``````````````````````

**The Egyptian E-Learning University**
**Faculty of Computers & Information Technology**

# E-commerce Mobile Application

**GRADUATION PROJECT**

**PREPARED BY**

| | |
|---|---|
| Mohamed Mostafa Ahmed | ID: 20-01478 |
| Hazem Khamis Mahmoud | ID: 20-01577 |
| Mazen Osama Mohamed | ID: 20-00261 |
| Maryam Nassar Mohamed | ID: 19-01370 |
| Joy Emad Edwarda Hanna | ID: 20-00206 |
| Melcia Nabil Ghobrial | ID: 20-00948 |
| Silvana Sameh | ID: 20-01111 |

**SUPERVISED BY**

**Dr. Safi Ibrahim**
Professor of Computer Engineering and Information Technology Egyptian E-Learning University

**Eng. Rana Tamer**
Demonstrator, Department of Information Technology at Egyptian E-Learning University

**A Graduation Project Report Submitted for the Partial Fulfillment of the Requirements of the B.S.C Degree**

**Alexandria 2023-2024**
``````````````````````

Acknowledgement

In completing this graduate project I have been fortunate to have help, support and encouragement from many people. I would like to acknowledge them for their cooperation.

First and foremost deeply thankful to Professor Dr. Safi Ibrahim, for her wonderful guidance during this project work in field of Computer Science, at Egyptian E-Learning University .

I am also thankful for her continuous feedback and encouragement throughout this project work. Her broad knowledge and hardworking attitude has left me with very deep impressions and they will greatly benefit me throughout my life. I would like to thank my project Readers Eng.Rana tamer for her support throughout this project work.

Abstract

In today’s fast-changing business environment, it’s extremely important to be able to respond to client needs in the most effective and timely manner.

If your customers wish to see your business online and have instant access to your products or services.

Online Shopping is a lifestyle e-commerce mobile application, which retails various fashion and lifestyle products (Currently Men’s Wear).

This project allows viewing various products available enables registered users to purchase desired products and also can place order by using Cash on Delivery (Pay Later) option. This is a project with the objective to develop a basic application where a consumer is provided with a shopping cart application and also to know about the technologies used to develop such an application.

The business-to-consumer aspect of product commerce (e-commerce) is the most visible business use of the World Wide Web. The primary goal of an e-commerce site is to sell goods online.

This project deals with developing an e-commerce website for Online Product Sale. It provides the user with a catalog of different product available for purchase in the store.

In order to facilitate online purchase a shopping cart is provided to the user.

TABLE OF CONTENTS

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

- HISTORY
- MOTIVATION
- RELATED WORK
- PROBLEM STATEMENT
- PROBLEM SOLUTIONS
- PROJECT PHASES

Introduction

E-commerce is fast gaining ground as an accepted and used business paradigm. More and more Business houses are implementing mobile application providing functionality for performing commercial Transactions over the application.

It is reasonable to say that the process of shopping on the application is Becoming common place.

The objective of this project is to develop a general purpose e-commerce store where product like clothes can be bought from the comfort of home through the Internet. However, for Implementation purposes, this paper will deal with an online shopping for clothes.

An online store is a virtual store on the Internet where customers can browse the catalog and Select products of interest. The selected items may be collected in a shopping cart.

At checkout Time, the items in the shopping cart will be presented as an order. At that time, more Information will be needed to complete the transaction.

Usually, the customer will be asked to fill or select a billing address, a shipping address, a shipping option, and payment information Such as credit card number.

An e-mail notification is sent to the customer as soon as the order Is placed.

History

The first recorded instance of e-commerce was in 1971 when students at Stanford University used the ARPANET (the precursor to the internet) to buy and sell marijuana. This early experiment was short-lived, however, as the university soon shut down the operation.

It wasn't until 1979 that the first commercial transaction was made online. This transaction involved the sale of a Sting album from a company called Net Market.

1982: The first e-commerce company launches

Boston Computer Exchange launched in 1982. It was an online marketplace for people interested in selling their used computers.

The 1990s saw the emergence of e-commerce as we know it today, with companies such as Amazon and eBay leading the way. These companies allowed individuals to buy and sell goods online, creating an entirely new market for businesses to tap into.

1992: The first e-commerce marketplace launches

Book Stacks Unlimited was launched in 1992 by Charles M. Stack. Originally it was a dial-up bulletin board, but it was later launched as an online marketplace from the Books.com domain

1995: Amazon launches

Jeff Bezos launches the business to become the world’s largest e-commerce marketplace. It was initially started as an e-commerce platform for books. That same year the famous security protocol SSL was launched, which helped make online sales more secure.

1999: Alibaba launches

Alibaba Online launched as an online marketplace. It had more than $25 million in funding and by 2001 it was profitable. It quickly became the largest online e-commerce platform for B2B, C2C, and B2C transactions.

Motivation

Global Market

Having an e-commerce website designed will give you the opportunity to reach out and offer your products and services to customers around the world, regardless of the distance and time zone.

Open All Hours

Customers will enjoy the round-the-clock convenience of being able to purchase what they want and when they want it, and you won't lose out on sales with an online shop that is open 24 hours a day, 7 days a week, compared to regular store hours.

Broaden Your Brand

Diversify your product range and increase sales with intangible goods like eBooks that can be sold on your online store. Customers can also place orders for personalized items which makes it easy to provide all the necessary information to suit their requirements.

Higher Conversion Rate

Potential buyers are more likely to make purchases when they can place their order instantly, rather than waiting for a regular store to open.

Marketing Through Social Media & Search Engines

With good search engine optimization, your website will appear in the **top results of search engines** such as Google, and working with an **SEO services** provider will allow you to understand how your target market is searching online. Using this information, you can then develop an online strategy that will expose your website to relevant, highly motivated customers.

Also, social media websites like Facebook and Twitter will provide you with a platform to build trust with your customers through reviews and ratings, as well as keeping them informed with regular posts about your products and offers. Keep your customers engaged with competition and **shareable content** to drive even more traffic to your website.

More Convenient

These days' customers are appreciating the convenience of online shopping more and more. Rather than spending hours searching in physical stores, people are now making purchases over the Internet during breaks at work, before the school run and in harsh weather conditions, all the times that would be difficult to make a trip to a shop on the high street. The presence of the market has made it easier for everyone including busy parents with pushchairs and wheel-chair users to shop around and make decisions on purchasing without the hassle of being in an environment that is not fully accessible

Related Work

Amazon (amazon.com):

Amazon is a global e-commerce giant that started as an online bookstore. It has expanded to offer a wide range of products, including electronics, apparel, and more. Amazon also provides services like Amazon Prime, Amazon Web Services (AWS), and streaming through Amazon Prime Video.

Alibaba (alibaba.com):

Alibaba is a Chinese multinational conglomerate specializing in e-commerce, retail, internet, and technology. It operates various platforms, including Alibaba.com for global wholesale trade, Taobao for C2C online shopping, and T-mail for B2C online retail.

EBay (ebay.com):

EBay is an online marketplace that facilitates consumer-to-consumer and business-to-consumer sales through auctions and fixed-price listings. Users can buy and sell a wide variety of goods, including electronics, collectibles, fashion items, and more.

Wayfair (wayfair.com):

Wayfair is an e-commerce company that specializes in home goods, furniture, decor, and more. It connects buyers with a vast selection of items for every room in the house, often offering a wide range of styles and price points.

Problem Statement and Problem solution

Customer Experience

Customer experience or user experience is key to a successful e-commerce website. Shoppers expect a similar if not same experience as one they would get in a brick and mortar store. The flow of the website, the segmentation of the website and the retail personalization of products based on the shopper's preferences are imperative.

Solution:

There are several ways to improve the user experience. The most important would be to have a clean and simple website so that shoppers can navigate through easily. The next point would be to have clear CTAs (call to action) so that the shopper knows exactly what to do. Here is a post that shows you 10 ways to improve user experience.

Product Return & Refund Policies

According to ComScore, more than 60% of online shoppers say that they look at a retailer's return policy before making a purchase. When an e-commerce site says "no returns or refunds" it makes a shopper nervous and less likely to trust the retailer. When shopping online, customers want the flexibility of making a mistake that doesn't cost them.

Solution:

Customer satisfaction is the most important factor for any retailer. Therefore having a flexible return and refund policy not only helps with customer satisfaction with it also helps with customers making purchases without being nervous.

Project Phases

A. Research

B. Analysis

C. Prototyping

D. Implementation

E. Development

Chapter 2

OVERALL DESCRIPTION

- PRODUCT PRESPCTIVE
- PRODUCT FUNCTIONS
- USER CHARACTERISTICS
- ASSUMPTIONS AND DEPENDENCIES

Overall description

This section will give you an overview about the application. This app allows you to use your mobile for making shopping online, browse product catalogs, create wish lists, add items to a cart, and complete purchases, shipping and let users review and rate products

Product Perspective

The suggested framework is a solution for the online purchase/sale of goods.

Product functions

A homepage for the customer where can view new products

An option of searching for products on the homepage

Mark products as favorites or put them in the cart and order them.

A customer may post reviews of a product.

User characteristics

Simple and Clear Design

First things first, an e-commerce app’s interface should be super convenient, simple, and straightforward. The home screen contains feature a clear layout and an easy navigation system so that users can get to their desired section hassle-free. Each button, menu, or content should be minimalist and well-organized. The UI should be intuitive with a simple color scheme and sufficient spacing.

Searching

The search bar is essential to help users directly find the products they’re interested in. a number of possible suggestions will come up when the user types in the first few letters. This will save users’ time

Wishlist

Just like in a traditional brick-and-mortar store, digital customers often browse through a range of products before making a final purchase. They may also be low on money when looking through your app, and may only be ready to make a purchase later. To make their experience easier and to provide them an opportunity to come back to the products they liked,

Assumptions and dependencies

That are:-

a) The coding should be error free.

b) The system should be user friendly so that it is easy to use for the users.

c) The system should have more capacity and provide fast access to the database.

d) The system should provide search facility and support quick transactions.

e) The application should running twenty four hours a day.

f) User must have their correct usernames and passwords to enter into their online accounts and do actions.

Chapter 3

Software Design

- ➢ FUNCTIONAL REQUIREMENT
- ➢ NON FUNCTIONAL REQUIREMENT
- ➢ USE CASE DIAGRAM
- ➢ ACTIVITY DIAGRAM
- ➢ CLASS DIAGRAM
- ➢ SEQUENCE DIAGRAM
- ➢ USER EXPERIENCE

Functional Requirement

1) User Registration and Authentication:

Users should be able to register, login, and logout securely.

2) Product Browsing:

Display a catalog of products with relevant details (name, price, description, images, ratings, reviews, etc.).

3) Product Details:

Allow users to view detailed information about each product

4) Shopping Cart:

Enable users to add items to their shopping cart, view the cart contents, update quantities, and remove items.

5) User Profile:

Enable users to manage their profiles, including personal information.

6) Wishlist and Favorites:

Allow users to add products to their Wishlist or favorites list for future reference.

Non Functional Requirement

1) Security:

Authentication and Authorization: The application should implement strong authentication mechanisms to verify the identity of users and ensure that they have appropriate permissions to access sensitive features and data.

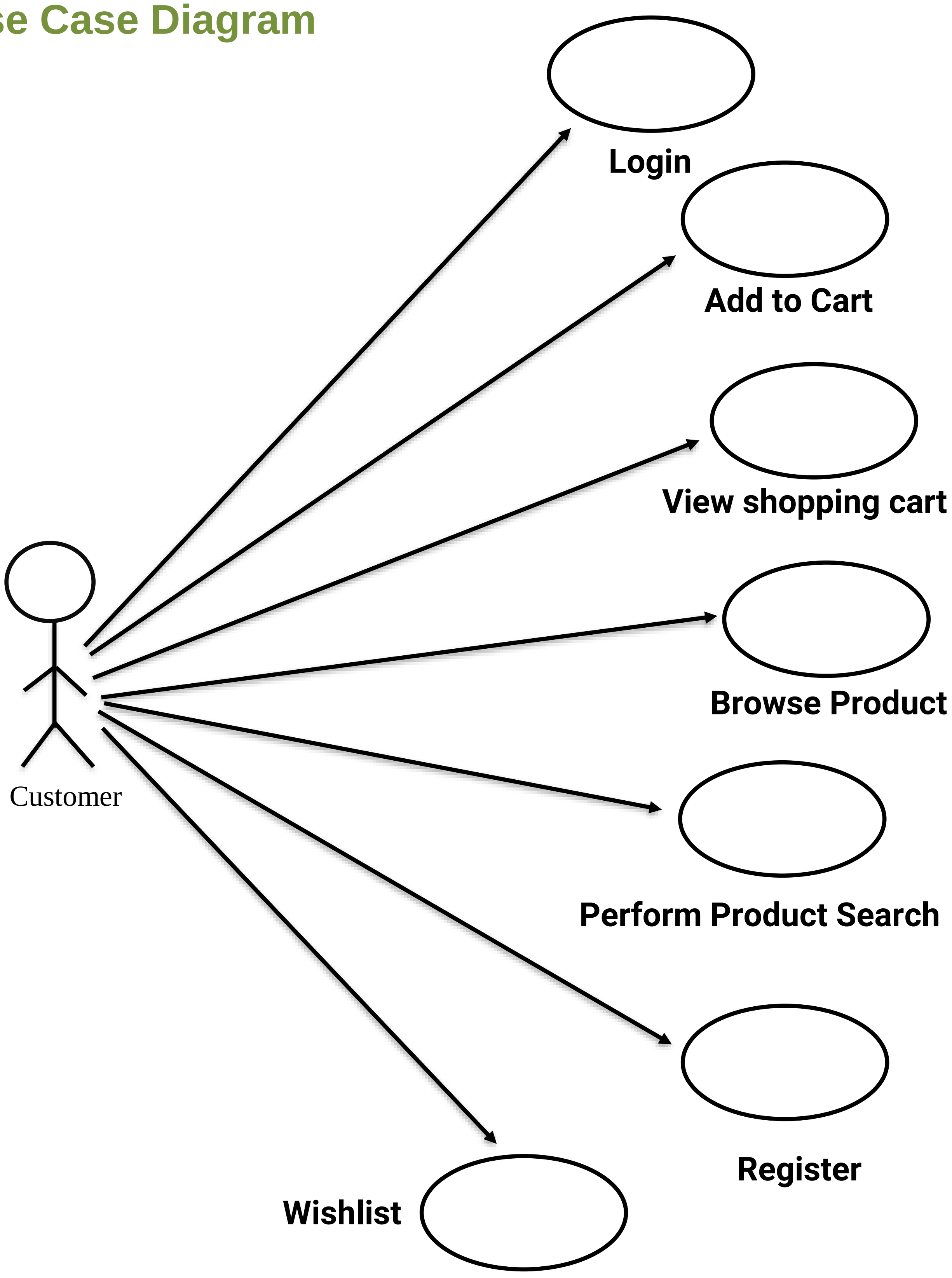
2) Usability:

Responsive Design: The application should be optimized for various screen sizes and device types, ensuring that it functions effectively on smartphones and tablets with different resolutions and aspect ratios.

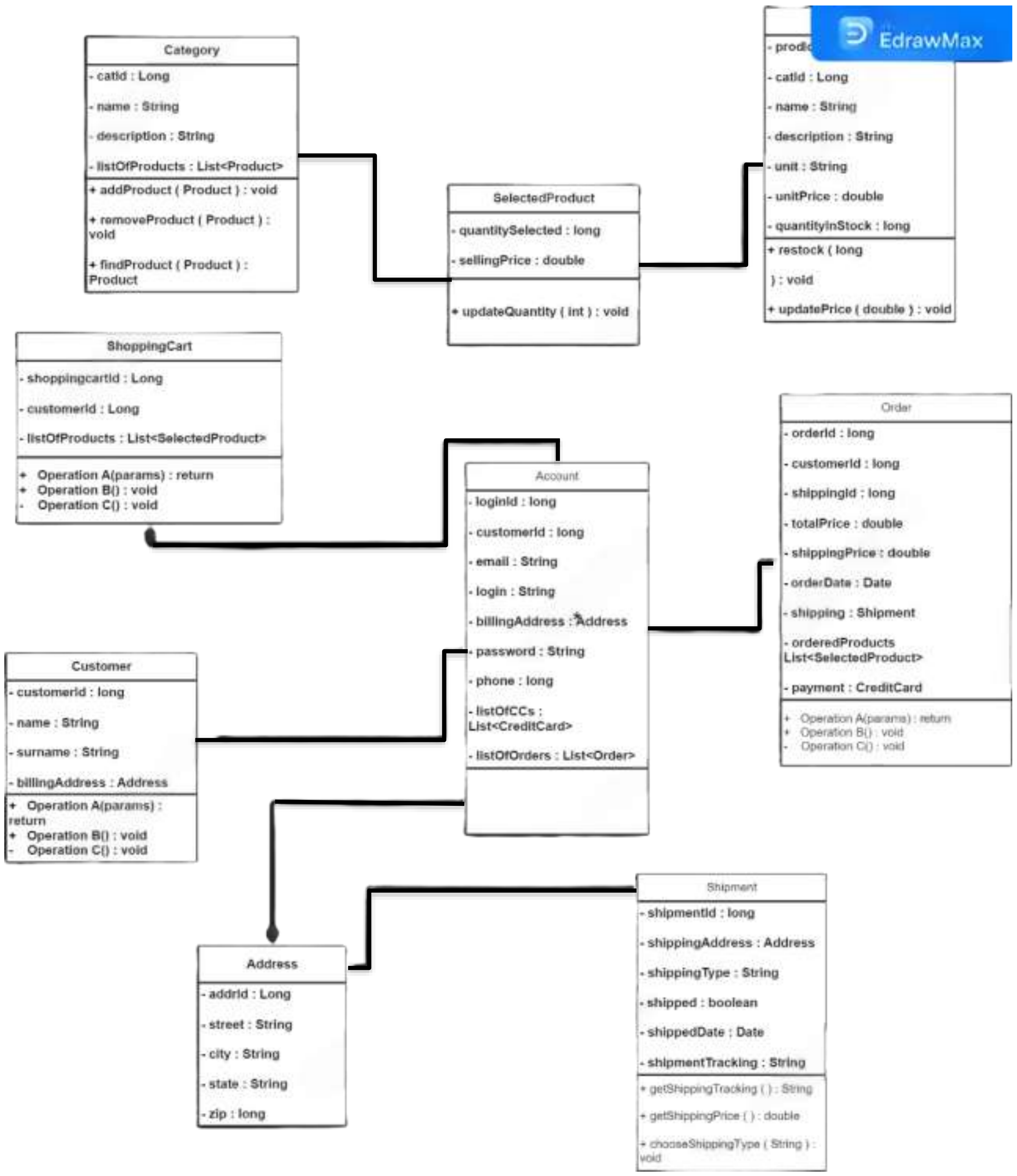
3)Performance:

Response Time: The application should respond to user interactions within an acceptable time frame, ensuring that actions such as product search, loading product details.

