipping and taxes. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| OrderItem | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | orderId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | ProdcutId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d2 | Unique identifier for the document in the database. |
|  | Quantity | int | Required | N/A | 2 | Quantity of the product ordered. |
|  | Price | float | Required | N/A | 19.99 | Price of the product for this specific order item, usually in the relevant currency. |
|  | Size | String | Optional | N/A | "M" | Size of the product (if applicable, e.g., "S", "M", "L"). |
|  | color | String | Optional | N/A | "Red" | Color of the product (if applicable). |
|  | TotalPrice | Float | Required | N/A | 199.99 | Total price of the order, including shipping and taxes. |
| Cart | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | UserId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | TotalPrice | Float | Required | N/A | 199.99 | Total price of the order, including shipping and taxes. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| CartItem | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | CartId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d2 | Unique identifier for the document in the database. |
|  | ProdcutId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d2 | Unique identifier for the document in the database. |
|  | Quantity | int | Required | N/A | 2 | Quantity of the product ordered. |
|  | Price | float | Required | N/A | 19.99 | Price of the product for this specific order item, usually in the relevant currency. |
|  | Size | String | Optional | N/A | "M" | Size of the product (if applicable, e.g., "S", "M", "L"). |
|  | color | String | Optional | N/A | "Red" | Color of the product (if applicable). |
|  | TotalPrice | Float | Required | N/A | 199.99 | Total price of the order, including shipping and taxes. |
| Wishlist | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | UserId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | Comments | Array | Optional | N/A | ["603d9b98b8c53b15bc7c31d1","603d9b98b8c53b15bc7c31d1"] | Array of product IDs or names added by the user |
| Notification | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | SenderId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | ReciverId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | Content | String | Required | N/A | "Your order has been shipped!" | The content of the notification message. |
|  | Type | String | Required | N/A | "Order Update" | Type of notification (e.g., "Order Update", "Promotion"). |
|  | IsRead | Bool | Optional | FALSE | TRUE | Whether the notification has been read by the user. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
| Admin Analytics | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | AdminId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | TotalSales | Float | Optional | 0 | 1,250,000.50 | Total sales generated across the entire platform. |
|  | UsersNumbers | Int | Optional | 0 | 10,000 | Total number of users registered on the platform. |
|  | ProductsNumbers | Int | Optional | 0 | 5,000 | Total number of products available on the platform. |
|  | BrandsNumbers | Int | Optional | 0 | 250 | Total number of brands listed on the platform. |
|  | CategoryNumbers | Int | Optional | 0 | 20 | Total number of product categories. |
|  | OrdersNumbers | Int | Optional | 0 | 50,000 | Total number of orders placed across the platform. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| Merchant Analytics | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | MerchantId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | TotalSales | Float | Optional | 0 | 50000.75 | Total sales generated by the merchant from their products. |
|  | ProductsNumbers | Int | Optional | 0 | 120 | Total number of products listed by the merchant. |
|  | Orders\_Numbers | Int | Optional | 0 | 350 | Total number of orders placed for the merchant's products. |
|  | AveregeRatings | Float | Optional | 0 | 4.5 | Average rating of the merchant's products based on customer reviews. |
|  | SalesGrowth | float | Optional | 0 | 33.5 | sales groth of merchant brand products |
|  | StockLevel | Int | Optional | 0 | 250 | Total stock level for all products managed by the merchant. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |
| Customer Analytics | id | ObjectId | Primary key, Required | Auto-generated | 603d9b98b8c53b15bc7c31d0 | Unique identifier for the document in the database. |
|  | CustomerId | ObjectId | Foreign key, Required | N/A | 603d9b98b8c53b15bc7c31d1 | Unique identifier for the document in the database. |
|  | OrdersNumbers | Int | Optional | 0 | 15 | Total number of orders placed by the customer. |
|  | TotalReviews | Int | Optional | 0 | 8 | Total number of reviews submitted by the customer. |
|  | PointsEarned | Int | Optional | 0 | 1200 | Total points earned by the customer through purchases, promotions, or referrals. |
|  | PurchaseHistory | Array | Optional | N/A | ["Product1", "Product2"] | Array of product IDs or names purchased by the user. |
|  | TotalSpending | Float | Optional | 0 | 1500.5 | Total amount spent by the customer, including all orders and transactions. |
|  | CreatedAt | Date | Required | Current timestamp | "2024-12-03T08:00:00Z" | Date and time when the document was created. |
|  | UpdatedAt | Date | Required | Current timestamp | "2024-12-03T08:30:00Z" | Date and time when the document was last updated. |

### 5.10 Algorithms & Pseudocode

#### 5.10.1 Customer Algorithms and pseudocodes

**Step 1: Sign Up**

**Algorithm: User Registration**

1. Display the "Sign Up" form with fields: email, password, name, and address.
2. On form submission:
   * Validate user input (e.g., check for empty fields, valid email format, password strength).
   * Store user data securely in the database (hash the password).
   * Send a confirmation email to the registered email address.
3. Notify the user to verify their account via email.

**Pseudocode:**

Input: email, password, name, address

Validate(email, password, name, address)

If valid:

hashed\_password = Hash(password)

SaveUser(email, hashed\_password, name, address)

SendConfirmationEmail(email)

Display("Please verify your account via the confirmation email.")

Else:

Display("Invalid input. Please try again.")

**Step 2: Account Confirmation**

**Algorithm: Verify Account**

1. Wait for the user to enter the verification code from the confirmation email.
2. On submission:
   * Check if the code matches the one sent.
   * If valid, activate the account and redirect the user to the login page.

**Pseudocode:**

Input: verification\_code

If Match(verification\_code, UserCode):

ActivateAccount(user)

RedirectTo("LoginPage")

Else:

Display("Invalid verification code.")

**Step 3: Login**

**Algorithm: User Login**

1. Display the login form with email and password fields.
2. On submission:
   * Validate credentials against the database.
   * If correct, start a session and redirect to the dashboard.
   * If incorrect, prompt the user to retry or reset the password.

**Pseudocode:**

Input: email, password

If Authenticate(email, password):

StartSession(user)

RedirectTo("Dashboard")

Else:

Display("Invalid credentials.")

**Step 4: Profile Management**

**Algorithm: Update Profile**

1. Allow the user to edit fields such as name, address, or phone.
2. Validate changes and save them to the database.

**Pseudocode:**

Input: updated\_name, updated\_address, updated\_phone

Validate(updated\_name, updated\_address, updated\_phone)

If valid:

UpdateProfile(user, updated\_name, updated\_address, updated\_phone)

Display("Profile updated successfully.")

Else:

Display("Invalid input.")

ExchangePoints:

Input: user\_id, selected\_coupon\_id

user\_points = GetUserPoints(user\_id)

coupon = GetCouponDetails(selected\_coupon\_id)

required\_points = coupon.required\_points

If user\_points >= required\_points:

user\_points = user\_points - required\_points

UpdateUserPoints(user\_id, user\_points)

AddCouponUsert(user\_id, coupon)

NotifyUser(user\_id, "You have successfully redeemed your points for a coupon!")

Else:

Display("Insufficient points to redeem this coupon.")

**Step 5: Browse Products**

**Algorithm: Search and Browse Products**

1. Display product categories and a search bar.
2. On user interaction:
   * Filter products by category, brand, or search query.
   * Show product details when a product is selected.

**Pseudocode:**

Input: search\_query, category, brand, price\_range

products = FetchProducts(search\_query, category, brand, price\_range)

If products are not empty:

Display(products)

Else:

Display("No products found. Try adjusting your search or filters.")

**Step 6: Manage Wishlist**

**Algorithm: Add to Wishlist**

1. Allow the user to add a product to their Wishlist.
2. Check if the product already exists in the user's wishlist before adding it.
3. Allow users to view their wishLists and to delete a product from their wish list.
4. Store the product in the user's Wishlist database.

**Pseudocode:**

Input: product\_id

If Not IsInWishlist(user, product\_id):

AddToWishlist(user, product\_id)

Display("Product added to Wishlist.")

Else:

Display("Product is already in your Wishlist.")

ViewWishlist(user):

wishlist = FetchWishlist(user)

If wishlist is not empty:

Display(wishlist)

Else:

Display("Your Wishlist is empty.")

RemoveFromWishlist(product\_id):

RemoveProductFromWishlist(user, product\_id)

Display("Product removed from Wishlist.")

**Step 7: Manage Cart**

**Algorithm: Add to Cart**

1. Allow the user to add products to their cart.
2. Allow users to update quantities or remove items.
3. Update the cart database with product details and quantities.

**Pseudocode:**

Input: product\_id, quantity

AddToCart(user, product\_id, quantity)

Display("Product added to cart.")

UpdateCart(user, product\_id, new\_quantity):

If IsStockAvailable(product\_id, new\_quantity):

UpdateCartQuantity(user, product\_id, new\_quantity)

Display("Cart updated successfully.")

Else:

Display("Insufficient stock for the selected quantity.")

ViewCart(user):

cart = FetchCart(user)

If cart is not empty:

Display(cart)

Show product details, quantities, and total cost

Else:

Display("Your cart is empty.")

RemoveFromCart(user, product\_id):

RemoveProductFromCart(user, product\_id)

Display("Product removed from cart.")

**Step 8: Checkout**

**Algorithm: Checkout Process**

1. Collect shipping and payment information.
2. Confirm the order details.
3. Save the order and generate an order tracking ID.
4. Send an order confirmation email with order details and tracking ID.

**Pseudocode:**

Input: shipping\_info, payment\_method, cart

If Validate(shipping\_info, payment\_method):

order\_id = CreateOrder(user, cart, shipping\_info, payment\_method)

ClearCart(user)

SendOrderConfirmation(user, order\_id)

Display("Order placed successfully. Track your order with ID: " + order\_id)

Else:

Display("Checkout failed. Please try again.")

CacelOrder:

Input: order\_id, user\_id

Fetch order details

order = GetOrderDetails(order\_id)

Validate cancellation eligibility

If order.status in ["Pending", "Processing"] and not order.shipped:

UpdateOrderStatus(order\_id, "Canceled")

NotifyUser(user\_id, "Your order has been canceled.")

NotifyMerchant(order.merchant\_id, "Order ID {order\_id} has been canceled.")

Else:

Display("Order cannot be canceled. Contact support for assistance.")

ViewOrdersHistory:

Input: user\_id

orders = GetUserOrders(user\_id)

If orders:

Display("Your Order History:", orders)

Else:

Display("You have no orders yet.")

**Step 9: Customer Support**

**Algorithm: Contact Support**

1. Provide support options (email, phone).
2. Log the user's query for follow-up.

**Pseudocode:**

Input: query

LogSupportQuery(user, query)

Display("Your query has been submitted. Our team will contact you shortly.")

**Step 10: Reviews and Ratings**

**Algorithm: Submit Review**

1. Allow the user to write a review for purchased products.
2. Allow users to edit or delete their reviews.
3. Save the review and rating to the database.

**Pseudocode:**

Input: product\_id, review\_text, rating

If Validate(review\_text, rating):

SaveReview(user, product\_id, review\_text, rating)

Display("Thank you for your review.")

Else:

Display("Invalid input. Please try again.")

**Step 11: Chatbot for Product Recommendation**

**Algorithm: Input Sources for Recommendation**

1. User-provided preferences (e.g., category, brand, price range).
2. User's browsing history or purchase history.
3. General popularity or trending products.

**Pseudocode:**

Input: user\_query, user\_preferences (optional), user\_history (optional)

# Step 1: Preprocessing

If user\_history exists:

preferences = AnalyzeHistory(user\_history)

Else:

preferences = user\_preferences

# Step 2: Recommendation Logic

recommendations = FetchRecommendations(preferences, user\_query)

# Step 3: Display Output

If recommendations:

Display("Here are some recommendations for you:")

Display(recommendations)

ProvideOptions("Add to Cart", "Add to Wishlist", "View Details")

Else:

Display("Sorry, no products match your query. Please refine your preferences.")

#### 5.10.2 Merchant Algorithms and pseudocodes

**Step 1: Access Merchant Dashboard**

**Algorithm: Access Merchant Dashboard**

1. Authenticate merchant credentials.
2. Retrieve and display merchant-specific data: product list, orders, and analytics.

**Pseudocode:**

Input: merchant\_id

If AuthenticateMerchant(merchant\_id):

products = GetMerchantProducts(merchant\_id)

orders = GetMerchantOrders(merchant\_id)

analytics = GetMerchantAnalytics(merchant\_id)

DisplayDashboard(products, orders, analytics)

Else:

Display("Access denied. Please log in.")

**Step 2: Product Listing**

**Algorithm: Add Product**

1. Display a form for product details.
2. Validate and save product details in a "Pending Approval" state.
3. Notify the merchant about the submission status.

**Pseudocode:**

Input: product\_name, description, price, quantity, availability, category, brand, images, variations

If Validate(product\_name, description, price, quantity, availability, category, brand, images):

SaveProduct(merchant\_id, product\_name, description, price, quantity, availability, category, brand, images, variations, status="Pending")

Display("Product added and pending approval.")

Else:

Display("Invalid product details. Please try again.")

**Step 3: Edit Product**

**Algorithm: Edit Product**

1. Allow merchants to select and edit a product.
2. Validate changes and update the database.

**Pseudocode:**

Input: product\_id, updated\_details

If Validate(updated\_details):

UpdateProduct(product\_id, updated\_details)

Display("Product details updated successfully.")

Else:

Display("Invalid input. Please try again.")

**Step 5: Product Listing Review**

**Algorithm: Monitor Product Status**

1. Display status of pending and approved products.
2. Notify the merchant when a product is approved or rejected.

**Pseudocode:**

Input: merchant\_id

pending\_products = GetPendingProducts(merchant\_id)

Display(pending\_products)

# Notifications

If ProductStatusChanged(product\_id):

NotifyMerchant(product\_id, status)

**Step 8: Manage Orders**

**Algorithm: Manage Orders**

1. Retrieve merchant-specific orders.
2. Allow merchants to update order statuses and communicate with customers.

**Pseudocode:**

Input: merchant\_id

orders = GetMerchantOrders(merchant\_id)

Display(orders)

# Update order status

Input: order\_id, new\_status

UpdateOrderStatus(order\_id, new\_status)

Display("Order status updated.")

**Step 9: View Analytics**

**Algorithm: Analyze Sales Data**

1. Retrieve merchant-specific sales data.
2. Calculate key metrics and display analytics.

**Pseudocode:**

Input: merchant\_id

sales\_data = FetchSalesData(merchant\_id)

analytics = CalculateMetrics(sales\_data)

Display(analytics)

# Show product reviews

reviews = GetProductReviews(merchant\_id)

Display(reviews)

#### 5.10.3 Admin Algorithms and pseudocodes

**Step 1: Access Admin Dashboard**

**Algorithm: Access Admin Dashboard**

1. Authenticate admin credentials.
2. Retrieve and display key platform metrics and options.

**Pseudocode:**

Input: admin\_id

If AuthenticateAdmin(admin\_id):

metrics = FetchPlatformMetrics()

DisplayDashboard(metrics, admin\_options)

Else:

Display("Access denied. Please log in.")

**Step 2: User Management**

**Algorithm: Manage Users**

1. Allow admins to create, edit, or delete user accounts.
2. Assign roles and permissions.

**Pseudocode:**

# Create user

Input: user\_details, role

If Validate(user\_details, role):

CreateUser(user\_details, role)

Display("User created successfully.")