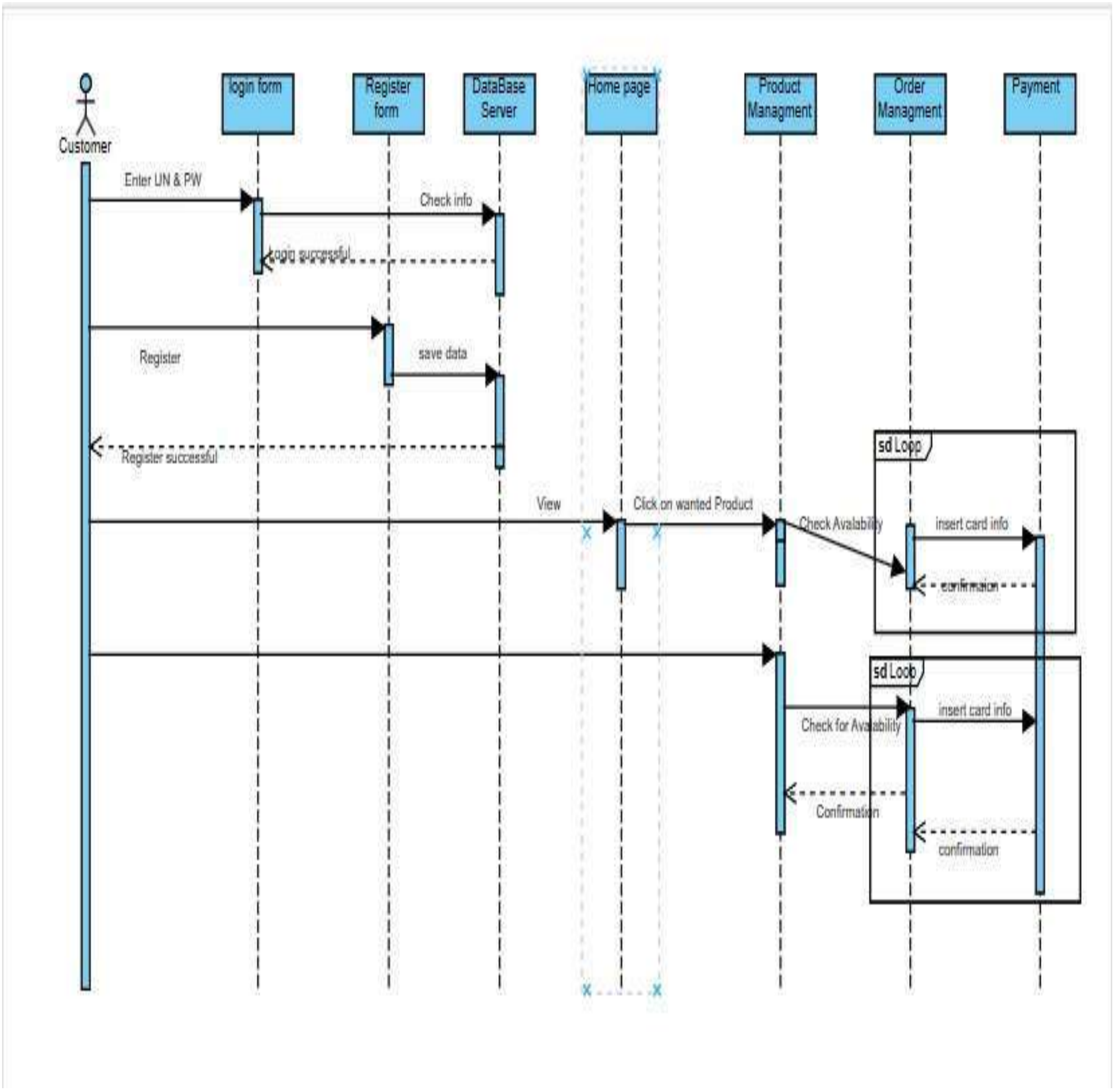
Use Case Diagram



Class Diagram

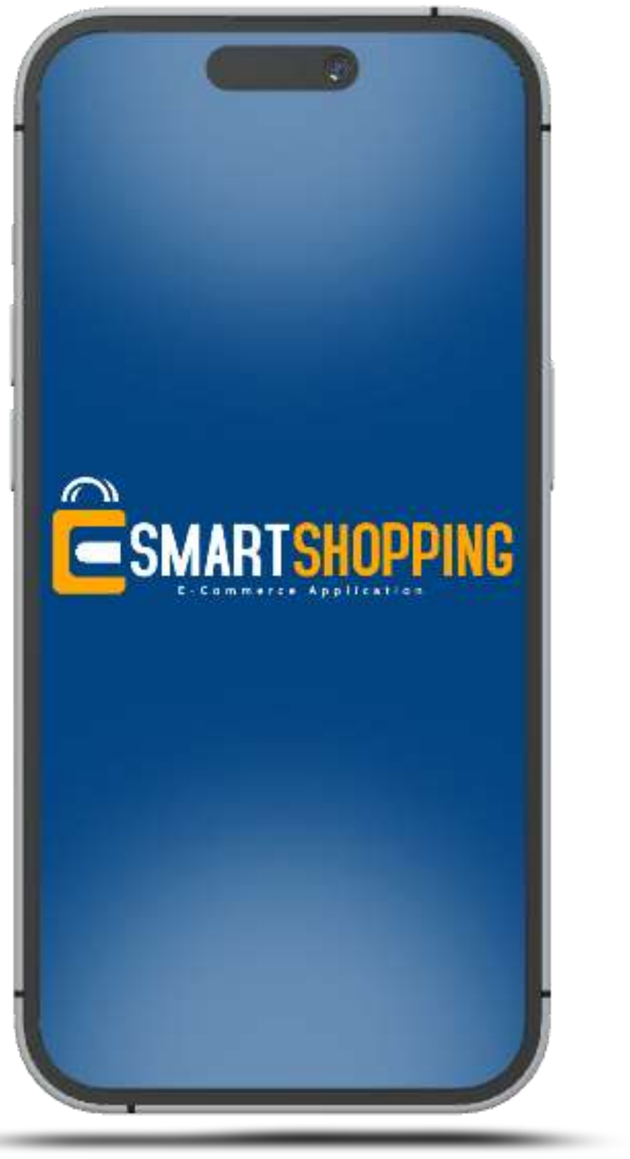


Sequence Diagram



User Experience

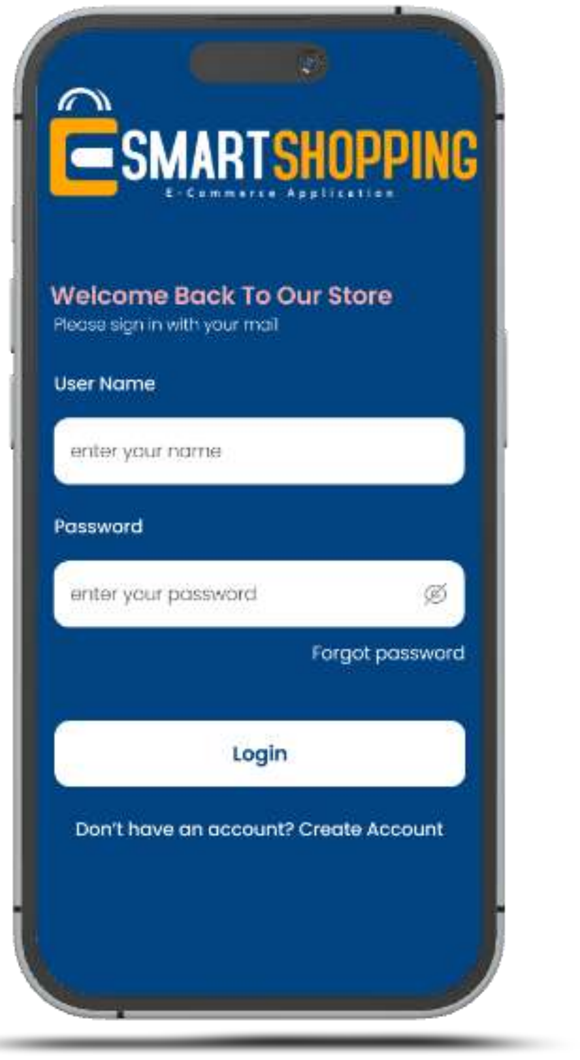
Splash Screen



Splash Screen used for establishes brand identity
And recognition, Provides visual feedback to users that the application is loading and Enhances user
Experience through branding and entertainment.

User Experience

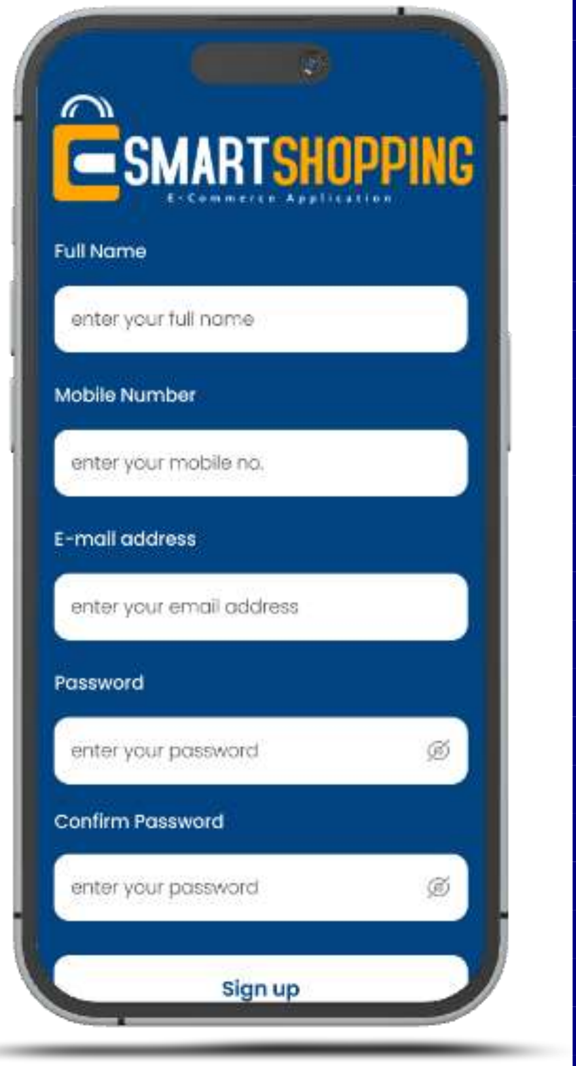
Login Screen



This page is a login page that allows the user to enter The app by entering his name, email address and Password if he has an account in the app

User Experience

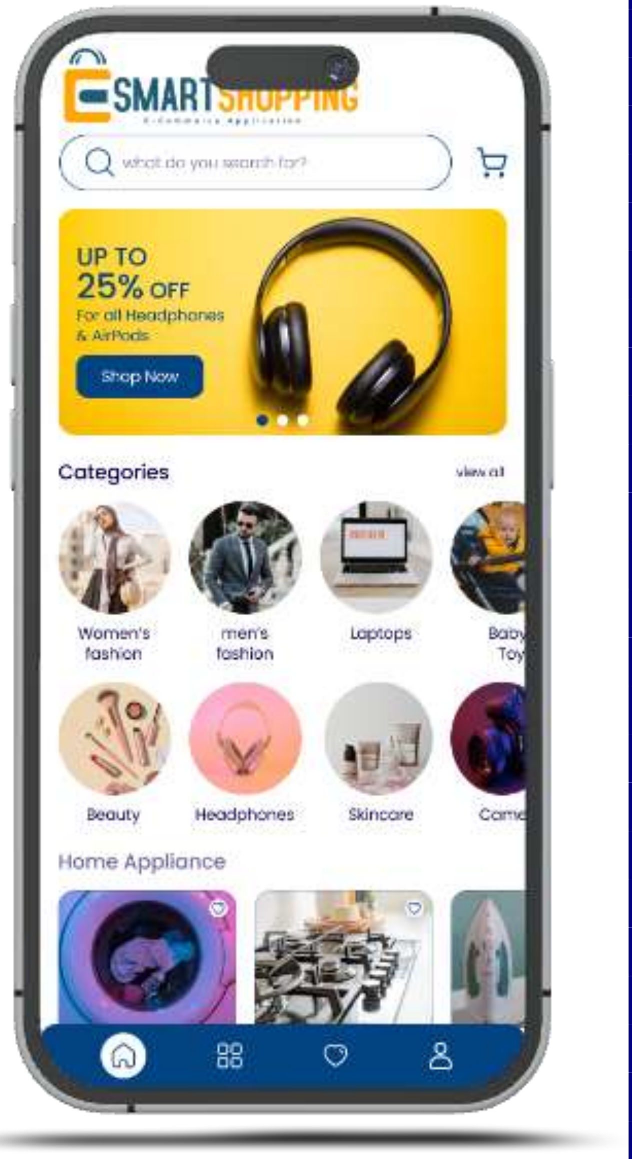
Register Screen



This page is designed for if you don’t already have an Account in the app so you will write your full name, Mobile phone, e-mail and password when you finish You click sign up and this way you were successfully Done.

User Experience

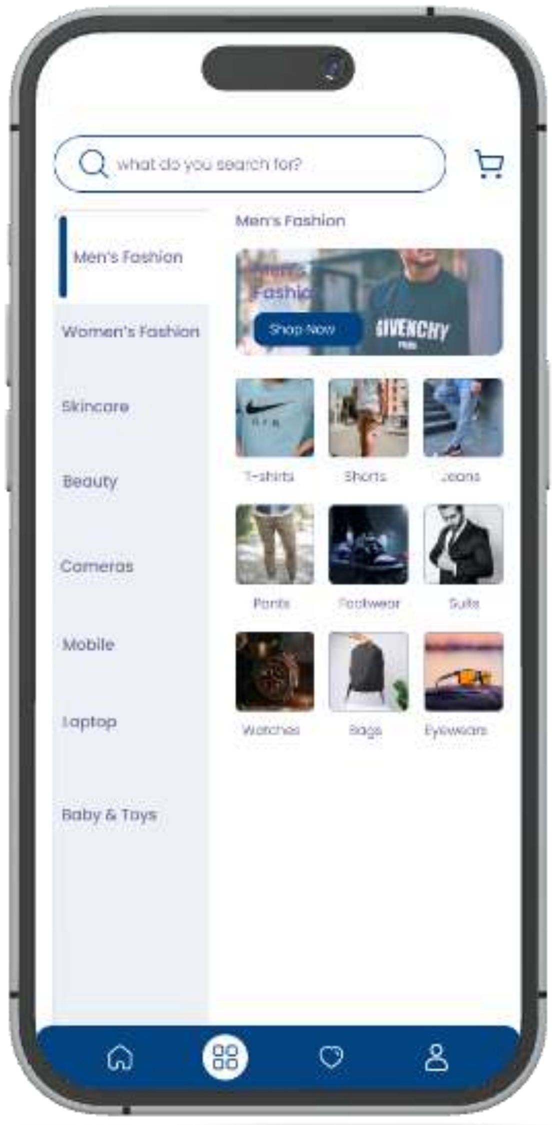
Home Screen



The e-commerce app home screen features featured products, promotions, and search options to provide a visually appealing and personalized shopping experience.

User Experience

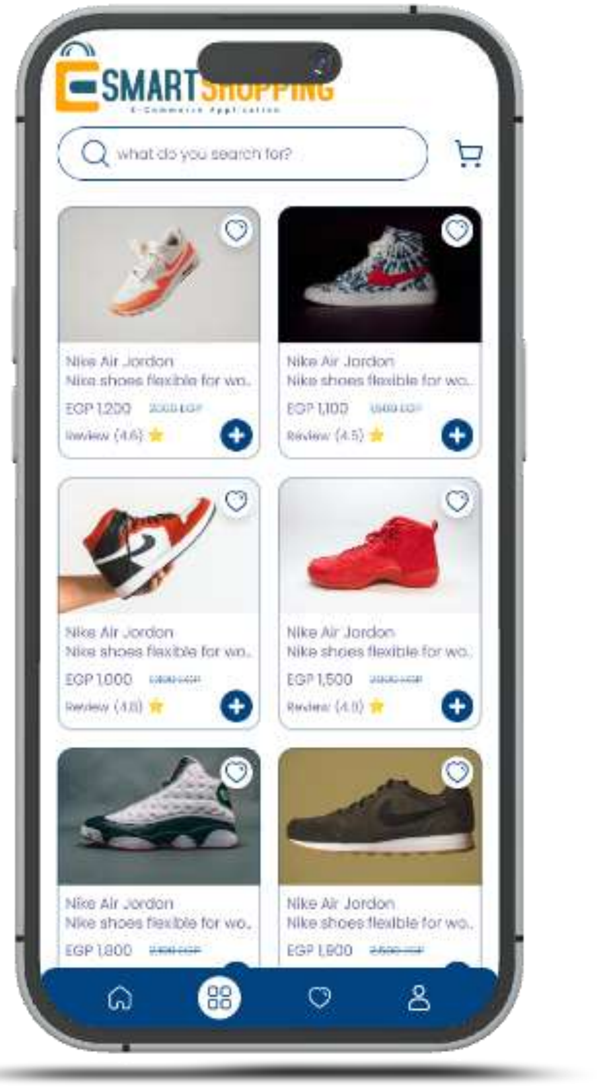
Category



The e-commerce app's category section offers efficient navigation, organizing products into distinct sections like clothing, electronics, and home goods. Users can easily explore desired items, enhancing their shopping experience and saving time.

User Experience

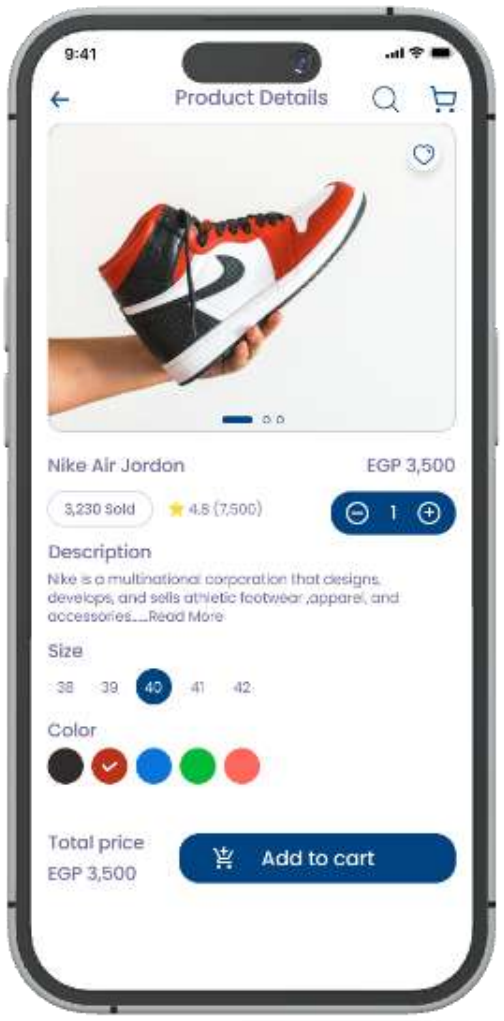
Product list



The product list in the e-commerce app presents a comprehensive selection of items within each category, catering to diverse preferences and needs. Users can browse through visually appealing displays, accessing detailed information and customer reviews for informed decision-making.

User Experience

Product details

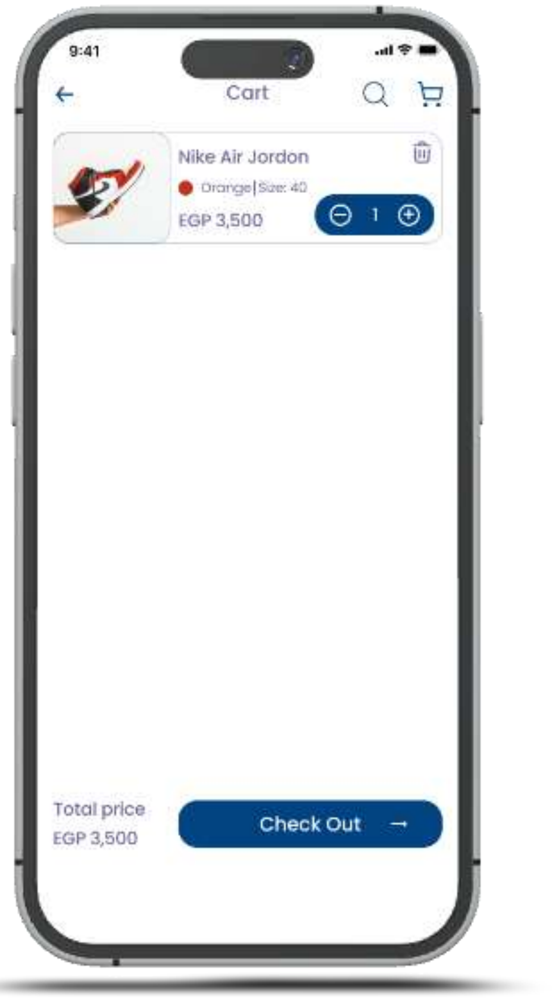


Nike is a multinational corporation that designs, develops, and sells athletic footwear ,apparel, and accessories.....Read More

It offer comprehensive information, including descriptions, specifications, and pricing, aiding users in making informed purchasing decisions. High-quality images provide a clear representation of the product, complemented by user reviews for added authenticity and trust.