Else:

Display("Invalid input.")

# Edit user

Input: user\_id, updated\_details

If Validate(updated\_details):

UpdateUser(user\_id, updated\_details)

Display("User details updated.")

Else:

Display("Invalid input.")

# Delete user

Input: user\_id

DeleteUser(user\_id)

Display("User deleted.")

**Step 3: Product Management**

**Algorithm: Review and Manage Products**

1. Retrieve merchant-submitted products for review.
2. Approve or reject products.

**Pseudocode:**

Input: product\_id

If ReviewProduct(product\_id):

ApproveProduct(product\_id)

NotifyMerchant(merchant\_id, "Your product has been approved.")

PublishProduct(product\_id)

Else:

RejectProduct(product\_id)

NotifyMerchant(merchant\_id, "Your product was not approved.")

**Step 4: Platform Settings and Analytics**

**Algorithm: Manage Settings and Analytics**

1. Update platform settings, including branding and system configurations.
2. Retrieve detailed analytics for platform performance.

**Pseudocode:**

# Update settings

Input: branding\_details, payment\_settings, shipping\_options, tax\_rates

If Validate(branding\_details, payment\_settings, shipping\_options, tax\_rates):

UpdatePlatformSettings(branding\_details, payment\_settings, shipping\_options, tax\_rates)

Display("Settings updated successfully.")

Else:

Display("Invalid input.")

# Fetch analytics

platform\_data = FetchAnalyticsData()

Display(platform\_data)

**Step 5: Security and Compliance**

**Algorithm: Implement Security Measures**

1. Update security policies and enforce compliance regulations.

**Pseudocode:**

# Security measures

UpdatePasswordPolicy(policy\_details)

EnableDataEncryption()

Display("Security measures implemented.")

# Compliance

If CheckCompliance(platform\_data, regulations):

Display("Platform is compliant.")

Else:

Display("Non-compliance detected. Take action.")

**Step 7: Customer Support**

**Algorithm: Handle Support Inquiries**

1. Retrieve support tickets from users.
2. Allow admins to respond and resolve issues.

**Pseudocode:**

Input: support\_ticket\_id, response

ticket = GetSupportTicket(support\_ticket\_id)

If ResolveIssue(ticket, response):

NotifyUser(ticket.user\_id, "Issue resolved.")

CloseTicket(support\_ticket\_id)

Else:

Display("Resolution failed.")

### 5.11 System Architecture Diagram

A group of white circles

Description automatically generated

### 5.12 User Interface

#### 5.12.1 Web Interface

#### 5.12.2 Mobile Interface

# Chapter 6



## 6. Implementation

### 6.1 Development Environment Setup

#### 6.1.1 Introduction

Provide an overview of the development environment setup, including its purpose and prerequisites.

* **Purpose**: This document describes the steps required to set up a local development environment for the e-commerce platform project.
* **Target Audience**: Developers, QA engineers, and anyone contributing to the project.

#### 6.1.2 Prerequisites

List tools, dependencies, and accounts needed before starting the setup.

* **Operating System**: Ensure compatibility (e.g., Windows, macOS, Linux).
* **Tools Required**:
  + **Mobile Platform**: Android Studio, Android SDK (latest version).
  + **Code Editor**: Visual Studio Code
  + **Package Manager**: npm (for JavaScript), pip (for Python)
  + **Version Control**: Git
* **Accounts**:
  + Google Play Developer account for Android testing.
  + Apple Developer accounts for iOS testing.
  + Repository access (GitHub)

#### 6.1.3 Development Stack Overview

Provide an outline of the technologies used in the project.

* **Frontend**: React.js (JavaScript framework) - Tailwind CSS
* **Backend**: Node.js with Express – JWT
* **Database**: MongoDB
* **Mobile**: Flutter

#### 6.1.4 Step-by-Step Setup Instructions

##### 6.1.4.1 Step 1: Clone the Repository

1. **Access the Repository**
2. **Clone & Navigate the Project**:
   * **Git clone** [**https://github.com/ahmedalgrgawy/Graduation-Project.git**](https://github.com/ahmedalgrgawy/Graduation-Project.git)
   * **Cd** Graduation-Project

##### 6.1.4.2 Step 2: Install Required Software and Dependencies

1. **Install Node.js and npm and Flutter SDK**:
   * Ensure Node.js (version X.X.X) and npm (version X.X.X) are installed.
   * Confirm installation: node -v // npm -v.
   * Add Flutter to Path: “export PATH="$PATH:`pwd`/flutter/bin"”
2. **Install dependencies:**
   * Run “npm run build”
   * set up the Android SDK.
   * Create a .env file in the directory of the project.
3. **Add the environment variables in the format KEY=VALUE:**
   * MONGO\_URL= VALUE, PORT=VALUE, etc.

##### 61.4.3 Step 3: Run the Development Environment

1. **Start the server using “npm run dev”**
2. **Access the website folder using “cd client/”**
3. **Start the website using “npm run dev”**
4. **Access Mobile folder using "cd mobile/”**
5. **Start the app Starting the emulator then using “flutter run”**
6. **Access the Website: Open a browser and go to http://localhost:5173 to view the client side.**

### 6.2 Coding Standards

#### 6.2.1 Front-End Development Coding Standards:

##### 6.2.1.1 General Guidelines

**Use ES6+ Syntax**: Utilize modern JavaScript features such as arrow functions and template literals.

**Modular Code**: Organize components into separate files, ensuring that each file has a single responsibility.

**Consistent Naming Conventions**:

Variables and Functions: **camelCase** (fetch Data)

Components and Classes: **Pascal Case** (MainHeader)

Constants: **UPPER\_SNAKE\_CASE** (API\_URL)

##### 6.2.1.2 React Component Standards

**Function Components**: Prefer functional components over class components. Use hooks for state management and lifecycle methods.

**Prop Types**: Use Prop Types to define expected props for components and to provide default values when necessary.

**Component Composition**: Encourage reusability and composition by building smaller, reusable components instead of large, monolithic ones.

**State Management:**

**Utilize React's built-in hooks (useState, useReducer, useContext).**

For complex states, consider using Redux or Zustand.

##### 6.2.1.3 Tailwind CSS Standards

**Utility Classes**: Use Tailwind’s utility classes for styling. Avoid CSS customs unless absolutely necessary.

**Responsive Design**: Utilize Tailwind's responsive utilities to ensure components are adaptable across different screen sizes.

**Custom Styles**: If additional styles are needed, define them in a separate CSS file or use Tailwind’s @apply directive in a CSS file for clarity

##### 6.2.1.4 Code Reviews

**Peer Reviews**: All codes should undergo peer reviews before merging. Use pulls requests to facilitate discussions and feedback.

**Review Criteria**:

Code Quality: Ensure adherence to coding standards and best practices.

Functionality: Verify that the code works as intended and meets the requirements.

Testing: Check for adequate test coverage and meaningful test cases.

##### 6.2.1.5 Files Structure

project-root/

├── src/

│ ├── assets/ # Static assets like images, fonts, etc.