User Experience

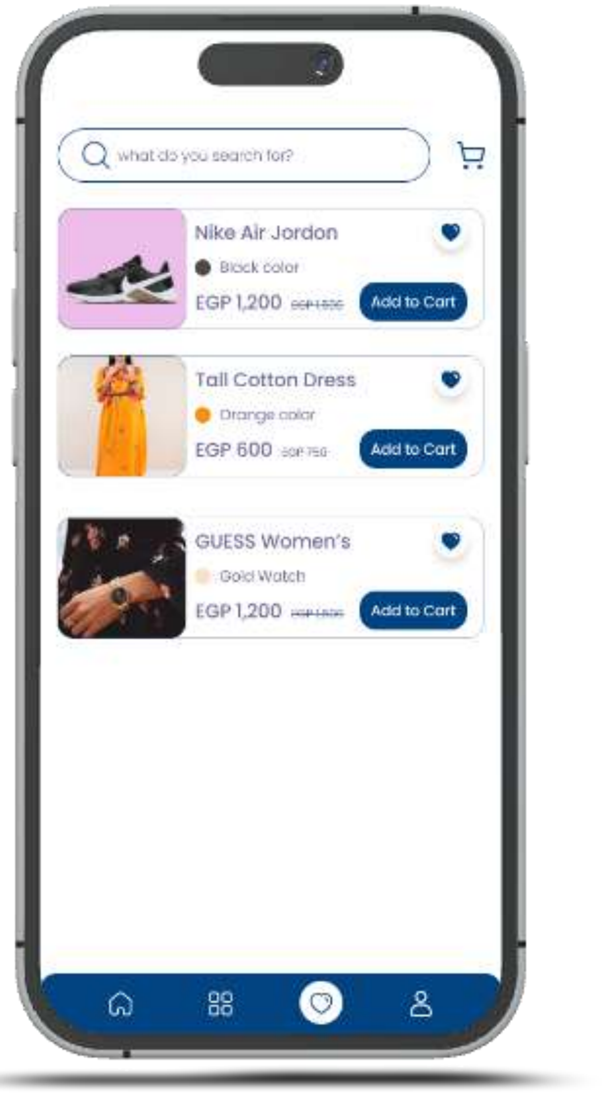
Add to cart



Allows users to effortlessly select desired items for purchase, streamlining the shopping process. With just a simple tap, chosen products are conveniently stored for checkout, enhancing user convenience and satisfaction.

User Experience

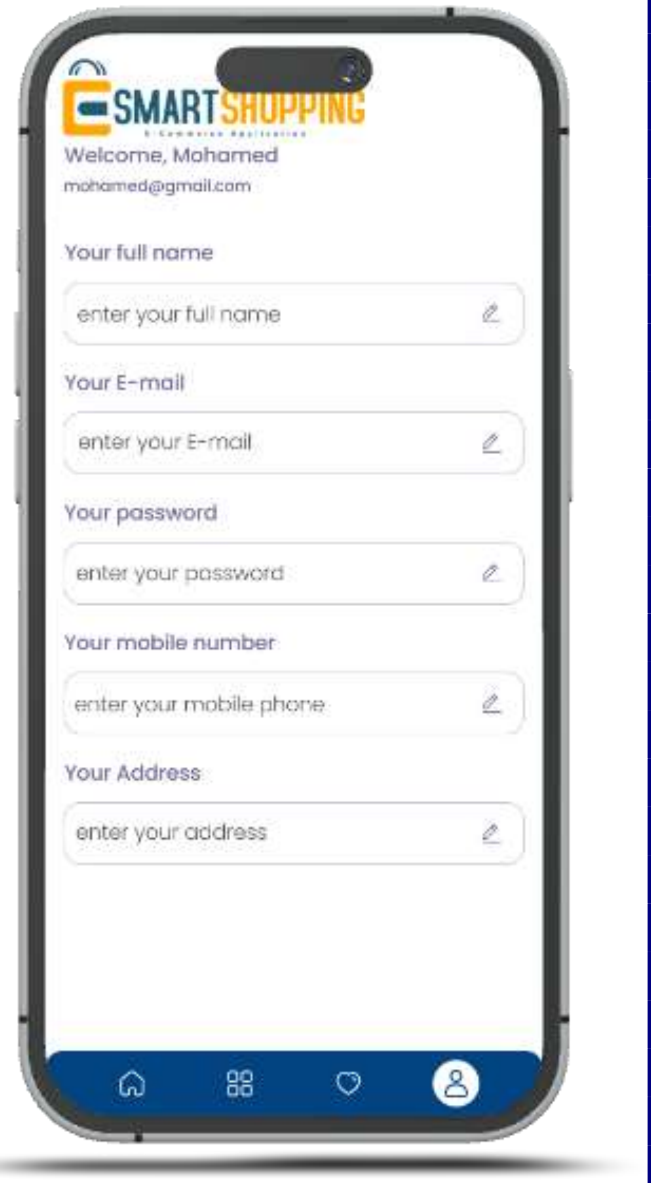
Wishlist



It collection of desired items for future purchase consideration. With a simple tap, users can add products they're interested in, fostering engagement and facilitating repeat visits. Wishlist functionality enhances user experience by allowing for easy tracking and organization of preferred items.

User Experience

User account



Welcome, Mohamed
mohamed@gmail.com

Your full name
enter your full name

Your E-mail
enter your E-mail

Your password
enter your password

Your mobile number
enter your mobile phone

Your Address
enter your address

It provides personalized access to order history, preferences, and saved information, enhancing user engagement and convenience. Users can easily manage their profile, track orders, and access customer support, fostering a seamless and satisfying shopping experience.

Chapter 4

Machine learning

- BACKGROUND
- OVERVIEW
- ALGORITHMS
- DATASET
- LIBRARIES USED
- COMPARE MODEL

Background

➢ Introduction

The goal of ML is often to understand the structure of data and integrate that data into models that can be understood and used by humans, it is known that ML is an application of artificial intelligence (AI). Although ML is an area of computer science, it differs from traditional computer science approaches. In a traditional computer, an algorithm is a set of explicitly programmed instructions used by the computer to calculate or solve a problem. Instead, ML algorithms allow computers to train on data inputs and use statistical analysis to generate values within a specific range. For this reason, machine learning enables computers to create models from sample data to automate decision-making processes based on input data. Machine learning assists today's technology users. Facial recognition technology enables social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts an image of text into a movable type. The recommendation engine, powered by machine learning, recommends which movies or TV shows to watch next based on user preferences. Self-driving cars that rely on ML for navigation could soon be available to consumers.
Machine learning is an ever-evolving field. For this reason, there are a number of considerations you need to keep in mind when working with machine learning methods or analyzing the impact of ML processes.

➢Machine Learning Methods

In ML, tasks are often categorized into broad categories. These categories are based on how learning is received or how feedback on learning is provided to the developed system. Two of the most widely applied ML methods are supervised learning that trains algorithms on labeled human input and output data and unsupervised learning that provides algorithms. math has no labeled data to allow it to find structure in its input data. Let's explore these methods in more detail.

➢ Supervised Learning

In supervised learning, the computer is given sample inputs labeled with their desired output. The purpose of this approach is that the algorithm can "learn" by comparing its actual output with the "taught" output to seek out errors and modify the model accordingly. Thus supervised learning uses models to predict label values on unlabeled additional data. For example, with supervised learning, an algorithm could be fed data with images of sharks labeled as fish and images of oceans labeled as water. By being trained on this data, the supervised learning algorithm will be able to later identify shark images that are not labeled as fish and ocean images that aren’t labeled as water.

➢ Unsupervised Learning

Unsupervised learning is used to identify patterns in a data set that contain unclassified as well as unlabeled data points. Algorithms are thus allowed to classify, label and/or group the data points contained in the dataset without any external instructions in performing this task. In other words, unsupervised learning allows the system to identify patterns in the data set on 6 its own. In unsupervised learning, an AI system will group unsorted information based on similarities and differences, even if no categories are provided. Unsupervised learning algorithms can perform more complex processing tasks than supervised learning systems. Also, applying a system for unsupervised learning is one way to test AI.

➢ Semi-supervised Learning

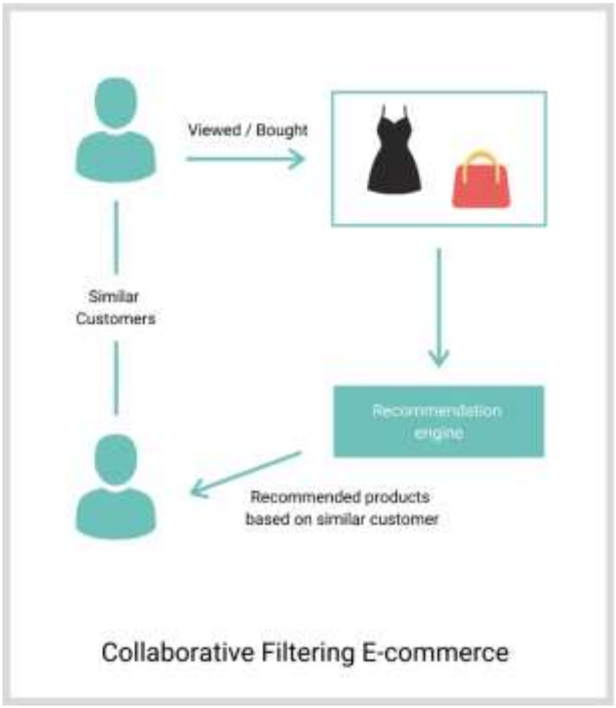
Semi-supervised ML algorithms fall somewhere between supervised and unsupervised learning, as they use both labeled and unlabeled data for training - often a small amount of data to be trained. Labeled and large amounts of unlabeled data. Systems using this approach can significantly improve the accuracy of learning. Usually, semi-supervised learning is chosen when the labeled data obtained requires relevant and authoritative resources to train/learn from. Otherwise, collecting unlabeled data usually requires no additional resources.

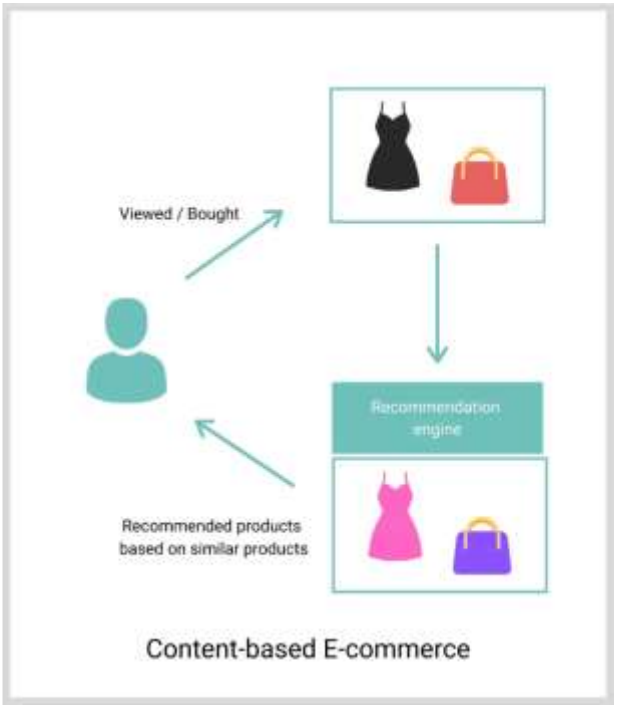
Introduction for recommendation:

We all have used services like Netflix, Amazon, and YouTube. These services use very sophisticated systems to recommend the best items to their users to make their experiences great. But, how do they achieve such great systems

We have two type of recommendation:

Collaborative Based Filtering	Content Based Filtering
Suggests items to a user based on information from multiple users who had similar preferences from past interactions. It’s based on the idea that people who agreed on the evaluation of certain items in the past are likely to agree again in the future.	Content Based Filtering recommends items based on the previous items the same consumer selected in the past. This method is best used when the focus is on one user and the attributes of the items are the most crucial factor in making a recommendation.


Viewed / Bought
Similar Customers
Recommendation engine
Recommended products based on similar customer
Collaborative Filtering E-commerce



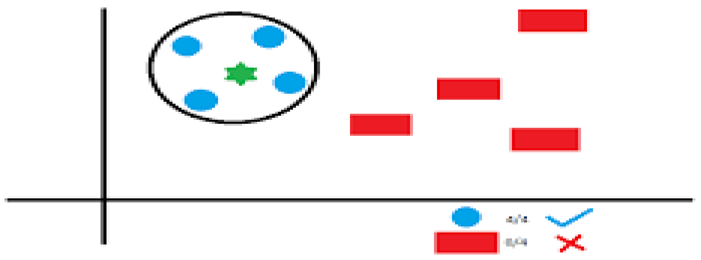
Overview

In the retail industry, personalized product recommendations play a crucial role in enhancing customer experience and boosting sales. This project focuses on building a content-based recommendation system using Support Vector Machine (SVM) classifier. The system recommends products based on their similarity to a given input product. It utilizes the TF-IDF vectorization technique to represent product names as numerical features and trains an SVM model to classify products into different categories. The trained model is then used to recommend similar products from the same category.

Algorithms

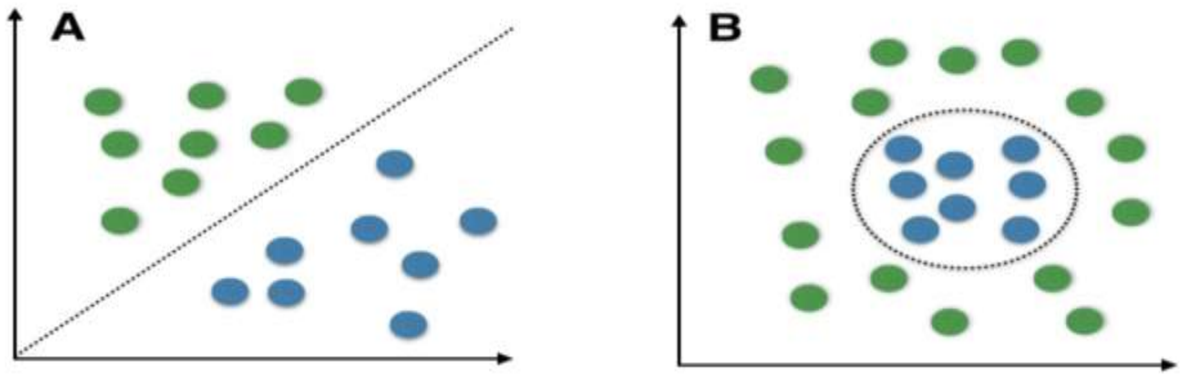
KNN (k-Nearest Neighbor):

The KNN algorithm is one of the simplest supervised classification algorithms. Even with such simplicity, it can give highly competitive results. KNN algorithm can also be used for regression problems. Using the kNN algorithm we achieved an accuracy of 93.92%



SVM (Support vector machine):

Support Vector Machine (SVM) is a relatively simple Supervised Machine Learning Algorithm used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2- dimensional space, this hyper-plane is nothing but a line
Using the svm algorithm we achieved an accuracy of 95.76%.



Algorithm	Accuracy
SVM	95.76%
kNN	93.92%

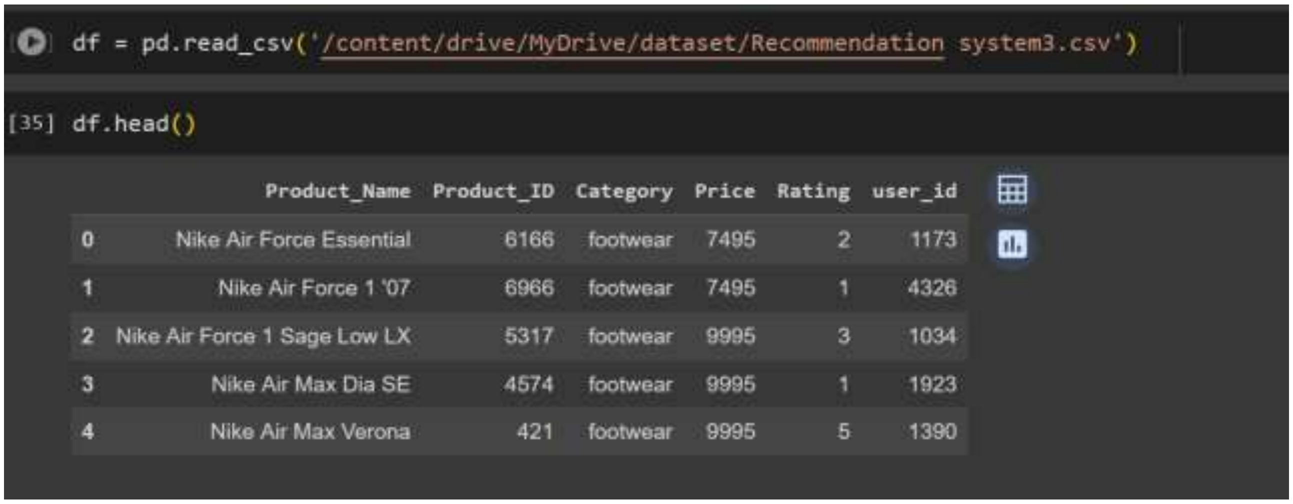
Dataset

➢ Description

The dataset comprises various product entries with information such as product names, product IDs, categories, prices, ratings, and user IDs.

➢ Data Field

- Product Name: The name of the product.
- Product ID: Unique identifier for the product.
- Category: The category of the product (e.g., footwear).
- Price: The price of the product.
- Rating: The rating given to the product.
- User_id: User identifier



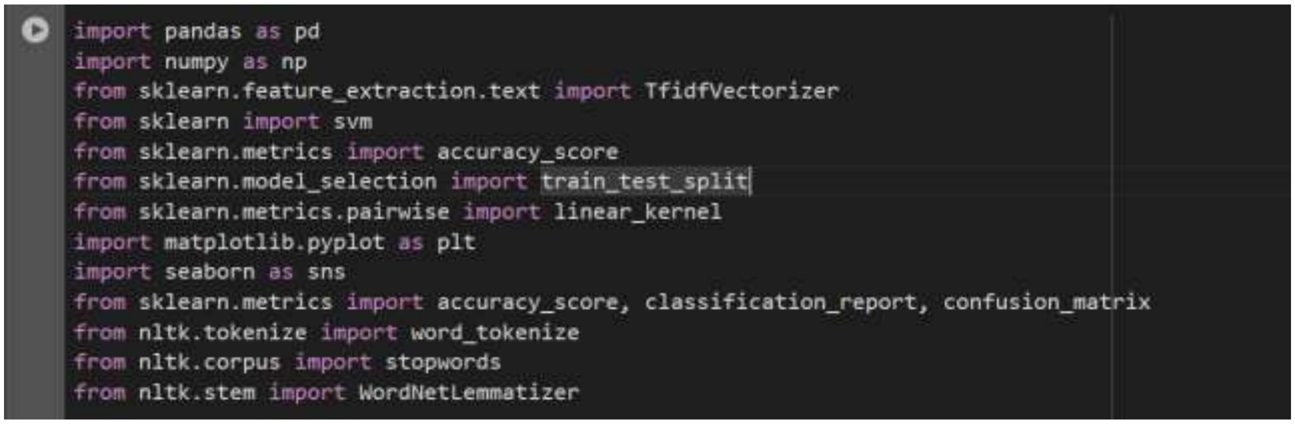
```
df = pd.read_csv('/content/drive/MyDrive/dataset/Recommendation system3.csv')
```

```
[35] df.head()
```

```
                  Product_Name  Product_ID  Category  Price  Rating  user_id
0      Nike Air Force Essential        6166  footwear   7495       2     1173
1           Nike Air Force 1 '07        6966  footwear   7495       1     4326
2  Nike Air Force 1 Sage Low LX        5317  footwear   9995       3     1034
3          Nike Air Max Dia SE        4574  footwear   9995       1     1923
4          Nike Air Max Verona         421  footwear   9995       5     1390
```

Libraries Used

- **Pandas**: Used for data manipulation and analysis.
- **Sklearn**: Utilized for machine learning algorithms, data preprocessing, and evaluation metrics.
- **Nltk**: Employed for text preprocessing tasks such as tokenization, stop word removal, and lemmatization.
- **Seaborne and matplotlib**: Used for data visualization.



```python
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import linear_kernel
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

Steps:

1) Data Propagation

- The sample dataset containing product information such as Product ID, Product Name, Category, and Description is created.
- Handling missing values
- Check duplicate

2) Text Preprocessing

- Product names are converted to lowercase, tokenized into individual words, and lemmatized to reduce them to their base forms.
- Stop words are removed to eliminate common words that do not contribute much to the meaning. Check duplicate

3) Data Splitting:

- The dataset is split into training and testing sets using the train_test_split function from sklearn.model_selection.

4) TF-IDF Vectorization:

- Product names in both the training and testing sets are converted into TF-IDF (Term Frequency-Inverse Document Frequency) vectors using TfidfVectorizer from sklearn.feature_extraction.text.
- The TF-IDF victimizer is employed to convert product names into numerical representations, capturing their importance within the dataset.

5) Cosine Similarity Calculation

- Cosine similarity is used to measure the similarity between TF-IDF vectors, providing a quantitative measure of similarity between product names

Model Training:

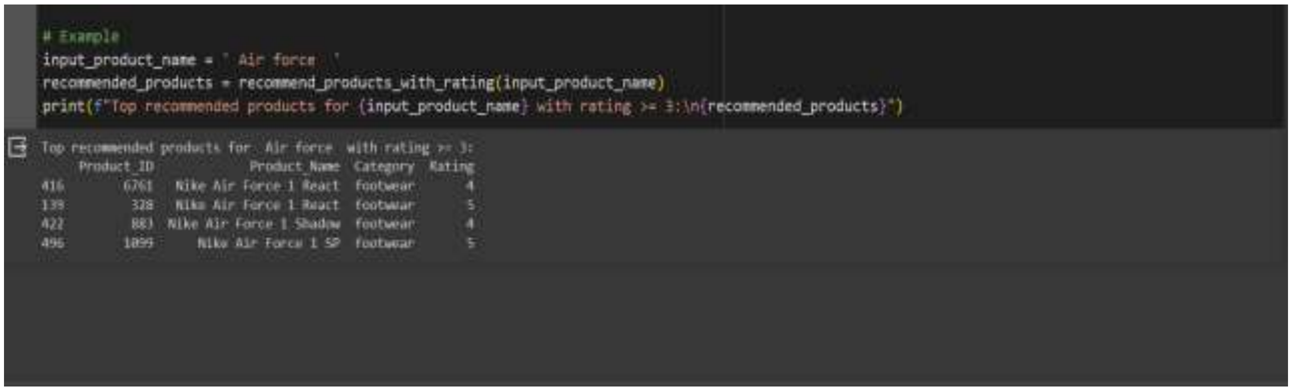
- An SVM classifier with a linear kernel is trained on the TF-IDF vectors of the training set using svm.SVC.

6) Prediction and Evaluation:

- The trained model is used to predict the categories of products in the testing set.
- Accuracy of the predictions is computed using accuracy score from sklearn. Metrics.
- Classification report and confusion matrix are generated to evaluate the performance of the model.

7) Recommendation:

- A function recommend products is defined to recommend similar products based on an input product name.
- The function calculates the TF-IDF vector for the input product name, predicts its category using the trained SVM model, and recommends similar products from the same category based on cosine similarity.



```
# Example
input_product_name = ' Air force '
recommended_products = recommend_products_with_rating(input_product_name)
print(f"Top recommended products for {input_product_name} with rating >= 3:\n{recommended_products}")
```

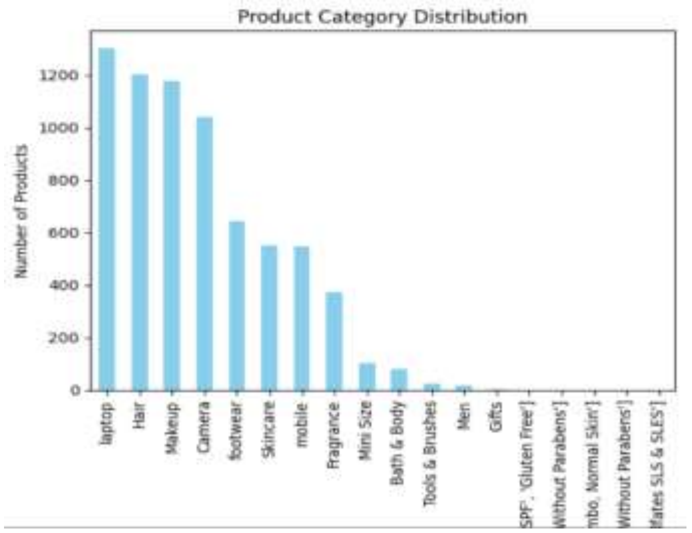
```
Top recommended products for  Air force  with rating >= 3:
     Product_ID              Product_Name  Category  Rating
416        6761     Nike Air Force 1 React  footwear       4
139         328     Nike Air Force 1 React  footwear       5
422         883    Nike Air Force 1 Shadow  footwear       4
496        1899         Nike Air Force 1 SP  footwear       5
```

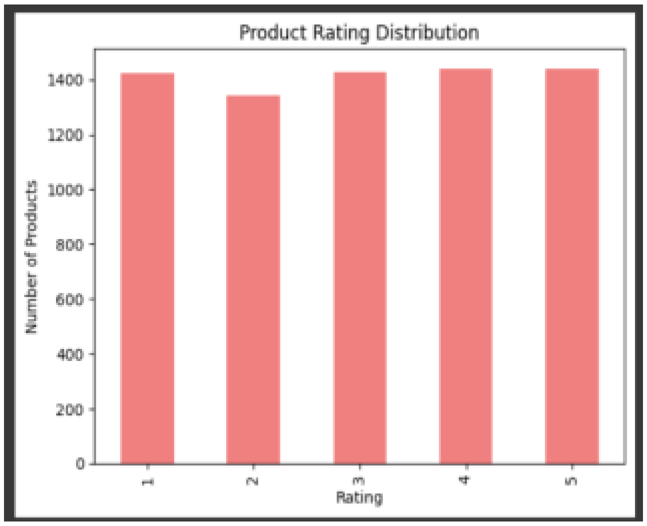
Compare models:

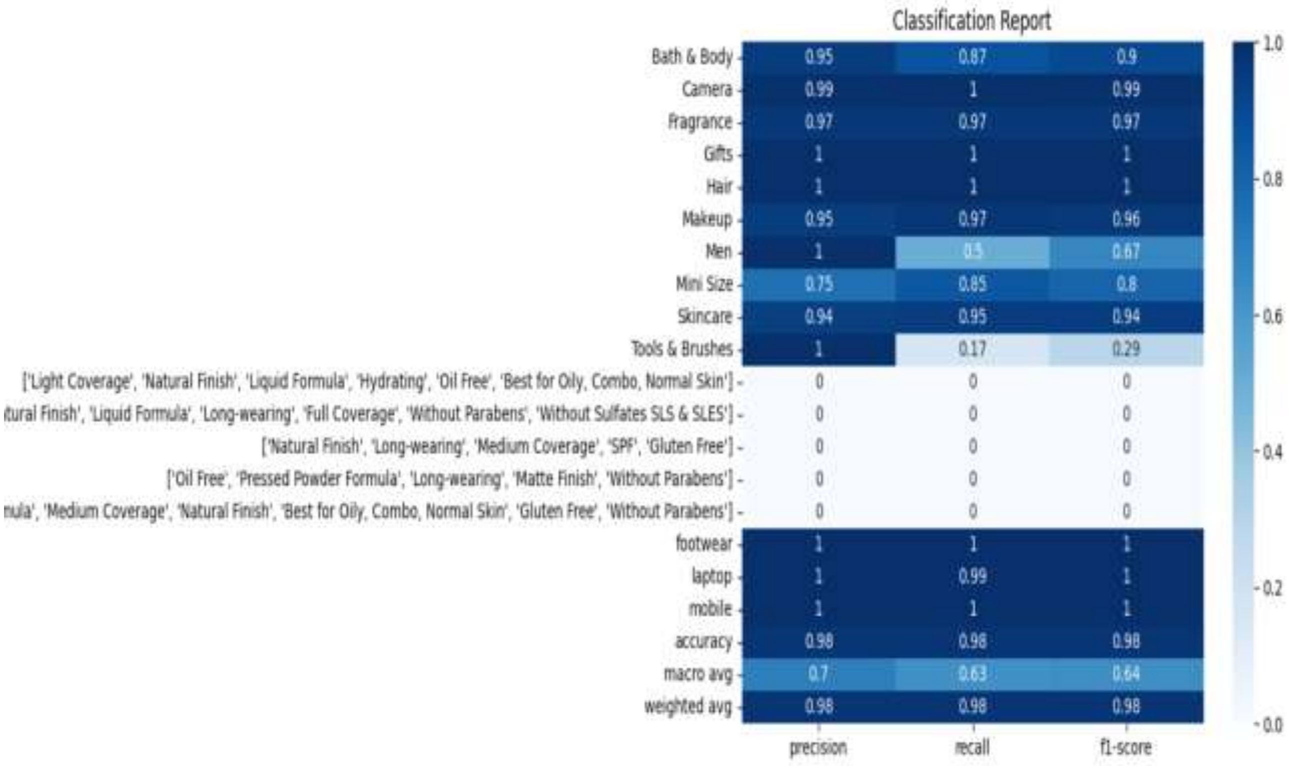
model	**svm**	**kNN**	**Random forest**	**Logistic Regression**	**KMeans**
accuracy	**95.76%**	**93%**	**99%**	**99%**	**63%**

The accuracy of the svm model was 95.76% for this Reason we choose Svm the Predictive Model.

List of Appendix:







	precision	recall	f1-score
Bath & Body	0.95	0.87	0.9
Camera	0.99	1	0.99
Fragrance	0.97	0.97	0.97
Gifts	1	1	1
Hair	1	1	1
Makeup	0.95	0.97	0.96
Men	1	0.5	0.67
Mini Size	0.75	0.85	0.8
Skincare	0.94	0.95	0.94
Tools & Brushes	1	0.17	0.29
['Light Coverage', 'Natural Finish', 'Liquid Formula', 'Hydrating', 'Oil Free', 'Best for Oily, Combo, Normal Skin']	0	0	0
tural Finish', 'Liquid Formula', 'Long-wearing', 'Full Coverage', 'Without Parabens', 'Without Sulfates SLS & SLES']	0	0	0
['Natural Finish', 'Long-wearing', 'Medium Coverage', 'SPF', 'Gluten Free']	0	0	0
['Oil Free', 'Pressed Powder Formula', 'Long-wearing', 'Matte Finish', 'Without Parabens']	0	0	0
nula', 'Medium Coverage', 'Natural Finish', 'Best for Oily, Combo, Normal Skin', 'Gluten Free', 'Without Parabens']	0	0	0
footwear	1	1	1
laptop	1	0.99	1
mobile	1	1	1
accuracy	0.98	0.98	0.98
macro avg	0.7	0.63	0.64
weighted avg	0.98	0.98	0.98

Chabot

➢ Introduction

The Question-Answering System for E-commerce FAQ is an innovative solution designed to revolutionize customer support interactions on e-commerce platforms. With the rapid growth of online shopping, providing timely and accurate assistance to users has become paramount for maintaining customer satisfaction and loyalty. This project addresses this challenge by leveraging state-of-the-art machine learning techniques to automate responses to frequently asked questions (FAQs), thereby enhancing user experience and optimizing resource allocation within customer service teams.

➢ Features

Text Classification At the core of the system lies a sophisticated text classification model trained to categorize user queries into predefined topics or themes. By analyzing the textual content of each query, the model assigns it to the most appropriate category, allowing for accurate retrieval of relevant answers from the FAQ database.

➢TF-IDF vectorization

Text data preprocessing is performed using the Term Frequency-Inverse Document Frequency (TF-IDF) technique, a widely-used method for transforming raw text into numerical feature vectors. By capturing the importance of words in documents relative to a larger corpus, TF-IDF enables the model to extract meaningful

➢ Process

Linear Support Vector Classifier The classification model is implemented using a Linear Support Vector Classifier (LinearSVC), a robust and efficient algorithm known for its effectiveness in text classification tasks. By identifying the optimal hyperplane that separates different classes in the feature space, LinearSVC achieves high

➢ Pipe line Architecture

To streamline the text processing and classification workflow, the system utilizes a scikit-learn pipeline, a flexible tool for chaining multiple processing steps into a single object. This modular architecture enables seamless integration of various preprocessing techniques and classification algorithms, facilitating experimentation and optimization of the model's performance.

➢ Accuracy Evaluation

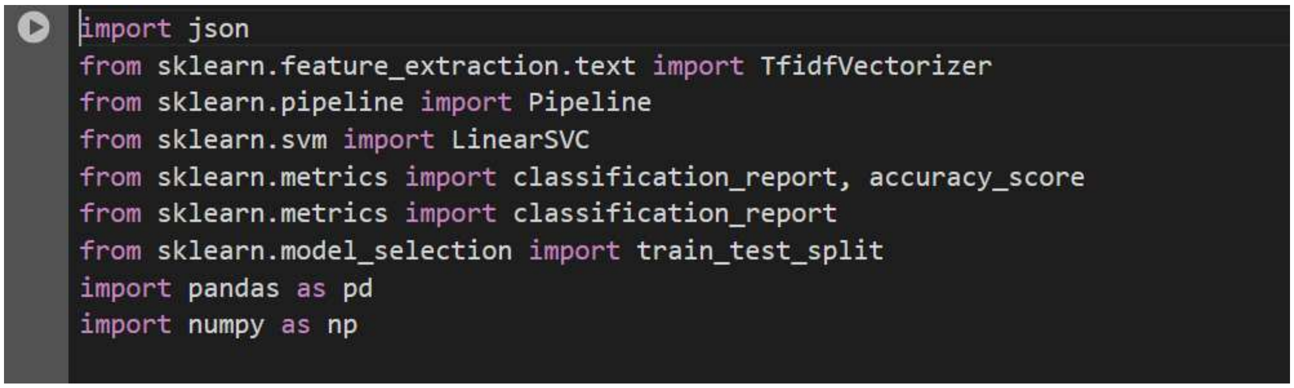
The performance of the trained model is evaluated using a comprehensive set of classification metrics, including precision, recall, F1-score, and overall accuracy. By quantifying the model's ability to correctly classify user queries and retrieve relevant answers, these metrics provide valuable insights into its effectiveness and reliability in real-world scenarios.

➢ Workflow

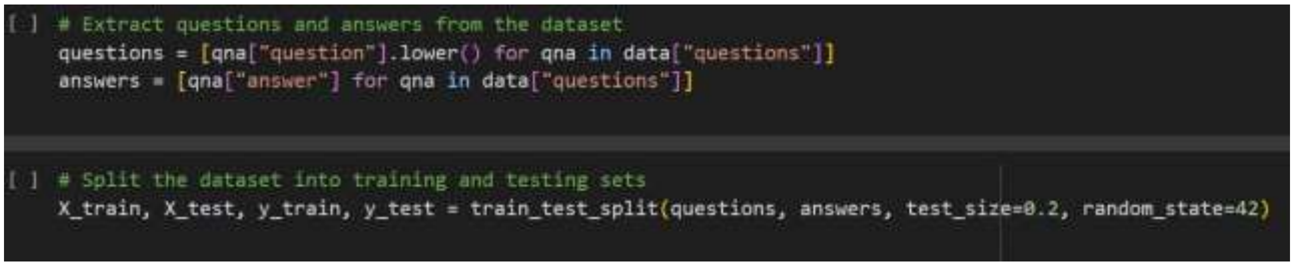
Data Loading: The system loads a rich dataset containing a diverse range of questions and their corresponding answers from a JSON file. This dataset serves as the foundation for training and evaluating the text classification model.

Preprocessing: Prior to training the model, the text data undergoes preprocessing to ensure consistency and uniformity. This includes converting all questions to lowercase, removing punctuation, and applying other text normalization techniques to enhance the quality of the input data.

Train-Test Split: The dataset is divided into training and testing sets using the train_test_split function from scikit-learn. This partitioning ensures that the model is trained on a sufficiently large and representative sample of data while retaining a separate portion for unbiased evaluation of its performance.

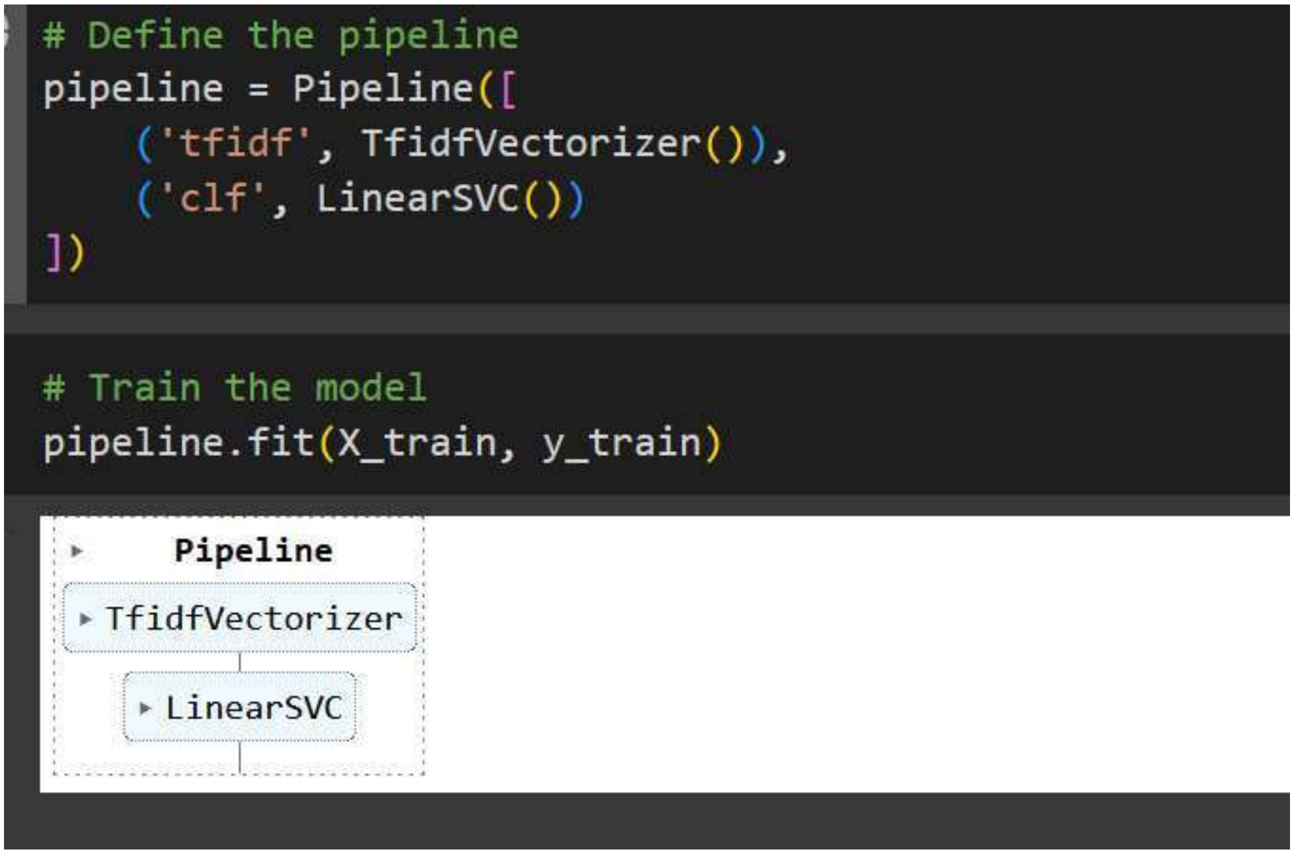


```python
import json
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```



```python
# Extract questions and answers from the dataset
questions = [qna["question"].lower() for qna in data["questions"]]
answers = [qna["answer"] for qna in data["questions"]]
```