│ │ ├── images/ # Images folder

│ │ ├── fonts/ # Fonts folder

│ │ └── ... # Additional assets

│ ├── components/ # Reusable components

│ │ ├── Button.jsx # Example: Button component

│ │ ├── Navbar.jsx # Example: Navbar component

│ │ └── ... # Additional components

│ ├── features/ # Redux slices or Zustand stores for feature-level state

│ │ ├── auth/

│ │ │ ├── authSlice.js # Redux slice for authentication

│ │ │ └── useAuthStore.js # Zustand store for authentication

│ │ ├── products/

│ │ │ ├── productsSlice.js # Redux slice for products

│ │ │ └── useProductStore.js # Zustand store for products

│ │ └── ... # Additional features

│ ├── hooks/ # Custom hooks

│ │ ├── useFetch.js # Example: Custom fetch hook

│ │ ├── useAuth.js # Example: Authentication-related hooks

│ │ └── ... # Additional hooks

│ ├── pages/ # Pages corresponding to routes

│ │ ├── HomePage.jsx # Home page

│ │ ├── LoginPage.jsx # Login page

│ │ ├── DashboardPage.jsx # Dashboard page

│ │ └── ... # Additional pages

│ ├── routes/ # App routing configuration

│ │ ├── AppRoutes.jsx # Route definitions

│ │ └── ProtectedRoute.jsx # Wrapper for protected routes

│ ├── state/ # Global state management

│ │ ├── store.js # Redux store configuration

│ ├── styles/ # Tailwind CSS and global styles

│ │ ├── globals.css # Global Tailwind setup

│ │ ├── theme.js # Custom Tailwind theme configuration

│ │ └── ... # Additional styles

│ ├── utils/ # Utility functions

│ │ ├── api.js # Axios or Fetch API setup

│ │ ├── constants.js # App constants

│ │ ├── helpers.js # Helper functions

│ │ └── ... # Additional utilities

│ ├── App.jsx # Root app component

│ └── main.jsx # Main entry point for setup (Vite-specific)

├── public/ # Static files served directly

│ ├── index.html # HTML template

│ ├── favicon.ico # Favicon

│ └── ... # Additional public assets

├── tailwind.config.js # Tailwind CSS configuration

├── postcss.config.js # PostCSS configuration

├── package.json # Project dependencies and scripts

├── package-lock.json # Dependency tree lock file

├── .env # Environment variables

├── .gitignore # Git ignore file

└── README.md # Project documentation

#### 6.2.2 Back-End Development Coding Standards:

##### 6.2.2.1 General Guidelines

* **Use ES6+ Syntax**: Adopt modern JavaScript features, such as arrow functions, destructuring, async/await, and template literals.
* **Code Structure**: Organize code into clear modules, each with a single responsibility, to enhance maintainability and readability.
* **Consistent Naming Conventions**:
  + Use **Pascal Case** for class names (e.g., UserClass).
  + Use **kebab-case** for file names (e.g., user.controller.js).
  + Use **camelCase** for regular JavaScript variables and function names (getUser).

##### 6.2.2.2 Express.js Standards

* **Middleware Usage**: Use middleware for error handling, authentication, logging, and data validation. Modularize middleware into separate files when necessary.
* **Routing**: Use router instances to define routes for different resources. Keep routes organized in separate files within the /routes directory.
* **Error Handling**: Implement centralized error handling to manage application errors gracefully. Use separate error handler middleware.

##### 6.2.2.3 MongoDB Standards

* **Mongoose Models**: Use Mongoose to define models and schemas. Maintain a clear structure and validation for each model.
* **Queries**: Use async/wait for database queries to improve readability and error handling.

##### 6.2.2.4 Environment Configuration

* **Environment Variables**: Use a .env file for sensitive configuration, such as database connection strings and API keys. Use dotenv package to load environment variables.
* **Configuration Management**: Centralize configuration in a dedicated config module for easier management.

##### 6.2.2.5 Code Reviews

* **Peer Reviews**: All codes should undergo peer reviews before merging. Use pulls requests to facilitate discussions and feedback.
* **Review Criteria**:
  + Code Quality: Ensure adherence to coding standards and best practices.
  + Functionality: Verify that the code works as intended and meets the requirements.
  + Testing: Check for adequate test coverage and meaningful test cases.

##### 6.2.2.6 Files Structure

project-root/

├── src/

│ ├── lib/

│ │ ├── db.js # Database connection setup

│ │ └── logger.js # Logger configuration

│ ├── controllers/

│ │ ├── userController.js # User-related logic

│ │ ├── productController.js # Product-related logic

│ │ └── ... # Additional controllers

│ ├── middlewares/

│ │ ├── authMiddleware.js # Authentication/Authorization logic

│ │ ├── errorMiddleware.js # Error handling middleware

│ │ └── ... # Additional middlewares

│ ├── models/

│ │ ├── User.js # User model/schema

│ │ ├── Product.js # Product model/schema

│ │ └── ... # Additional models

│ ├── routes/

│ │ ├── userRoutes.js # User-related routes

│ │ ├── productRoutes.js # Product-related routes

│ │ └── ... # Additional route files

│ ├── services/

│ │ ├── userService.js # Service logic for user-related operations

│ │ ├── productService.js # Service logic for product-related operations

│ │ └── ... # Additional services

│ ├── utils/

│ │ ├── constants.js # Application constants

│ │ ├── helpers.js # Utility/helper functions

│ │ └── ... # Additional utilities

│ ├── app.js # App initialization and middleware setup

│ └── server.js # Server setup and entry point

├── tests/

│ ├── unit/ # Unit tests

│ ├── integration/ # Integration tests

│ └── ... # Additional test folders

├── .env # Environment variables

├── .gitignore # Git ignore file

├── package.json # Project dependencies and scripts

├── package-lock.json # Dependency tree lock file

└── README.md # Project documentation

#### 6.2.3 Mobile Development Coding Standards

##### 6.2.3.1 General Guidelines

* **Dart Language Features**: Utilize Dart’s features such as null safety, async/wait, and extension methods.
* **Code Structure**: Organize code into clear, reusable components, and follow the separation of concerns principle.
* **Consistent Naming Conventions**:
  + Use **CamelCase** for class names (e.g., UserProfilePage).
  + Use **lowercase\_with\_underscores** for file names (e.g., user\_profile\_page. dart).
  + Use **camelCase** for variables and methods (e.g., getUserData).

##### 6.2.3.2 Flutter Standards

* **Widgets**: Create reusable widgets and avoid duplicating code. Group related widgets in a dedicated /widgets directory.
* **State Management**: Use a consistent state management solution (e.g., Provider, Riverpod, or Bloc) throughout the application. Avoid using setState in large widgets.
* **Navigation**: Use GetX to define all routes in a centralized location.

##### 6.2.3.3 Dart Standards

* **Type Safety**: Always declare variable types explicitly when possible, to improve readability and catch errors early.
* **Error Handling**: Use try-catch blocks for error handling, especially for asynchronous operations.

##### 6.2.3.4 Dependency Management

* **pubspec.yaml**: Maintain the pubspec.yaml file for managing dependencies. Use specific versions to avoid compatibility issues.
* **Organize Dependencies**: Keep dependencies organized by grouping them logically (e.g., state management, networking).

##### 6.2.3.5 Code Reviews

* **Peer Reviews**: All codes should undergo peer reviews before merging. Use pulls requests to facilitate discussions and feedback.
* **Review Criteria**:
  + Code Quality: Ensure adherence to coding standards and best practices.
  + Functionality: Verify that the code works as intended and meets the requirements.
  + Testing: Check for adequate test coverage and meaningful test cases.

##### 6.2.3.6 Files Structure

project-root/

├── lib/

│ ├── src/ # Main source directory

│ │ ├── assets/ # App assets (e.g., fonts, images)

│ │ │ ├── images/ # Image assets

│ │ │ ├── fonts/ # Font assets

│ │ │ └── ... # Other assets

│ │ ├── core/ # Core modules (global configurations, constants, utilities)

│ │ │ ├── constants/ # App-wide constants

│ │ │ │ ├── app\_colors.dart # Color definitions

│ │ │ │ ├── app\_strings.dart # App-wide strings

│ │ │ │ ├── app\_styles.dart # Text and widget styles

│ │ │ │ └── ... # Additional constants

│ │ │ ├── utils/ # Utility/helper functions

│ │ │ │ ├── validators.dart # Validation logic

│ │ │ │ ├── date\_helpers.dart # Date/time utilities

│ │ │ │ └── ... # Additional utilities

│ │ │ ├── theme/ # Theme and UI styling

│ │ │ │ ├── app\_theme.dart # Theme configuration

│ │ │ │ └── dark\_theme.dart # Dark theme definitions (if applicable)

│ │ │ └── config/ # Global app configurations

│ │ │ ├── api\_config.dart # API endpoint and keys

│ │ │ └── app\_config.dart # General app configuration

│ │ ├── features/ # Feature-specific modules

│ │ │ ├── auth/ # Authentication feature

│ │ │ │ ├── models/ # Data models (e.g., User)

│ │ │ │ │ ├── user.dart # User model

│ │ │ │ ├── screens/ # UI screens

│ │ │ │ │ ├── login\_screen.dart

│ │ │ │ │ ├── register\_screen.dart

│ │ │ │ │ └── forgot\_password\_screen.dart

│ │ │ │ ├── services/ # Business logic and API services

│ │ │ │ │ ├── auth\_service.dart

│ │ │ │ │ └── ... # Other services

│ │ │ │ ├── providers/ # State management (e.g., Riverpod, Provider)

│ │ │ │ │ ├── auth\_provider.dart

│ │ │ │ │ └── ... # Other providers

│ │ │ │ └── widgets/ # Reusable UI components for auth

│ │ │ │ ├── login\_form.dart

│ │ │ │ └── ... # Other widgets

│ │ │ ├── home/ # Home feature

│ │ │ │ └── ... # Similar structure as auth

│ │ │ └── ... # Additional features

│ │ ├── widgets/ # Global reusable widgets

│ │ │ ├── custom\_button.dart # Custom button widget

│ │ │ ├── app\_bar.dart # Custom app bar

│ │ │ └── ... # Additional global widgets

│ │ ├── navigation/ # App navigation

│ │ │ ├── app\_router.dart # Route definitions

│ │ │ ├── navigation\_service.dart # Navigation utilities

│ │ │ └── route\_names.dart # Centralized route name constants

│ │ ├── services/ # Global/shared services

│ │ │ ├── api\_service.dart # API interaction logic

│ │ │ ├── local\_storage.dart # Local storage handling

│ │ │ └── ... # Additional global services

│ │ └── main.dart # App entry point

├── test/ # Unit and widget tests

│ ├── auth/ # Tests for auth feature

│ │ ├── login\_test.dart

│ │ └── ... # Additional tests

│ ├── home/ # Tests for home feature

│ │ └── ...

│ └── ... # Additional tests

├── pubspec.yaml # Dependency configurations

├── pubspec.lock # Dependency lock file

├── analysis\_options.yaml # Static analysis configurations

├── .gitignore # Git ignore file

├── README.md # Project documentation

└── ... # Other project-level files

#### 6.2.4 UI/UX Design Standards for Figma

##### 6.2.4.1 General Guidelines

* **Consistency**: Ensure consistent use of colors, typography, icons, and components throughout the project.
* **Atomic Design Principle**: Structure design components in a modular way (atoms, molecules, organisms, templates, pages) to maximize reusability and scalability.
* **Accessibility**: Prioritize readability, legibility, and inclusive design. Follow **WCAG** guidelines, including color contrast and font sizing.
* **File and Naming Conventions:**
  + Use **Pascal Case** for file names (e.g., ButtonPrimary, UserProfileCard).
  + Name frames based on the screen name and purpose (e.g., Login\_Screen, Product\_Details).
  + Use prefixes for reusable components (e.g., C\_Button, C\_Card).

##### 6.2.4.2 Color and Typography Standards

* **Colors**
  + **Color Styles**: Define a color palette in Figma’s “Styles” for primary, secondary, accent, and neutral colors. Ensure colors meet accessibility contrast requirements.
  + **Naming Convention**: Use descriptive names for colors (e.g., Primary/Blue\_500, Secondary/Green\_300).
* **Typography**
  + **Text Styles**: Define text styles for headings, subheadings, body text, captions, etc.
  + **Naming Convention**: Use consistent and descriptive naming, such as Heading/Medium/24 or Body/Regular/16.

##### 6.2.4.3 Components and Variants and Layout and Grids

* **Component Creation**: Design elements (buttons, icons, form fields) should be created as components. Utilize Figma's **Variants** to handle multiple states (e.g., hover, active, disabled).
* **Component Naming**: Follow the structure Component/Type/State (e.g., Button/Primary/Active, Input Field/Text/Focused).
* **Component Organization**: Organize components in the /Components folder to make them easy to locate and reuse.
* **Grid System**: Define a responsive grid system (e.g., 8pt or 4pt) and set it up in Figma’s “Layout Grid” settings for each frame. Use grids to maintain consistent alignment and spacing.
* **Margins and Padding**: Use Figma’s **Auto Layout** feature to maintain consistent spacing within components and between elements.
* **Icons**: Use vector icons and ensure they are aligned to pixel grid (1x1) for sharpness. Group all icons in a dedicated folder and create them as components to keep them consistent across designs.
* **Illustrations**: Import SVG files for illustrations. Convert to components if they will be reused across multiple screens.

##### 6.2.4.4 Prototyping and Interaction Standards and Accessibility

* **Interactions**: Define interactions (e.g., button clicks, hover effects) in Figma’s prototype settings to demonstrate intended user flows.
* **Transitions**: Use consistent transitions (e.g., ease-in, ease-out) across components. Limit transition time to between **200-300ms** for an optimal user experience.
* **Linking**: Link all interactive elements (buttons, links) to respective screens in the prototype.
* **Contrast Ratios**: Ensure a minimum contrast ratio of **4.5:1** for text and **3:1** for UI components.
* **Alt Text and Descriptions**: Provide all text for icons, images, and illustrations.
* **Keyboard Accessibility**: For interactive prototypes, make sure components can be navigated using keyboard controls.

##### 6.2.4.5 Figma Project Structure

/PROJECT

├── /01 ASSETS

│   ├── COLORS

│   ├── TYPOGRAPHY

│   ├── ICONS

│   └── ILLUSTRATIONS

├── /02 COMPONENTS

│   ├── BUTTONS

│   ├── INPUT FIELDS

│   ├── CARDS

│   └── MODALS

├── /03 LAYOUTS

│   ├── HOME SCREEN

│   ├── PRODUCT PAGE

│   ├── CHECKOUT

│   └── USER PROFILE

└── /04 PROTOTYPES

#### 6.2.5 Penetration Testing Coding Standards

##### 6.2.5.1 General Guidelines

* **Ethical Boundaries**: Ensure all testing activities are authorized and adhere to the project's defined scope.
* **Script Reusability**: Design scripts and tools to be reusable and modular. Prioritize readability and clarity over complexity.
* **Data Security**: Encrypt sensitive data, and securely manage test data and credentials.

##### 6.2.5.2 Scripting Standards

Penetration testing often requires scripting in languages like Python, Bash, and PowerShell. Following standardized practices helps improve reliability and readability.

* **Naming Conventions**
  + **Files**: Use lowercase with underscores for file names (e.g., port\_scan.py).
  + **Variables**: Use descriptive **snake case** names for variables (e.g., target\_ip).
  + **Functions**: Use **snake\_case** for function names (e.g., check\_open\_ports).
  + **Constants**: Define constants in uppercase letters (e.g., DEFAULT\_PORTS).
* **Commenting and Documentation**
  + **Module Docstrings**: Include a brief docstring at the beginning of each script explaining its purpose, author, and date.
  + **Function Docstrings**: Document all functions with input, output, and purpose.
  + **Inline Comments**: Use comments to explain complex logic but avoid over-commenting trivial code.
* **Error Handling**
  + Implement robust error handling to manage unexpected input or network failures:
  + Use try-except blocks in Python or error checks in Bash scripts.
  + Log errors with descriptive messages for troubleshooting.
* **Input Validation and Output Sanitization**
  + **Input Validation**: Ensure all inputs (like IP addresses, URLs) are properly validated to avoid errors and injections.
  + **Output Sanitization**: Sanitize any sensitive data before logging or reporting to prevent unintentional information exposure.
* **Configuration and Parameters**
  + Avoid hardcoding parameters (e.g., IP addresses, credentials); instead, use configuration files or environment variables.
  + Store sensitive data like credentials in secure, encrypted files.