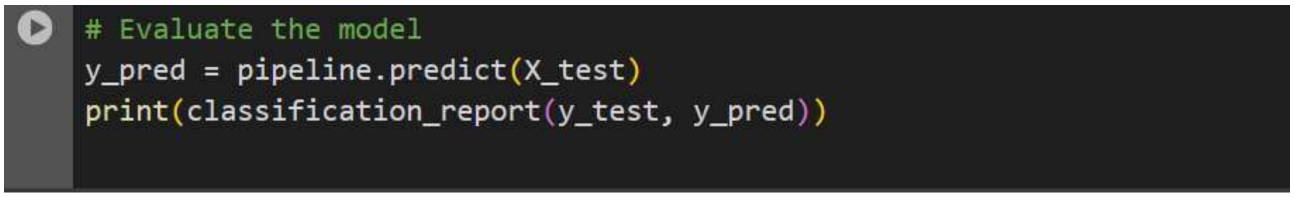
```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(questions, answers, test_size=0.2, random_state=42)
```



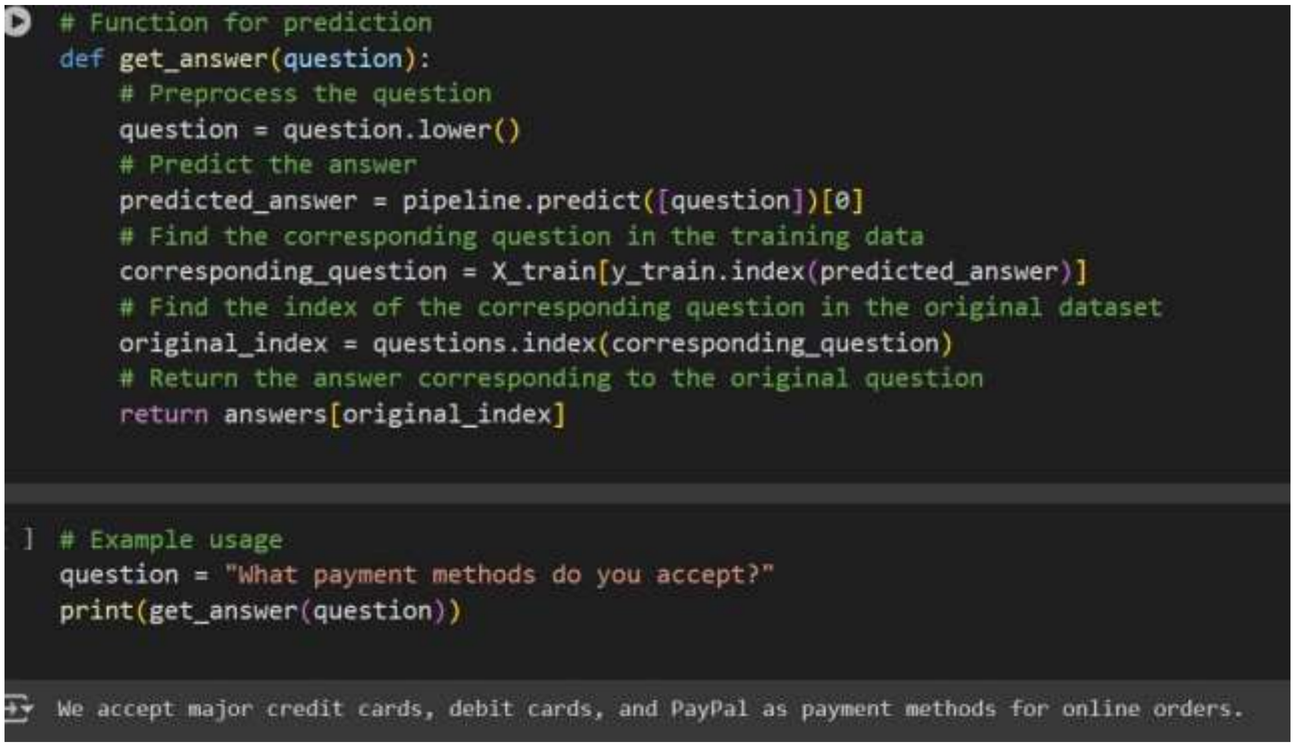
```python
# Define the pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LinearSVC())
])
```

```python
# Train the model
pipeline.fit(X_train, y_train)
```

```
▸ Pipeline
▸ TfidfVectorizer
▸ LinearSVC
```



```
# Evaluate the model
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))
```



```
# Function for prediction
def get_answer(question):
    # Preprocess the question
    question = question.lower()
    # Predict the answer
    predicted_answer = pipeline.predict([question])[0]
    # Find the corresponding question in the training data
    corresponding_question = X_train[y_train.index(predicted_answer)]
    # Find the index of the corresponding question in the original dataset
    original_index = questions.index(corresponding_question)
    # Return the answer corresponding to the original question
    return answers[original_index]
```

```
# Example usage
question = "What payment methods do you accept?"
print(get_answer(question))
```

```
We accept major credit cards, debit cards, and PayPal as payment methods for online orders.
```

Chapter 5

Implementation

- BACKEND
- FLUTTER

Project overview

Our project is the engine that powers an online store. It runs behind the scenes, handling all thecritical tasks that users don't directly see but are essential for a smooth shopping experience.

Our Goals are:

- **Efficient Product Management:** The backend manages the product, categories, Brands and subcategories catalogs, including adding, editing, deleting and storingtheir information, and keeping track of inventory levels.
- **Robust User Accounts:** The backend creates, manages, and secures user accounts. This includes storing user data securely, handling logins, and managinguser profiles.

Order Processing: It facilitates the entire order process - handling customer orders, tax calculations, and order fulfillment.

Functionalities:

1. Product Management:

- CRUD operations (Create, Read, Update, Delete) for product data.
- Inventory management and tracking.
- Image storage and retrieval.
- Category which the product belongs to, as subcategory and brand too.

2. Order Processing:

- Shopping cart management.
- Tax calculations.
- Order tracking and fulfillment.

3. Customer Management:

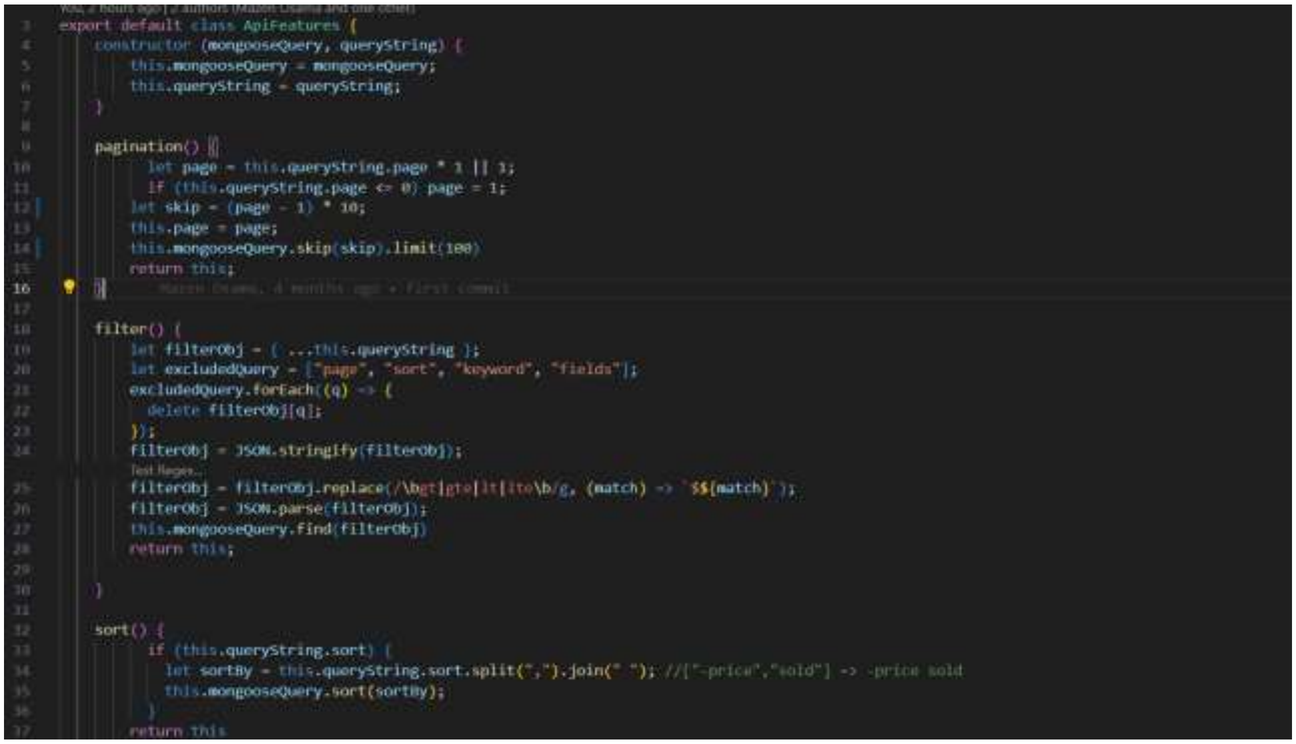
- User registration, login, and authentication.
- User profile management, order history and Wishlist.
- Secure user data storage.

4. Security:

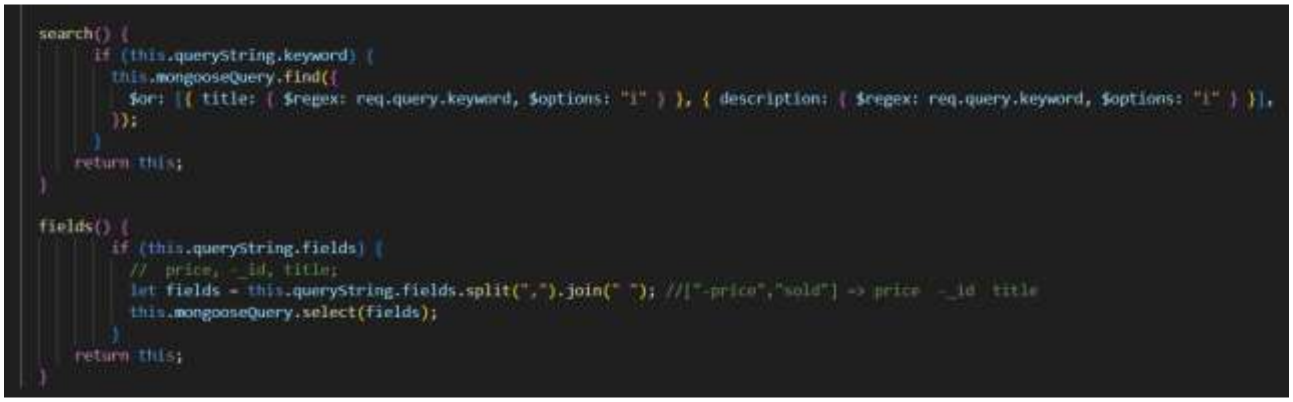
- Secure user authentication and authorization.
- Data encryption (like hashing user's password).

In the end we can say that the back-end is the backbone of an e-commerce store.

API Gateway: this is the entry point for all API requests. It routes requests to the appropriate back-end service.

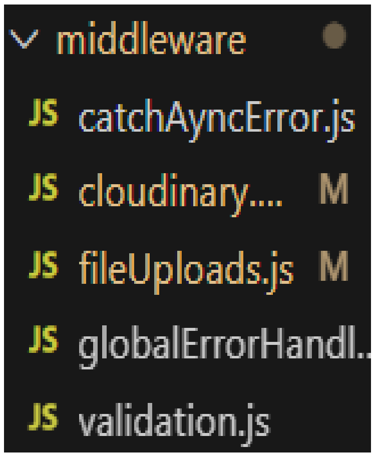


```
export default class ApiFeatures {
    constructor (mongooseQuery, queryString) {
        this.mongooseQuery = mongooseQuery;
        this.queryString = queryString;
    }

    pagination() {
          let page = this.queryString.page * 1 || 1;
          if (this.queryString.page <= 0) page = 1;
        let skip = (page - 1) * 10;
        this.page = page;
        this.mongooseQuery.skip(skip).limit(100)
        return this;
    }

    filter() {
        let filterObj = { ...this.queryString };
        let excludedQuery = ["page", "sort", "keyword", "fields"];
        excludedQuery.forEach((q) => {
          delete filterObj[q];
        });
        filterObj = JSON.stringify(filterObj);
        filterObj = filterObj.replace(/\bgt|gte|lt|lte\b/g, (match) => `$${match}`);
        filterObj = JSON.parse(filterObj);
        this.mongooseQuery.find(filterObj)
        return this;

    }

    sort() {
          if (this.queryString.sort) {
            let sortBy = this.queryString.sort.split(",").join(" "); //["-price","sold"] -> -price sold
            this.mongooseQuery.sort(sortBy);
          }
        return this
```

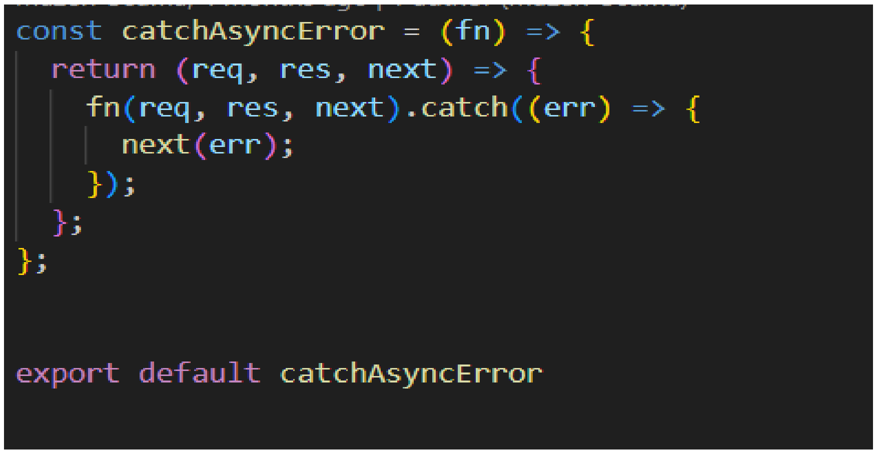


```
search() {
      if (this.queryString.keyword) {
        this.mongooseQuery.find({
          $or: [{ title: { $regex: req.query.keyword, $options: "i" } }, { description: { $regex: req.query.keyword, $options: "i" } }],
        });
      }
    return this;
}

fields() {
        if (this.queryString.fields) {
          //  price, -_id, title;
          let fields = this.queryString.fields.split(",").join(" "); //["-price","sold"] -> price  -_id  title
          this.mongooseQuery.select(fields);
        }
    return this;
}
```

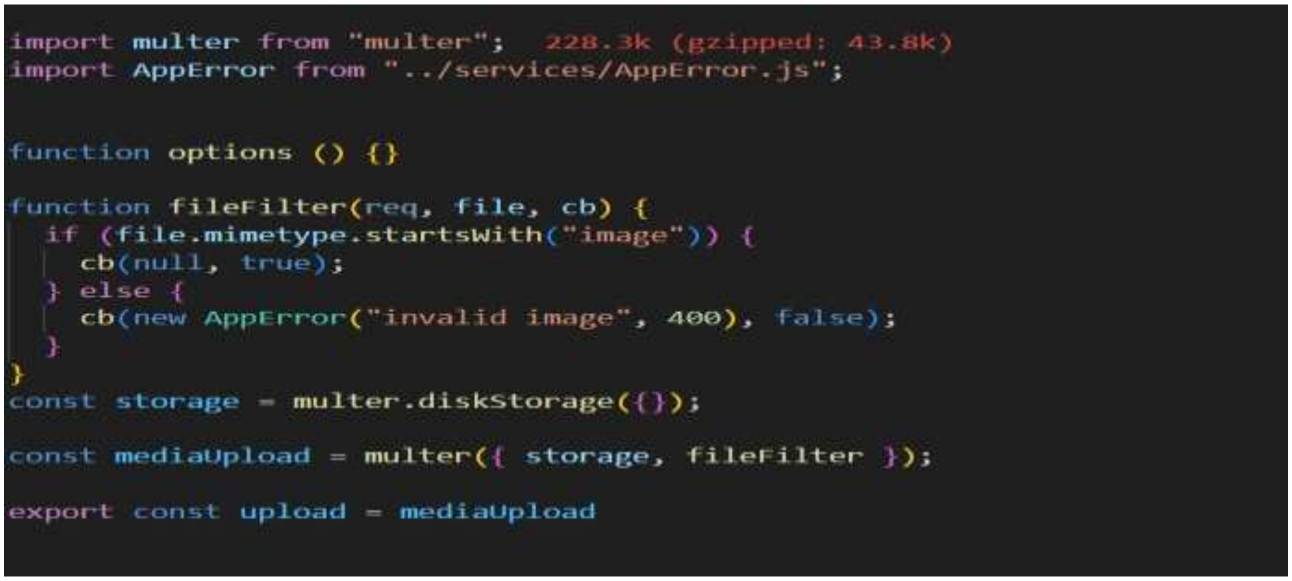
Business Logic Layer: the middleware that contains all the functions, Error handlers and Validation, it interacts with the database and other services to execute business logic.



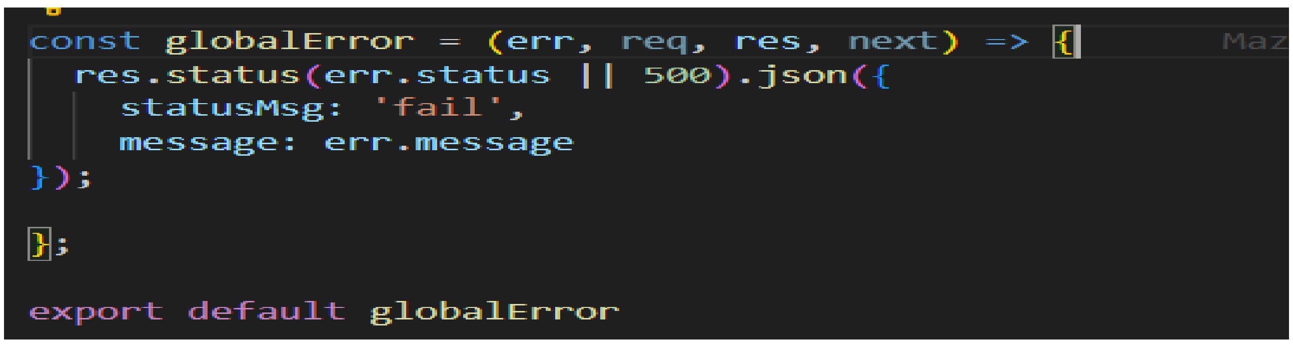
```
middleware
  JS catchAyncError.js
  JS cloudinary....  M
  JS fileUploads.js  M
  JS globalErrorHandl..
  JS validation.js
```



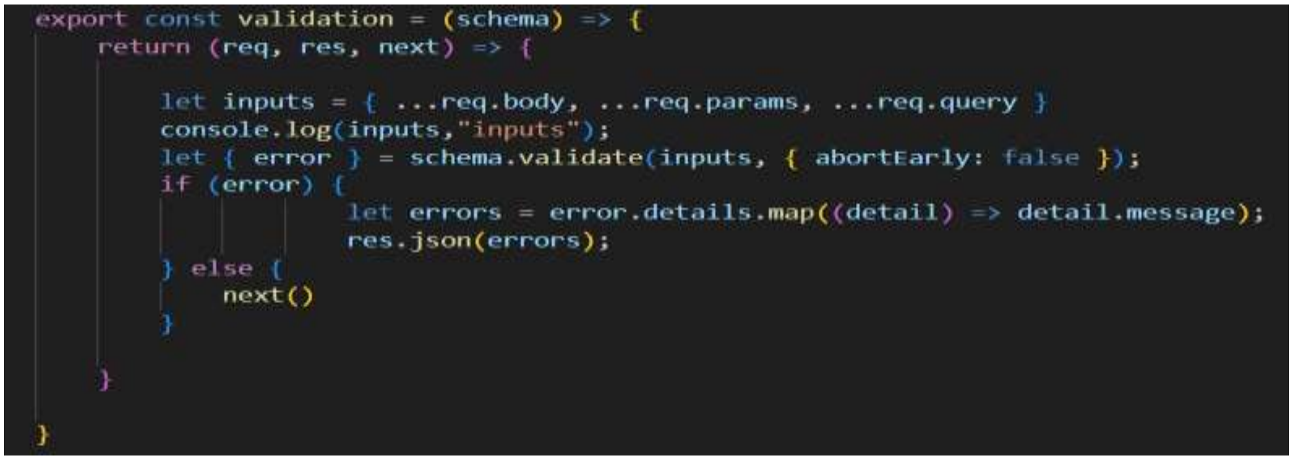
```
const catchAsyncError = (fn) => {
  return (req, res, next) => {
    fn(req, res, next).catch((err) => {
      next(err);
    });
  };
};


export default catchAsyncError
```



```
import multer from "multer";  228.3k (gzipped: 43.8k)
import AppError from "../services/AppError.js";


function options () {}

function fileFilter(req, file, cb) {
  if (file.mimetype.startsWith("image")) {
    cb(null, true);
  } else {
    cb(new AppError("invalid image", 400), false);
  }
}
const storage = multer.diskStorage({});

const mediaUpload = multer({ storage, fileFilter });

export const upload = mediaUpload
```