##### 6.2.5.3 Logging and Reporting Standards

* **Logging**
  + **Log Levels**: Use appropriate log levels (e.g., DEBUG, INFO, WARNING, ERROR) to capture relevant events.
  + **File Organization**: Organize logs by date or test stage for easy access. Follow naming conventions like scan\_report\_<date>.log.
  + **Sensitive Data**: Exclude sensitive information from logs.
* **Reporting**
  + **Clear Structure**: Use consistent report structure (e.g., Executive Summary, Findings, Vulnerability Details, Mitigations).
  + **Severity Ratings**: Include severity ratings (e.g., Low, Medium, High) to prioritize issues.
  + **Screenshots and Evidence**: Capture screenshots and record details as evidence for each vulnerability.
  + **Remediation Recommendations**: Provide clear, actionable remediation steps for identified vulnerabilities.

##### 6.2.5.4 Code Testing Standards

* **Unit Testing**: Create unit tests for reusable functions (e.g., network scanning functions).
* **Mocking**: Use mocking for testing interactions with external resources like databases or network services.
* **Automated Testing**: Automate tests where possible, especially for repetitive tasks.

#### 6.2.6 Ai Coding Standards

##### 6.2.6.1 General Guidelines

* **Consistency:**
  + Follow a consistent naming convention: Use camelCase for variables and functions, and PascalCase for class names.
  + Use descriptive and meaningful variable and function names (e.g., getUserRecommendations() instead of getRec()).
* **Modularity:**
  + Write modular, reusable, and self-contained functions.
  + Avoid large functions; split complex logic into smaller, testable units.

##### 6.2.6.2 Language-Specific Standards

**For Python (AI/ML Components):**

* Follow **PEP 8** guidelines:
  + Use 4 spaces for indentation.
  + Limit line length to 79 characters.
  + Use meaningful docstrings for functions and classes
* Type hinting
* Use virtual environments (e.g., **venv** or **conda**) to manage dependencies

### 6.3 Version Control (Git - GitHub)

#### 6.3.1 Repository Structure and Naming Conventions

##### 6.3.1.1 Repository Naming

* Use clear, descriptive names for repositories.
* Examples: ecommerce-platform-backend, ecommerce-frontend, mobile-app-flutter, documentation-repo.

##### 6.3.1.2 File Naming Conventions

* **File Names**: Use **snake\_case** for filenames and be consistent across the repository.
* **Configuration Files**: Use clear and versioned names (e.g., config\_v1.json).

##### 6.3.1.3 Repository Structure

project-repo/

├── README.md # Root documentation for the entire project

├── .gitignore # Git ignore file for the entire repo

├── server/ # Backend (Express)

│ ├── src/

│ │ ├── config/ # Configuration files

│ │ ├── controllers/ # Route handlers

│ │ ├── models/ # Database models

│ │ ├── routes/ # API routes

│ │ ├── services/ # Core business logic

│ │ ├── middlewares/ # Middleware logic

│ │ ├── utils/ # Utility functions

│ │ ├── app.js # Express app setup

│ │ ├── server.js # Server entry point

│ │ └── ... # Additional backend files

│ ├── package.json # Node.js dependencies

│ ├── package-lock.json # Lock file for Node.js dependencies

│ └── .env # Environment variables

├── client/ # React Frontend

│ ├── public/ # Public assets

│ ├── src/

│ │ ├── components/ # Reusable components

│ │ ├── pages/ # Page components

│ │ ├── store/ # Redux/Zustand store logic

│ │ ├── styles/ # Tailwind/global styles

│ │ ├── utils/ # Utility functions

│ │ ├── App.jsx # Main app entry

│ │ ├── index.jsx # React DOM entry point

│ │ └── ... # Additional frontend files

│ ├── package.json # Node.js dependencies

│ ├── package-lock.json # Lock file for Node.js dependencies

│ └── tailwind.config.js # Tailwind CSS configuration

├── flutter-app/ # Flutter Mobile App

│ ├── lib/

│ │ ├── src/

│ │ │ ├── assets/ # App assets

│ │ │ ├── core/ # Global configurations and constants

│ │ │ ├── features/ # Feature modules

│ │ │ ├── widgets/ # Reusable widgets

│ │ │ ├── navigation/ # Navigation logic

│ │ │ ├── services/ # Shared services

│ │ │ ├── main.dart # App entry point

│ │ │ └── ... # Additional app files

│ ├── pubspec.yaml # Flutter dependencies

│ └── pubspec.lock # Lock file for Flutter dependencies

├── chatbot/ # Machine Learning Models

│ ├── data/ # Datasets for training/testing

│ │ ├── raw/ # Raw data

│ │ ├── processed/ # Processed data

│ │ └── models/ # Pre-trained models

│ │ ├── recommendationModel.pkl # Serialized model file

│ │ └── ... # Additional models

│ ├── src/

│ │ ├── dataProcessing.py # Data preprocessing scripts

│ │ ├── modelTraining.py # Model training scripts

│ │ ├── modelInference.py # Model inference logic

│ │ ├── recommender/

│ │ │ ├── collaborative.py # Collaborative filtering logic

│ │ │ ├── contentBased.py # Content-based logic

│ │ │ ├── hybrid.py # Hybrid recommendation system

│ │ │ └── ... # Additional recommendation scripts

│ │ └── ... # Additional ML scripts

│ ├── requirements.txt # Python dependencies

│ └── README.md # ML-specific documentation

├── docs/ # Documentation for the project

#### 6.3.2 Branching Strategy

##### 6.3.2.1 Main Branches

* **main Branch**: The main production-ready branch. Always holds stable, released code.
* **develop Branch**: Integrates new features before merging into the main branch. All work should be merged here first before going to the main branch.

##### 6.3.2.2 Feature Branches

* Create a separate branch for each new feature, bug fix, or improvement, the branch will be created from the develop branch
* Naming convention: “**developer-name/feature-name**”

##### 6.3.2.3 Release Branches

* Used for preparing a release, including final bug fixes and optimizations.
* Naming convention: release/<version-number> (e.g., release/1.0.0).

##### 6.3.2.4 Hotfix Branches

* For emergency fixes to production, use the hotfix/<description> naming convention.
* Merge hotfixes directly to both the main and development branches.

#### 6.3.3 Commit Standards

##### 6.3.3.1 Commit Messages

* **Format**: <type>:<description> or <description> for features
* **Types**:
  + fix: Bug fix
  + docs: Documentation changes
  + style: Code style changes (formatting, etc.)
  + refactor: Code restructuring without functional changes
  + test: Adding or updating tests
  + chore: Maintenance or minor changes
* **Example Commit Messages**:
  + fix: user log in error
  + user login page
  + style: navbar color

##### 6.3.3.2 Committing Frequency

* Commit frequently but logically. Each commit should represent a single unit of work (e.g., a new function or fixed bug).
* Avoid large commits with multiple, unrelated changes.

#### 6.3.4 Pull Requests (PR) and Code Reviews

##### 6.3.4.1 Creating a Pull Request

* Create a pull request (PR) when your feature or fix is ready for integration.
* Name the PR concisely and include a description summarizing the work.
* Link the PR to the relevant issue or task.

##### 6.3.4.2 Review Process

* At least one team member should review each PR before merging.
* Reviewer checks for adherence to standards, potential issues, and opportunities for improvement.
* Provide constructive feedback and request necessary changes.