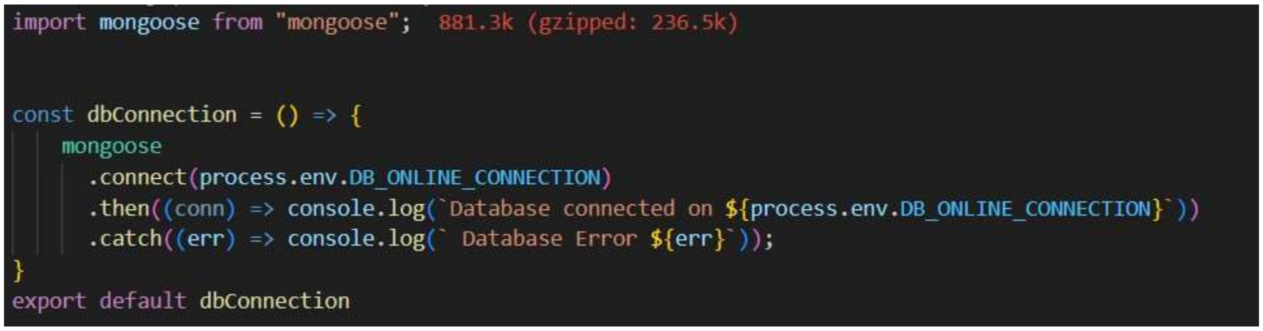


```
const globalError = (err, req, res, next) => {
    res.status(err.status || 500).json({
        statusMsg: 'fail',
        message: err.message
});

};

export default globalError
```

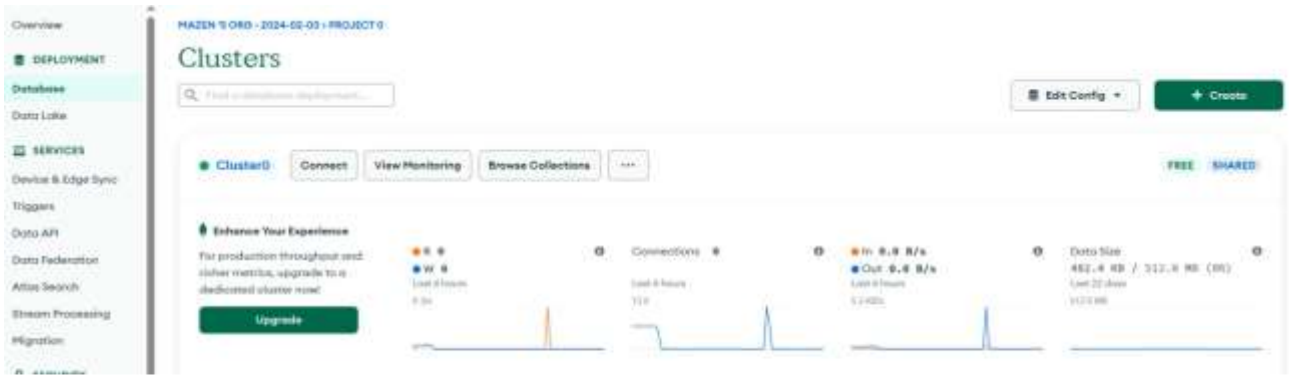


```
export const validation = (schema) => {
    return (req, res, next) => {

        let inputs = { ...req.body, ...req.params, ...req.query }
        console.log(inputs,"inputs");
        let { error } = schema.validate(inputs, { abortEarly: false });
        if (error) {
                    let errors = error.details.map((detail) => detail.message);
                    res.json(errors);
        } else {
            next()
        }

    }

}
```

Data Layer: This layer stores all the project data, typically in a relational database MongoDB (atlas). It manages data persistence, retrieval, and updates.



```
import mongoose from "mongoose";  881.3k (gzipped: 236.5k)


const dbConnection = () => {
    mongoose
      .connect(process.env.DB_ONLINE_CONNECTION)
      .then((conn) => console.log(`Database connected on ${process.env.DB_ONLINE_CONNECTION}`))
      .catch((err) => console.log(` Database Error ${err}`));
}
export default dbConnection
```



The Interactions:

- The user interacts with the front-end UI to browse products, add items to the cart,and initiate checkout.
- The front-end sends API requests to the API Gateway.
- The API Gateway routes the request to the appropriate service in the Business LogicLayer.
- The Business Logic Layer retrieves data from the Data Layer and performs necessary operations based on the request. This may involve calculations,validations.
- The Business Logic Layer sends a response back to the API Gateway.
- The API Gateway forwards the response to the front-end UI.

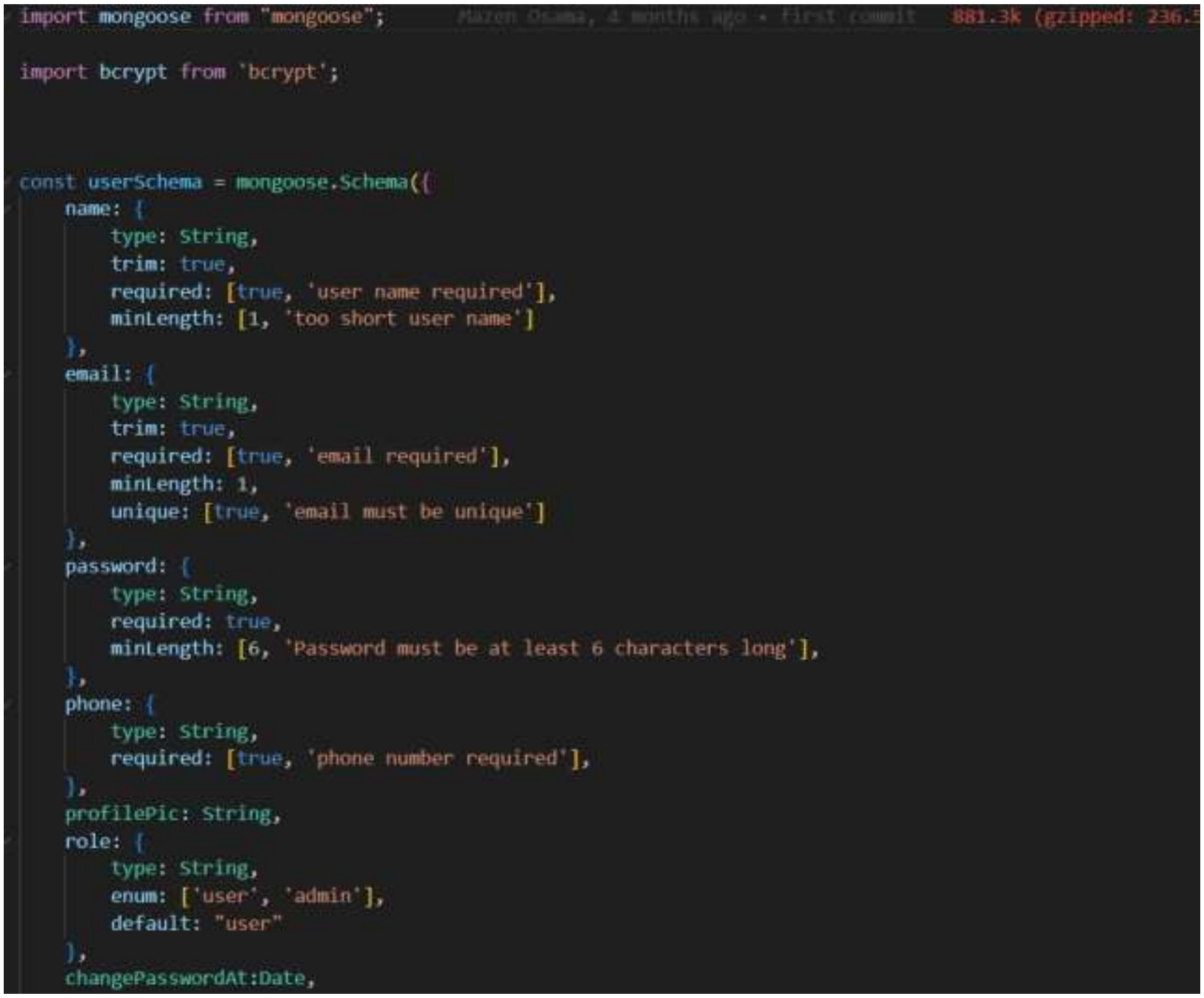
The front-end updates the UI based on the received information, providing feedbackto the user.

API References

User API:

- Schema Structure:

  - The user Schema represents the structure of a user document in a MongoDBdatabase.
  - It defines various fields that store information about a user.



```
import mongoose from "mongoose";      Mazen Osama, 4 months ago • first commit    881.3k (gzipped: 236.5

import bcrypt from 'bcrypt';


const userSchema = mongoose.Schema({
    name: {
        type: String,
        trim: true,
        required: [true, 'user name required'],
        minLength: [1, 'too short user name']
    },
    email: {
        type: String,
        trim: true,
        required: [true, 'email required'],
        minLength: 1,
        unique: [true, 'email must be unique']
    },
    password: {
        type: String,
        required: true,
        minLength: [6, 'Password must be at least 6 characters long'],
    },
    phone: {
        type: String,
        required: [true, 'phone number required'],
    },
    profilePic: String,
    role: {
        type: String,
        enum: ['user', 'admin'],
        default: "user"
    },
    changePasswordAt:Date,
```

- **Fields Explained:**

- Name: A string field representing the user's name. It is required, and its Minimum length is set to 1 character.
- Email: A string field representing the user's email address. It is required, Unique, and has a minimum length of 1 character.
- Password: A string field representing the user's password. It is required And must be at least 6 characters long.
- Phone: A string field representing the user's phone number. It is Required**.**
- Role: A string field representing the user's role (either 'user' or 'admin'). Itdefaults to 'user'.
- ChangePasswordAt: A date field that can be used to track when the userlast changed their password
- Is Active: A Boolean field indicating whether the user account is active.
- Wishlist: An array of references to product documents. This represents The user's Wishlist
- Address: An array of objects representing user addresses. Each addresshas city, street, and phone fields.
- verified: A Boolean field indicating whether the user's email address is Verified (defaults to false).

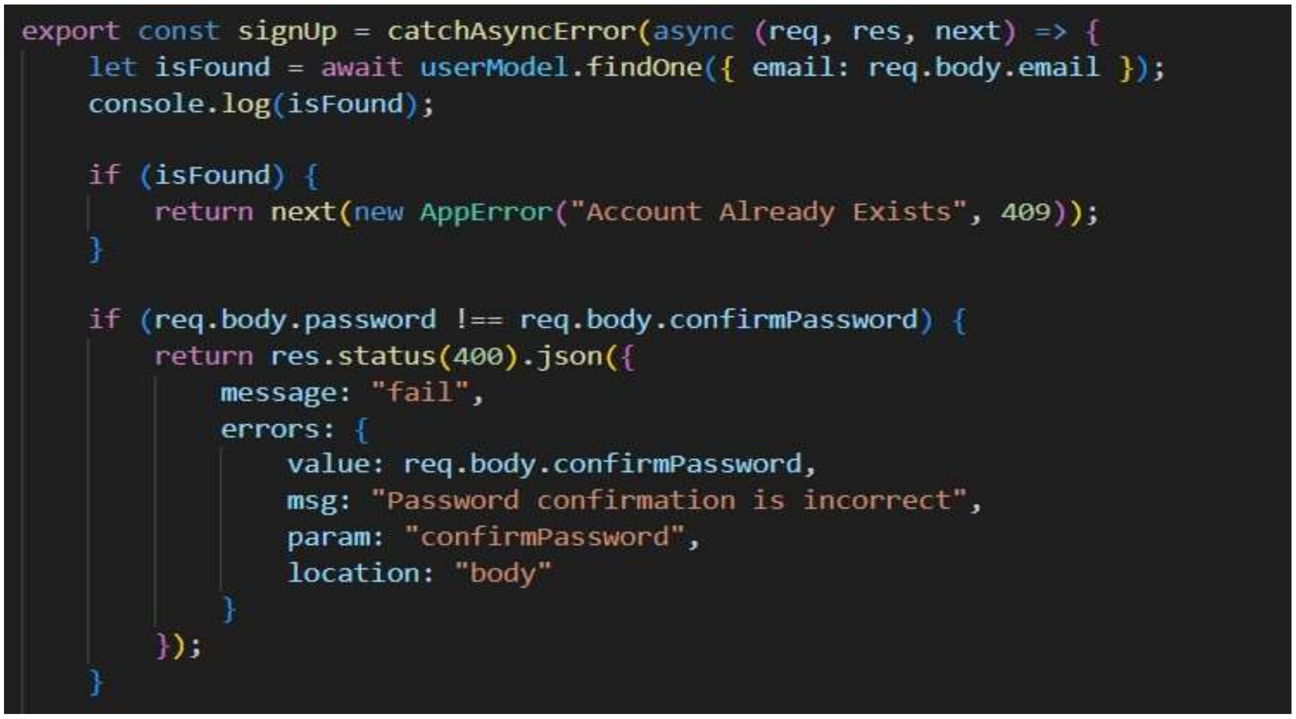
• Timestamp:

- The {timestamp: true} option automatically adds created at and updated at fields to the schema. These fields track when the user document was created or last updated.

• Password Hashing:

- The pre ('save') middleware ensures that the user's password is hashed using bcrypt before saving it to the database.
- If the password field is modified (e.g., during registration or password update), ithashes the new password.
- Similarly, the pre ('findOneAndUpdate') middleware hashes the updated password ifit exists in the update payload.

- **Virtual Field (commented out):**
  - The commented-out virtual field named Confirm-Password is not used in the schema. It appears to be related to password confirmation during registration.



```
export const signUp = catchAsyncError(async (req, res, next) => {
    let isFound = await userModel.findOne({ email: req.body.email });
    console.log(isFound);

    if (isFound) {
        return next(new AppError("Account Already Exists", 409));
    }

    if (req.body.password !== req.body.confirmPassword) {
        return res.status(400).json({
            message: "fail",
            errors: {
                value: req.body.confirmPassword,
                msg: "Password confirmation is incorrect",
                param: "confirmPassword",
                location: "body"
            }
        });
    }
```

Usage:

- You can use this schema to create and manage user accounts

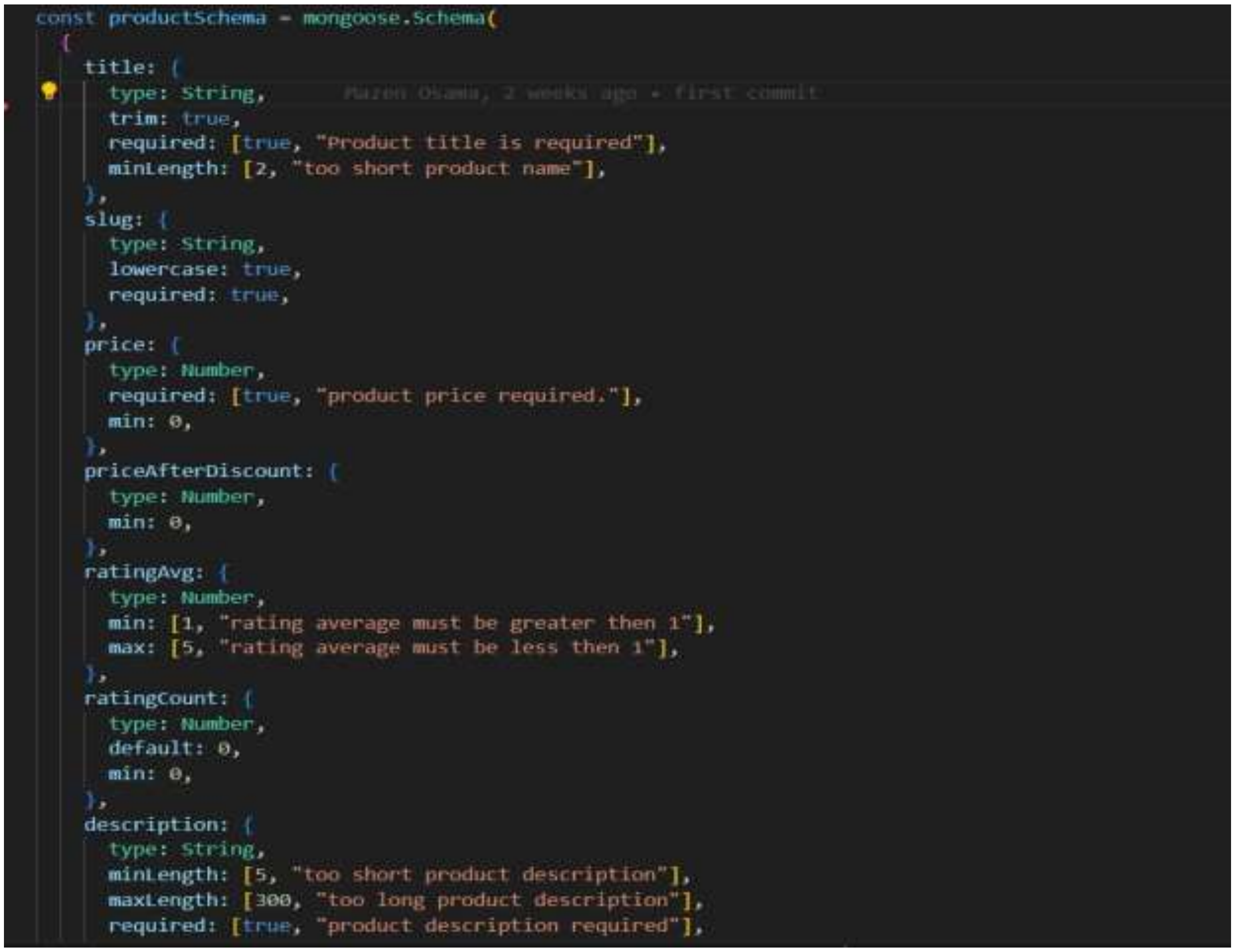
  in yourapplication.
- When a user registers, you'll create a new user document with the required Fields.
- The password hashing middleware ensures that passwords are

  securelystored.
- The Wishlist field allows users to save products they're interested in.
- Addresses can be stored in the address array.

Product API:

- Schema Structure:

The product Schema represents the structure of a product document in a MongoDB database.
It defines various fields that store information about a product.



```
const productSchema = mongoose.Schema(
  {
    title: {
      type: String,       Mazen Osama, 2 weeks ago • first commit
      trim: true,
      required: [true, "Product title is required"],
      minLength: [2, "too short product name"],
    },
    slug: {
      type: String,
      lowercase: true,
      required: true,
    },
    price: {
      type: Number,
      required: [true, "product price required."],
      min: 0,
    },
    priceAfterDiscount: {
      type: Number,
      min: 0,
    },
    ratingAvg: {
      type: Number,
      min: [1, "rating average must be greater then 1"],
      max: [5, "rating average must be less then 1"],
    },
    ratingCount: {
      type: Number,
      default: 0,
      min: 0,
    },
    description: {
      type: String,
      minLength: [5, "too short product description"],
      maxLength: [300, "too long product description"],
      required: [true, "product description required"],
```

- Fields Explained:
  - Title: A string field representing the product's title. It must be unique, have a Minimum length of 2 characters, and is required.
  - Slug: A string field storing a lowercase version of the title. It is also required.
  - Price: A numeric field representing the product's price. It must be greater than or Equal to 0 and is required.