##### 6.3.4.3 Merge Policies

* Only merge approved PRs into the development branch.
* Use **Squash and Merge** to keep a clean commit history if appropriate.

##### 6.3.4.4 Best Practices

* **Stay in Scope**: Limit changes in each branch to the specific task.
* **Use .gitignore**: Add unnecessary files (e.g., temporary, build artifacts) to .gitignore.
* **Regular Syncing**: Frequently pull changes from the main or development branch to keep your branch up-to-date and resolve conflicts early.

#### 6.3.5 Documentation and Collaboration

* Project purpose, Installation and setup instructions, Usage examples, Contribution guidelines
* Conduct regular team check-ins to discuss progress, PRs, blockers, and upcoming changes to the main or development branches.

### 6.4 Test Plan

#### 6.4.1. Introduction

* **Purpose**: This document outlines the security testing strategy for the e-commerce application to identify vulnerabilities and ensure data protection.
* **Scope**: Security testing will cover web application security, data security, and network security.

#### 6.4.2. Objectives

* Identify and mitigate security vulnerabilities.
* Ensure compliance with security standards and regulations (e.g., GDPR, PCI DSS).
* Protect sensitive user data, including payment information and personal details.

#### 6.4.3. Testing Methodologies

* **Static Analysis**: Review code for security vulnerabilities without executing the program.
* **Dynamic Analysis**: Test the application in its running state to identify runtime vulnerabilities.
* **Penetration Testing**: Simulate attacks to identify security weaknesses and assess the effectiveness of defenses.
* **Risk Assessment**: Evaluate the potential impact and likelihood of identified vulnerabilities.

#### 6.4.4. Security Testing Types

* **Authentication Testing**: Verify the effectiveness of user authentication mechanisms.
* **Authorization Testing**: Ensure users can only access resources and functions for which they have permissions.
* **Input Validation Testing**: Check for vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).
* **Session Management Testing**: Evaluate the security of session tokens and management processes.
* **Data Protection Testing**: Ensure sensitive data is encrypted and securely stored both at rest and in transit.
* **Error Handling Testing**: Assess how the application handles errors and whether sensitive information is exposed.

#### 6.4.5. Tools and Technologies

* **Static Analysis Tools**: SonarQube, Fortify
* **Dynamic Analysis Tools**: OWASP ZAP, Burp Suite
* **Penetration Testing Tools**: Metasploit, Nessus
* **Vulnerability Scanners**: Acunetix, Qualys

#### 6.4.6. Test Cases

* **Authentication**
  + Test for brute force protection on login.
  + Verify multi-factor authentication implementation.
* **Authorization**
  + Attempt to access restricted pages without proper permissions.
* **Input Validation**
  + Test for SQL injection on all input fields.
  + Check for XSS vulnerabilities by inputting malicious scripts.
* **Session Management**
  + Verify session expiration after inactivity.
  + Test for session fixation vulnerabilities.
* **Data Protection**
  + Confirm SSL/TLS implementation for secure data transmission.
  + Check for proper encryption of sensitive data in the database.

#### 6.4.7. Roles and Responsibilities

* **Security Tester**: Responsible for executing tests and reporting findings.
* **Development Team**: Address vulnerabilities and implement fixes.
* **Project Manager**: Ensure compliance and coordinate the testing process.

#### 6.4.8. Testing Schedule

* **Planning Phase**: [Start Date] to [End Date]
* **Execution Phase**: [Start Date] to [End Date]
* **Reporting Phase**: [Start Date] to [End Date]

#### 6.4.9. Reporting and Documentation

* Document all findings, including vulnerabilities, severity levels, and suggested mitigations.
* Provide a final report summarizing the security testing outcomes and recommendations for improvement.

#### 6.4.10. Follow-Up Actions

* Schedule remediation meetings to discuss vulnerabilities.
* Plan for retesting after issues have been addressed.
* Establish ongoing security monitoring and periodic testing.

### 6.5 Deployment Plan

#### 6.5.1. Setup Environment

* **Branching Strategy**: Implement Git branching (e.g., feature branches, develop, and main) to manage different environments.
* **Environment Configuration**: Use environment variables for secure configuration of secrets (API keys, database URIs) in development, staging, and production.

#### 6.5.2. CI/CD Pipeline

Set up a CI/CD pipeline to automate the build, testing, and deployment process. (GitHub Actions)

#### 6.5.3. Back-End Deployment (Node.js)

* **Hosting**: Use services like AWS EC2, DigitalOcean, or Heroku.
* **Database**: MongoDB Atlas (cloud-hosted MongoDB)
* **Steps**:
  + Set up Docker to containerize the application if scalability and portability are essential.
  + Configure the CI/CD pipeline to deploy the Node.js server after successful tests.
  + Implement process management tools like PM2 for load balancing and managing server uptime.

#### 6.5.4. Web Front-End Deployment (React.js)

* **Hosting**: Vercel, Netlify, or AWS Amplify for React applications.
* **Steps**:
  + Automate deployment through your CI/CD pipeline to the chosen platform, linking it to the main branch or the production-ready branch.
  + Enable HTTPS and custom domain setup for production deployment.
  + Integrate the back-end API endpoints within the React app using environment variables.

#### 6.5.5. Mobile Front-End Deployment (Flutter)

* **Distribution Channels**: Use Google Play Console for Android and App Store Connect for iOS.
* **Steps**:
  + Set up CI/CD for Flutter with tools like GitHub Actions.
  + Configure build pipelines for Android (.apk ) and iOS (.ipa) to streamline updates.
  + Ensure the app connects to the correct environment (staging/production) and includes proper API keys.

#### 6.5.6. Monitoring & Maintenance

* **Logging & Monitoring**: Set up error tracking and performance monitoring using services like New Relic, Sentry, or LogRocket.
* **Alerting**: Configure alerts for critical issues in real-time, covering both server health (back end) and client-side errors (front end)

### 6.6 User Manual

#### 6.6.1. Introduction

**Overview**

This manual provides instructions for using the platform, detailing various features available to each user role: Customer, Merchant, and Admin. The goal of the platform is to provide a secure, user-friendly environment for e-commerce transactions, product management, and customer support.

**Purpose**

This document guides platform users through the system's functionalities, detailing each feature according to user permissions.

**Platform Roles**

* **Customer**: Primarily for purchasing products.
* **Merchant**: For businesses managing product listings.
* **Admin**: Full control over the platform and its settings.

#### 6.6.2. User Roles and Permissions

**Customer**

* **Account Management**: Create and manage personal accounts.
* **Product Interaction**: Browse, filter, and search products; add items to the cart or Wishlist.
* **Orders and Reviews**: Place orders, track order history, and review purchased items.

**Merchant**

* **Product Management**: Add, edit, and delete own products; set prices, quantities, and availability.
* **Analytics**: Access to sales metrics and reviews to assess product performance.
* **Communication**: Interact with customers through reviews.

**Admin**

* **Platform Management**: Create, edit, and delete user accounts, products, categories, and brands.
* **System Settings**: Adjust platform settings, manage payments and refunds, monitor security, and oversee compliance.

#### 6.6.3. Authentication and Authorization

**User Registration**

* Users can register with a unique email and secure password.
* Social login options are available for Google.

**Login Process**

* Login requires valid credentials, with password protection measures in place.
* Multi-factor authentication is recommended for added security.

**Profile Management**

* Users can update their profile information, change passwords, and manage account preferences.

**Password Recovery**

* Users may recover passwords securely via email-based reset links.

**Role-Based Access Control (RBAC)**

* Each user has access to platform features based on their role permissions.

**Token-Based Authentication**

* Authenticated users receive tokens for secure, consistent access.

**Data Encryption and Input Validation**

* All sensitive data is encrypted, and inputs are validated to secure the platform.