- PriceAfterDiscount: An optional numeric field representing the product's price after applying any discounts.
- RatingAvg: A numeric field representing the average rating of the product. It must bebetween 1 and 5.
- Rating Count: An integer field representing the total count of ratings for the product(defaults to 0).
- Description: A string field containing the product description. It must have aminimum length of 5 characters and is required.
- Quantity: An integer field representing the available quantity of the product. Itdefaults to 0 and is required.
- Sold: An integer field tracking the total number of units sold (defaults to 0).
- ImgCover: A string field storing the URL or path to the product's cover image.
- Images: An array of strings representing additional images related to the product.
- Category: A reference to a category document (identified by its Object ID). It specifies the product's category and is required.
- Subcategory: A reference to a subcategory document (identified by its Object ID) It Specifies the product's subcategory and is required.
- Brand: A reference to a brand document (identified by its Object ID). It specifies the Product's brand and is required.

- Virtual Field:

  - The Review virtual field establishes a relationship with the "review" model It allows you to retrieve reviews associated with a product.
  - The local Field is set to _id, meaning it links the product's-id to the review's product field.

- Middleware:

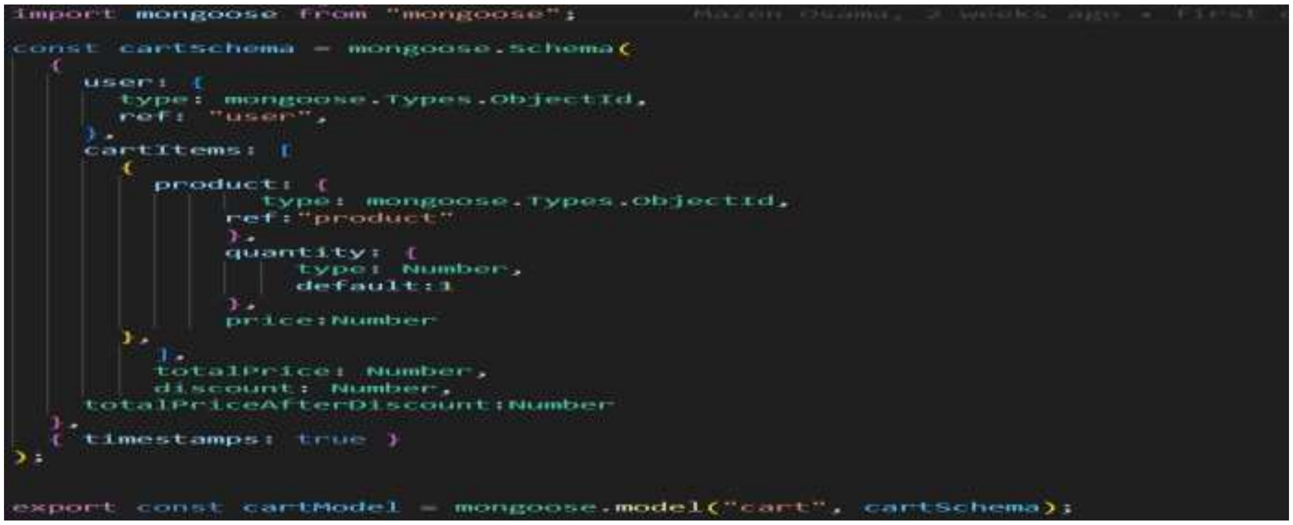
  - The post("init") middleware modifies the imgCover and images fields by appending
  - The base URL to their paths. This ensures that image URLs are complete.

- Usage:
  - You can use this schema to create and manage product information in yourapplication.
  - When adding a new product, populate the required fields (e.g., title, price,description, category, subcategory, and brand).
  - The virtual field my Review allows you to retrieve associated reviews for a product.
  - Ensure that you handle image paths and URLs correctly based on your application’s Setup.

Cart API:

- Schema Structure:

  - ➢ The cart Schema represents the structure of a shopping cart in a MongoDBdatabase.
  - It contains several fields, each serving a specific purpose.



```
import mongoose from "mongoose";          Mazen Osama, 2 weeks ago • First c

const cartSchema = mongoose.Schema(
  {
    user: {
      type: mongoose.Types.ObjectId,
      ref: "user",
    },
    cartItems: [
      {
        product: {
              type: mongoose.Types.ObjectId,
            ref:"product"
            },
            quantity: {
                type: Number,
                default:1
            },
            price:Number
      },
        ],
        totalPrice: Number,
        discount: Number,
    totalPriceAfterDiscount:Number
  },
  { timestamps: true }
);

export const cartModel = mongoose.model("cart", cartSchema);
```

- Fields Explained:

  - User: This field is of type mongoose.Types.ObjectIdand references a user document. It establishes a relationship between the cart and the user who owns it.
  - Cart Items: An array of objects representing individual items in the cart. Each itemhas the following properties:
    - Product: A reference to a product document (identified by its Object ID). Thislinks the cart item to a specific product.
    - Quantity: The quantity of the product in the cart. It defaults to 1 if not specified.
    - Price: The price of the product. You can set this value when adding items to thecart.
  - Total Price: Represents the total price of all items in the cart before any discountsare applied.
  - Discount: The discount amount (if any) applied to the cart.
  - TotalPriceAfterDiscount: The final total price after applying the discount.

- Timestamps:

  - The {timestamps: true} optionautomatically adds created at and updated at fieldsto the schema. These fields track when the cart was created or last updated.

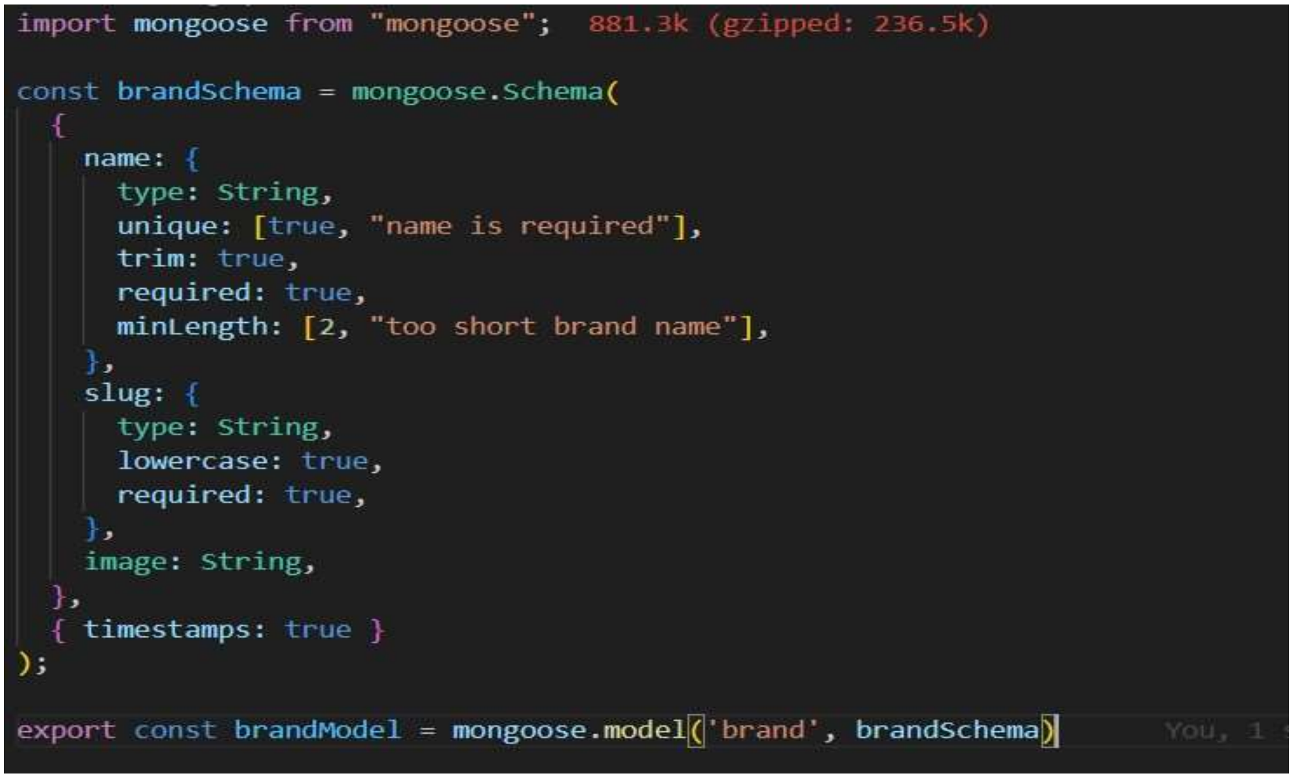
- Usage:

  - You can use this schema to create and manage shopping carts for users in your application.
  - When a user adds items to their cart, you'll create a new cart document with the Appropriate user reference and populate the cart Items array.

Brand API:

- Schema Structure:

  - The brand Schema represents the structure of a brand document in a MongoDB database.
  - It defines several fields related to brand information.



```
import mongoose from "mongoose";  881.3k (gzipped: 236.5k)

const brandSchema = mongoose.Schema(
  {
    name: {
      type: String,
      unique: [true, "name is required"],
      trim: true,
      required: true,
      minLength: [2, "too short brand name"],
    },
    slug: {
      type: String,
      lowercase: true,
      required: true,
    },
    image: String,
  },
  { timestamps: true }
);

export const brandModel = mongoose.model('brand', brandSchema)        You, 1
```

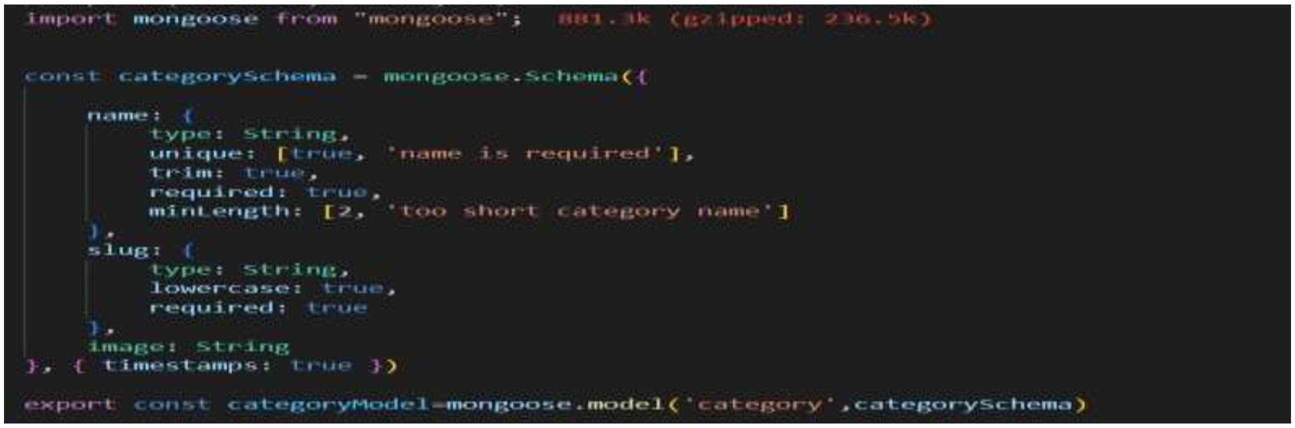
- Fields Explained:

  - Name: A string field representing the brand's name. It must be unique, have a minimum length of 2 characters, and is required.
  - Slug: A string field storing a lowercase version of the brand name. It is also required.
  - logo: A string field containing the URL or path to the brand's (brand-pic)

- Timestamps:
  - The {timestamps: true} option automatically adds created At and updated At fields to the schema. These fields track when the brand document was created or last updated.
- Middleware:
  - The post ("init") middleware modifies the logo field by appending the base URL to its path. This ensures that the logo URL is complete.
- Usage:
  - You can use this schema to create and manage brand information in your application.
  - When adding a new brand, populate the required fields (e.g., name, logo).
  - The slug field can be used for generating SEO-friendly URLs.
  - Ensure that you handle image paths and URLs correctly based on your application's setup.

Category API:

- Schema Structure:
  - The category Schema represents the structure of a category document in aMongoDB database.
  - It defines several fields related to category information.



```
import mongoose from "mongoose";  881.3k (gzipped: 236.5k)

const categorySchema = mongoose.Schema({

    name: {
        type: String,
        unique: [true, 'name is required'],
        trim: true,
        required: true,
        minLength: [2, 'too short category name']
    },
    slug: {
        type: String,
        lowercase: true,
        required: true
    },
    image: String
}, { timestamps: true })

export const categoryModel=mongoose.model('category',categorySchema)
```

- Fields Explained:

  - Name: A string field representing the subcategory’s name. It must be unique, have a minimum length of 2 characters, and is required.
  - Slug: A string field storing a lowercase version of the subcategory name. It is alsorequired.
  - Category: A reference to a category document (identified by its Object-Id). Itspecifies the parent category to which the subcategory belongs.
  - Image: A string field containing the URL or path to an image associated with thesubcategory.

- Timestamps:

  - The {timestamps: true} option automatically adds created at and updated at fields tothe schema. These fields track when the subcategory document was created or lastupdated.

- Middleware:

  - The post ('init') middleware modifies the image field by appending the base URL to itspath. This ensures that the image URL is complete.

- Usage:

  - You can use this schema to create and manage subcategory information in yourapplication.
  - When adding a new subcategory, populate the required fields (e.g., name, slug, category, and image). • The slug field can be used for generating SEO-friendly URLs.
  - Ensure that you handle image paths and URLs correctly based on your application’s setup.

Tech stack

1. Programming Languages:

- **Server-Side Languages:** since us developing back-end project, so we need to usea server-Side Languages. We used Node.js and here is the reason we used it.
    - **Node.js (Express.js):** Well-suited for real-time applications and APIs, oftenused with the Express.js framework for building web applications.

2. Frameworks:

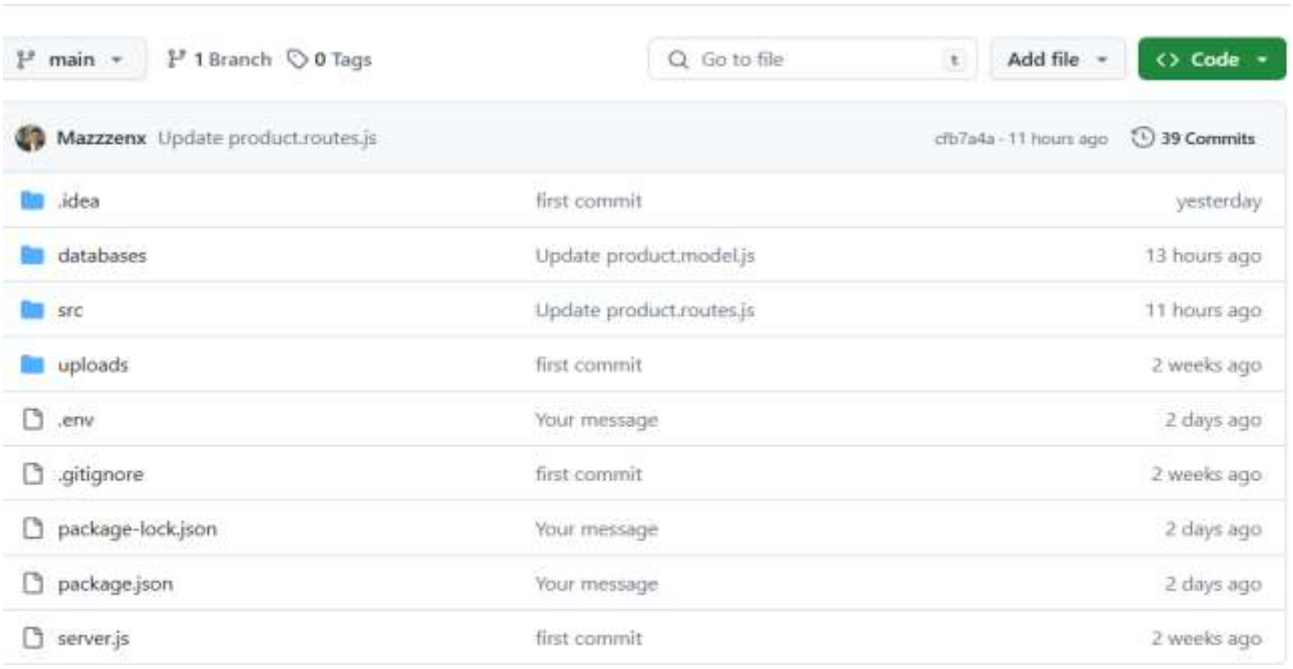
- These frameworks built on top of programming languages provide pre-builtstructures and functionalities, speeding up development:
    - We used {**Express.js**} **framework**

3. Databases:

- In our project we used **NoSQL Databases(** Flexible data storage forunstructured or semi-structured data), and it was:
    - **MongoDB**

Deployment

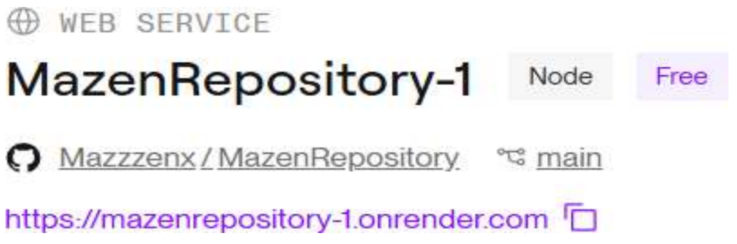
- We first uploaded the whole project on GitHub.



Mazzzenx Update product.routes.js cfb7a4a · 11 hours ago 39 Commits

.idea	first commit	yesterday
databases	Update product.model.js	13 hours ago
src	Update product.routes.js	11 hours ago
uploads	first commit	2 weeks ago
.env	Your message	2 days ago
.gitignore	first commit	2 weeks ago
package-lock.json	Your message	2 days ago
package.json	Your message	2 days ago
server.js	first commit	2 weeks ago

- Then we deployed it with OnReander:



WEB SERVICE

MazenRepository-1 Node Free

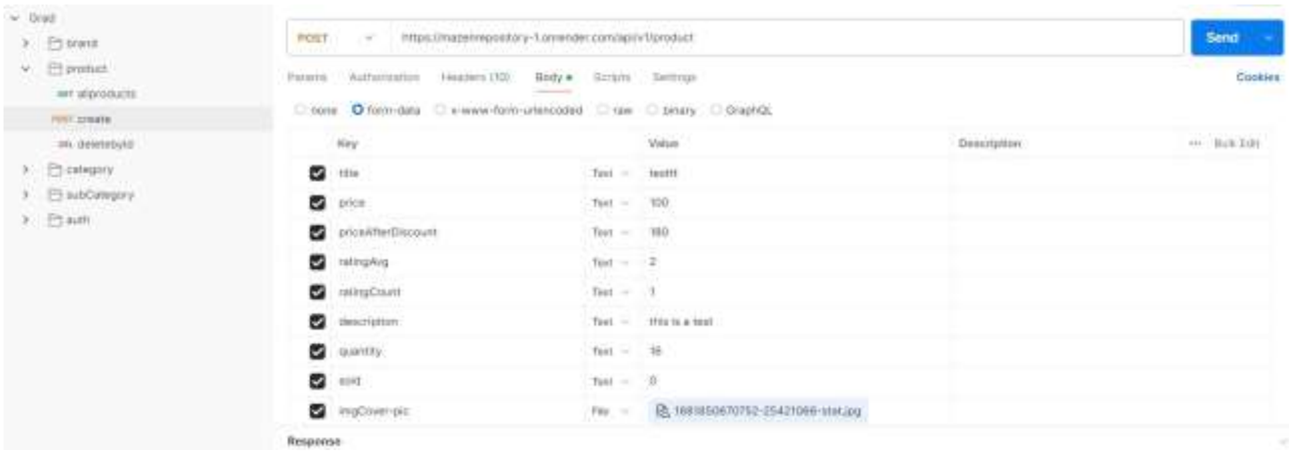
Mazzzenx / MazenRepository main

https://mazenrepository-1.onrender.com

Testing

API Testing: Testing the functionality, performance, and security of APIs used forcommunication between the backend and other systems.

We used tool (postman to test the API).