#### 6.6.4. Features by Role

**Product Management**

* **Admin**: Manages all products, categories, brands, and stock levels across the platform.
* **Merchant**: Adds and manages products for their specific brand, including pricing, categories, and stock levels.

**Product Search and Filtering**

* Users can search by keywords, descriptions, or brands and use filters for category, price, and other attributes.

**Shopping Cart**

* **Product Addition and Removal**: Customers can add/remove products in their shopping cart, with quantity adjustments and real-time cart total updates.
* **Cart Persistence**: Carts are saved across sessions.
* **Cart Summary**: A detailed overview of products, prices, and total cost.

**Wishlist Functionality**

* Customers can add products to a Wishlist and seamlessly move them to the cart.

**Checkout and Payment**

* **Shipping Information**: Collects details for delivery, with options for shipping estimates.
* **Payment Options**: Accepts credit/debit cards, digital wallets, and cash on delivery.
* **Order Confirmation and Tracking**: Provides an order summary and tracking link.
* **Secure Payment Integration**: Integrates with secure payment gateways for data security.

**Reviews and Ratings**

* Customers can review purchased products and rate them. Merchants can reply to customer reviews.
* Reviews are moderated by Admin to maintain content quality.

**Customer Support and Profile Information**

* **Profile Management**: Customers can edit personal details and manage their password.
* **Customer Support**: Provides channels for email, phone, and live chat, with an FAQ section for common issues.

# Chapter 7



## Conclusion

### 7.1 Summary

This project aims to establish a comprehensive online marketplace exclusively for Egyptian brands and stores, providing a dedicated platform for local businesses to reach a wider customer base. The platform will offer a unique shopping experience that highlights high-quality Egyptian products, with an emphasis on supporting local businesses, promoting Egyptian culture, and driving economic growth. The marketplace is designed to facilitate the growth of local businesses by enabling them to showcase their products, offer exclusive discounts, and engage with consumers. With features like AI-powered search, easy navigation, secure payments, and customer support, the platform promises to meet the needs of both consumers and merchants, fostering a sense of community and national pride.

### 7.2 Impact Assessment

The impact of this project is far-reaching, with significant benefits at various levels:

* Economic Development: By empowering local businesses to grow, the platform will boost consumer spending, generate jobs, and contribute to the growth of the local economy. The online marketplace will also create opportunities for Egyptian products to be exported, expanding the reach of local businesses beyond domestic borders.
* Social Impact: The project will empower local businesses, especially small and medium enterprises (SMEs), to thrive in a competitive digital market. It will also preserve traditional craftsmanship by supporting local artisans, ensuring that Egyptian heritage and culture are preserved for future generations.
* Technological Advancement: The platform will serve as a catalyst for digital transformation in Egypt's retail sector, encouraging innovation, improving logistics efficiency, and streamlining supply chains. It will position Egypt as a leader in e-commerce innovation within the region.
* Global Impact: As Egypt’s visibility grows on the international e-commerce stage, the project will help position the country as a global player in the market. The platform's success could attract foreign investment, further boosting Egypt's position in the global economy.

### 7.3 Future Work

Looking ahead, the future of the platform includes several key developments and opportunities for growth:

* Expansion of Product Categories: As the marketplace gains traction, additional product categories can be added, catering to a wider audience and meeting the evolving needs of consumers.
* Enhancing AI Features: The AI-powered search functionality can be further developed to include more advanced features, such as personalized product recommendations, improving the user experience and increasing conversion rates.
* Global Expansion: As the platform establishes itself in Egypt, there is potential to expand into other regions and attract international customers, helping Egyptian brands gain a global presence.
* Sustainability Initiatives: Future work could also include incorporating eco-friendly practices, such as supporting sustainable brands and offering environmentally conscious shipping options.

In conclusion, this project has the potential to transform the e-commerce landscape in Egypt, providing significant economic, social, and technological benefits. Through continuous development and adaptation, it will help local businesses flourish and increase the visibility of Egyptian culture and products on the global stage.

### 7.4 Benefits

#### ****7.4.1 For Local Businesses****

* **Increased Visibility:** Provides a dedicated platform to showcase high-quality Egyptian products, allowing businesses to reach a wider audience.
* **Enhanced Sales:** By connecting brands with a larger customer base, the platform helps boost sales and supports revenue growth.
* **Cost-Effective Marketing:** Offers an affordable way for small businesses and artisans to market their products without significant advertising investments.
* **Brand Empowerment:** Enables brands to manage their product listings and promotions independently, fostering growth and self-sufficiency.

#### ****7.4.2 For Consumers****

* **Access to Diverse Products:** Features a wide range of Egyptian products, from traditional crafts to modern goods, in one convenient marketplace.
* **Personalized Shopping Experience:** AI-powered recommendations and advanced search capabilities create a seamless and tailored user experience.
* **Exclusive Rewards:** Provides customers with points, promo codes, and discounts, enhancing shopping satisfaction and engagement.
* **Support for Student Businesses:** Empowers student entrepreneurs by allowing them to showcase and sell their products on the platform, with added benefits for those registering with university emails.

#### ****7.4.3 For the Economy and Society****

* **Economic Growth:** Stimulates local economic activity by promoting consumer spending on domestic goods and services.
* **Job Creation:** Generates employment opportunities across e-commerce, logistics, and support services.
* **Cultural Preservation:** Showcases Egyptian heritage and craftsmanship, fostering national pride and preserving traditional skills.
* **Community Development:** Strengthens the bond between businesses and consumers, creating a collaborative ecosystem.

#### ****7.4.4 For Egypt's Global Presence****

* **International Visibility:** Positions Egypt as a leader in e-commerce, showcasing the country’s innovative approaches to supporting local industries.
* **Export Opportunities:** Opens doors for Egyptian products to reach international markets, elevating their global reputation.
* **Foreign Investment Potential:** Attracts investors by demonstrating the growth and potential of Egypt’s e-commerce and technology sectors.

#### ****7.4.5 Technological Advancements****

* **Cutting-Edge Features:** Implements modern technologies like AI-driven recommendations and chatbot support to enhance user experiences.
* **Digital Transformation:** Supports the digital transformation of Egypt’s retail industry, making it more competitive and efficient.
* **Platform Analytics:** Offers valuable insights for businesses, enabling them to make data-driven decisions.

### 7.5 References

* Git: <https://git-scm.com/doc>
* GitHub: <https://docs.github.com/>
* HTML: <https://developer.mozilla.org/en-US/docs/Web/HTML>
* CSS: <https://developer.mozilla.org/en-US/docs/Web/CSS>
* JavaScript: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
* React: <https://react.dev/>
* Tailwind CSS: <https://tailwindcss.com/docs>
* Daisy UI: https://daisyui.com/
* Node.js: https://nodejs.org/docs
* Express: https://expressjs.com/en/guide
* MongoDB: <https://www.mongodb.com/docs/>
* JWT (JSON Web Token): https://jwt.io/introduction/
* Mongoose: https://mongoosejs.com/docs/
* Redis: https://redis.io/docs/
* Cloudinary: <https://cloudinary.com/documentation>
* Restful Apis: https://restfulapi.net/
* Postman: https://www.postman.com/
* Redux: https://redux.js.org/introduction/getting-started
* Zustand: <https://docs.pmnd.rs/zustand/introduction>
* Flutter: https://docs.flutter.dev/
* Dart: <https://dart.dev/guides>
* Figma: https://help.figma.com/
* Notion: https://www.notion.so/help
* Draw.io (Diagrams.net): https://www.diagrams.net/doc/
* Google Drive: <https://support.google.com/drive/>
* Visual Studio Code: <https://code.visualstudio.com/docs>
* Android Studio: <https://developer.android.com/studio/>
* Discord: https://support.discord.com/hc/en-us