POST https://mazenrepository-1.onrender.com/api/v1/product

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

Key		Value	Description
title	Text	testtt	
price	Text	100	
priceAfterDiscount	Text	180	
ratingAvg	Text	2	
ratingCount	Text	1	
description	Text	this is a test	
quantity	Text	16	
sold	Text	0	
imgCover-pic	File	1681850670752-25421066-stat.jpg	

➢ Unit Testing:

- Focuses on isolating individual units of code (functions, classes) and testing theirfunctionality independently.
- Ensures each unit performs as expected with various inputs.
- Typically performed by developers during development.

➢ Integration Testing:

- Tests how different components of the backend system interact with eachother.
- Verifies data flow and communication between services like the Business LogicLayer and the Data Layer.
- May involve testing API interactions between the backend and the front-end

Tools

1. **IDF:** we used Visual studio because it is a powerful Integrated Development Environment (IDE), and it offers a comprehensive set of features to streamline the softwaredevelopment process (code editing, Debugging, testing tools).
2. **Joi Library:** This was used for data validation in the project.
3. **Mongoose Library:** we used this library to gain access to its functionalities for interactingwith MongoDB databases in your Node.js application
4. **Bcrypt Library:** is used for secure password hashing.
5. **Cloudnary Library:** used for Uploading and Managing Media
   - **Simplified Uploads:** The library provides methods for uploading media filesfrom your application to Cloudinary's secure storage.
   - **Automatic Transformations:** You can define transformations (resizing, cropping, watermarking) on the fly during upload, eliminating the need for pre-processing media files on your server.
   - **Delivery Optimization:** Cloudnary automatically delivers media assets in themost efficient format and size based on the user's device and network conditions.
6. **JWT Library:** JSON Web Tokens, JWT is commonly used for implementing authenticationand authorization in Node.js applications.
7. **Express Library:** Express.js is a minimal and flexible Node.js web application framework that simplifies routing, middleware integration, and handling HTTP requests and responses.
8. **Slugify Library:** This JavaScript library is designed to generate SEO (Search EngineOptimization) friendly slugs (short and descriptive text strings) from text content.
9. **QRCode Library:** used to gain access to functions and methods for generating QR Codes.
10. **Multer:** that simplifies handling multipart/form-data (data sent through HTML forms) which typically includes file uploads.

11. **Dotenv Library:** Storing sensitive information like API keys or passwords directly in code is a securityrisk. Version control systems can expose this information if not properly configured

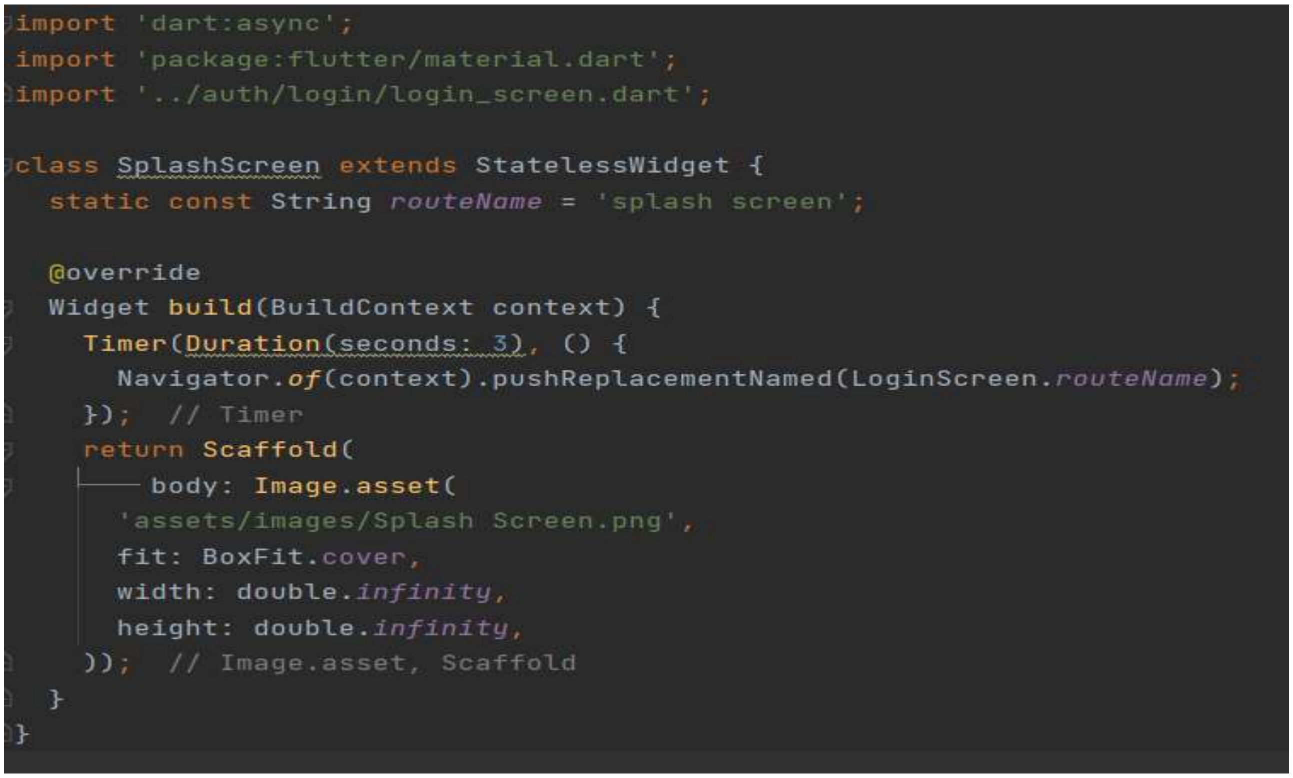
.

- Hardcoding environment variables directly in your code makes it difficult to managedifferent configurations (development, testing, and production) that might require different values.
  - ➢ The dotenv library addresses these challenges.

12. **Morgan library:** middleware library used for HTTP request logging, It sits in the request-response cycle of your application and logs information about incoming HTTP requests.
13. **CROS library:** It stands for Cross-Origin Resource Sharing and plays a crucial role inenabling secure communication between different web applications.

Flutter

➢ Splash Screen



```
import 'dart:async';
import 'package:flutter/material.dart';
import '../auth/login/login_screen.dart';

class SplashScreen extends StatelessWidget {
  static const String routeName = 'splash screen';

  @override
  Widget build(BuildContext context) {
    Timer(Duration(seconds: 3), () {
      Navigator.of(context).pushReplacementNamed(LoginScreen.routeName);
    }); // Timer
    return Scaffold(
        body: Image.asset(
      'assets/images/Splash Screen.png',
      fit: BoxFit.cover,
      width: double.infinity,
      height: double.infinity,
    )); // Image.asset, Scaffold
  }
}
```

- **Dart: async:** This package provides classes and functions to work with asynchronous operations in Dart.

- **Package: flutter/material.dart:** This package contains the Flutter framework, including widgets and material design components.

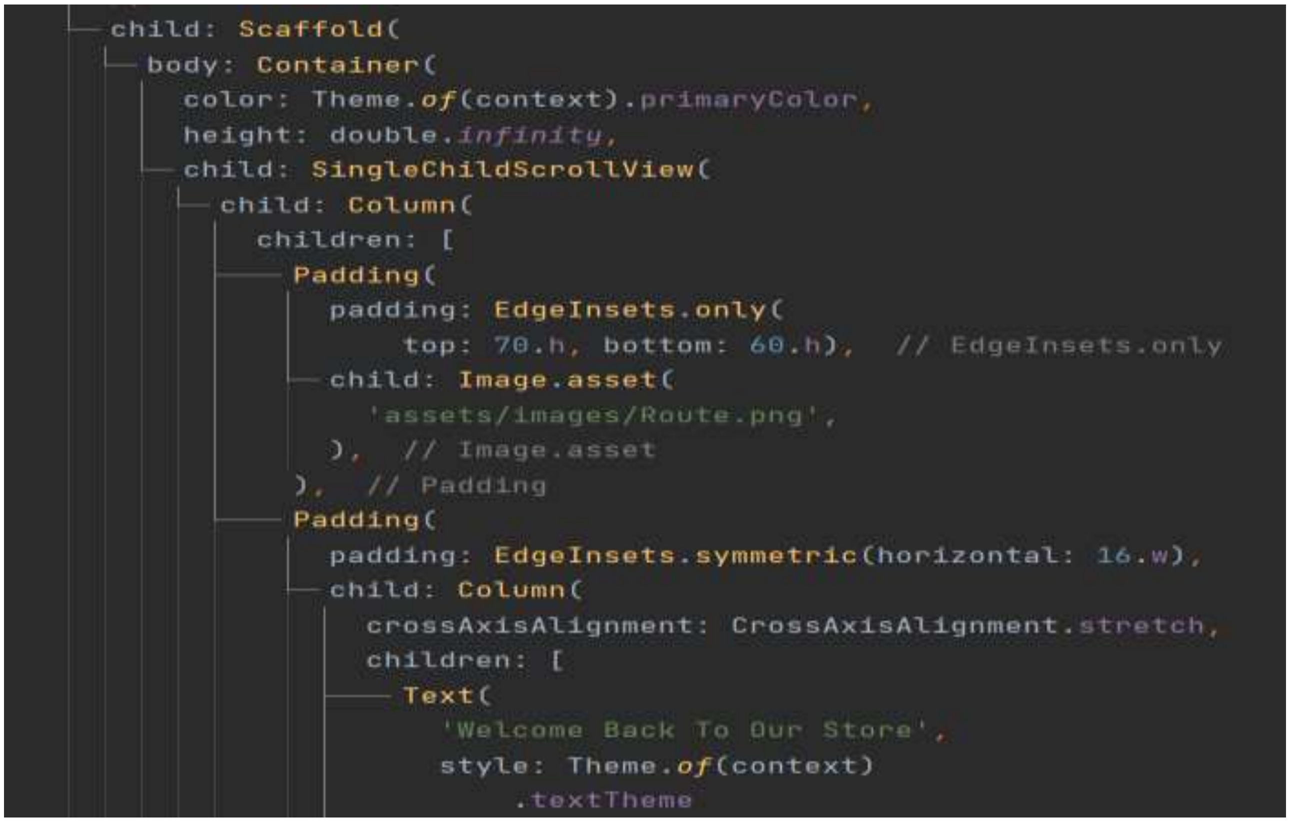
- **'../auth/login/login_screen.dart':** This imports the Login Screen class from the login_screen.dart file located in the auth/login directory relative to the current file.

- **Build (Build Context context):** This method overrides the build method from the Stateless Widget class and returns a widget tree.
- Timer (Duration (seconds: 3), () {...}): This creates a timer that fires after 3 seconds. When the timer expires, it executes the provided function. In this case, it navigates to the login screen.

- **Navigator. Of (context)**.pushReplacementNamed (LoginScreen.routeName): This line navigates to the login screen by pushing the login screen route onto the navigation stack and replacing the current route.
- **Scaffold:** This widget provides a basic layout structure for the screen.
- **Body:** Image. Asset (...): This sets the body of the scaffold to an image loaded from an asset file.

- **'assets/images/Splash Screen.png':** This is the path to the image asset representing the splash screen image.
- **Fit: BoxFit.cover:** This property specifies how the image should be resized to fit the screen. In this case, it covers the entire screen.
- **Width: double. Infinity:** This sets the width of the image to fill the entire width of the screen.
- **Height: double. Infinity:** This sets the height of the image to fill the entire height of the screen.

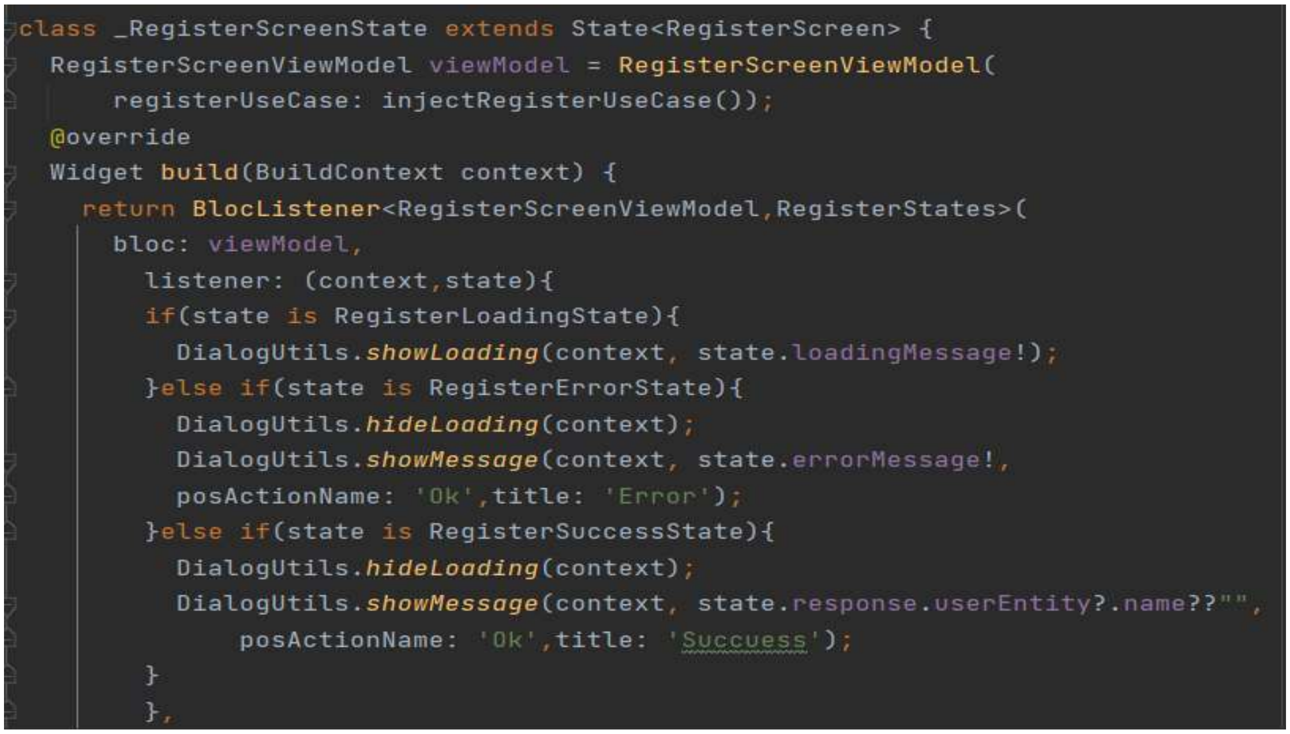
➢Login Screen



```
    child: Scaffold(
      body: Container(
        color: Theme.of(context).primaryColor,
        height: double.infinity,
        child: SingleChildScrollView(
          child: Column(
            children: [
              Padding(
                padding: EdgeInsets.only(
                    top: 70.h, bottom: 60.h),  // EdgeInsets.only
                child: Image.asset(
                  'assets/images/Route.png',
                ),  // Image.asset
              ),  // Padding
              Padding(
                padding: EdgeInsets.symmetric(horizontal: 16.w),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.stretch,
                  children: [
                    Text(
                      'Welcome Back To Our Store',
                      style: Theme.of(context)
                          .textTheme
```

- **Build (Build Context context):** Builds the UI for the login screen.
- **Bloc Listener:** Listens to state changes from the LoginScreenViewModel.
- **Scaffold:** Provides the basic material design visual layout structure.
- **Container:** Contains the main content of the screen.
- **SingleChildScrollView:** Enables scrolling if the content exceeds the screen height.
- **Column:** Arranges child widgets vertically.
- Various widgets like Padding, Image. Asset, Text, Form, TextFieldItem, Elevated Button, and Row are used to construct the UI elements of the login screen.

➢ Register Screen

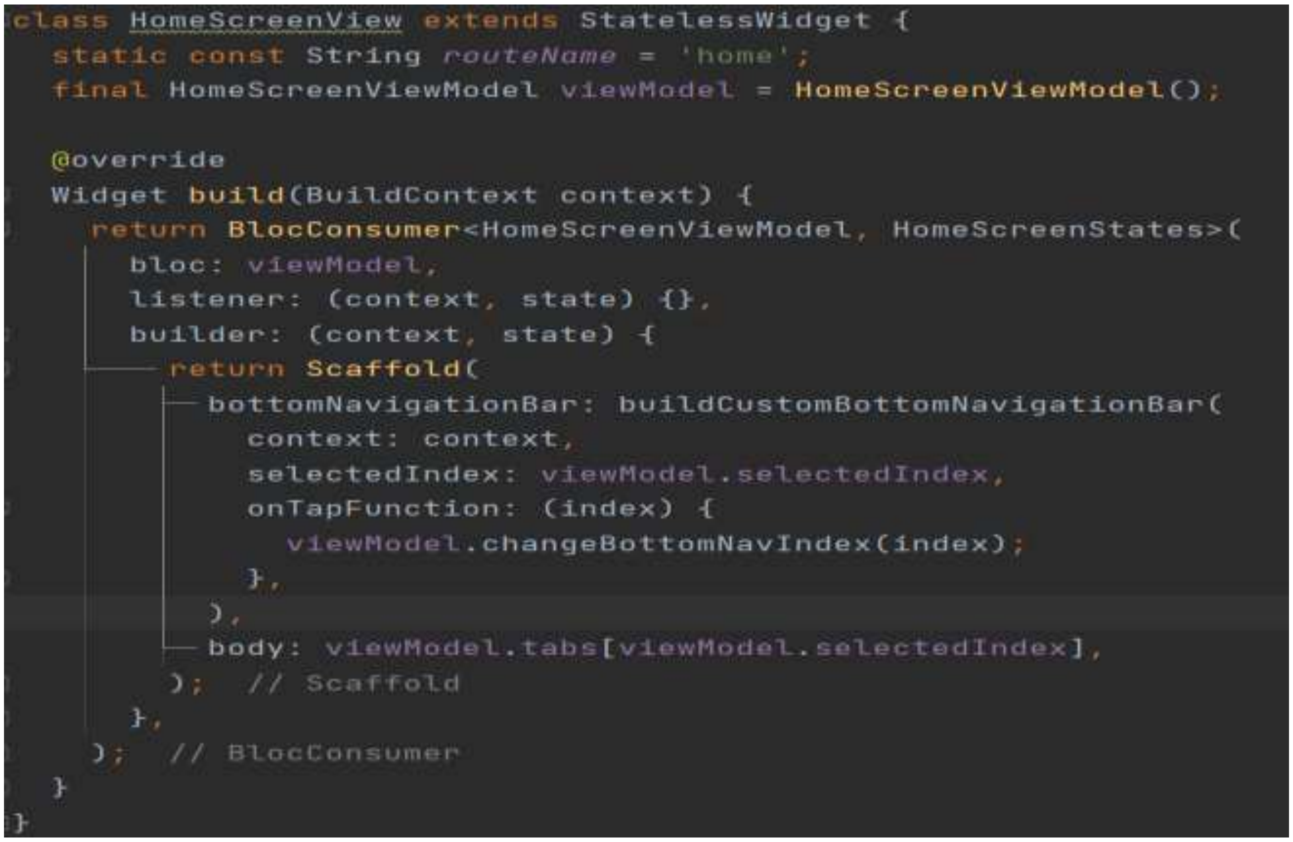


```
class _RegisterScreenState extends State<RegisterScreen> {
  RegisterScreenViewModel viewModel = RegisterScreenViewModel(
      registerUseCase: injectRegisterUseCase());
  @override
  Widget build(BuildContext context) {
    return BlocListener<RegisterScreenViewModel,RegisterStates>(
      bloc: viewModel,
        listener: (context,state){
        if(state is RegisterLoadingState){
          DialogUtils.showLoading(context, state.loadingMessage!);
        }else if(state is RegisterErrorState){
          DialogUtils.hideLoading(context);
          DialogUtils.showMessage(context, state.errorMessage!,
          posActionName: 'Ok',title: 'Error');
        }else if(state is RegisterSuccessState){
          DialogUtils.hideLoading(context);
          DialogUtils.showMessage(context, state.response.userEntity?.name??"",
              posActionName: 'Ok',title: 'Succuess');
        }
        },
```

- **Build (Build Context context):** Builds the UI for the register screen.
- **Bloc Listener:** Listens to state changes from the RegisterScreenViewModel.
- **Scaffold:** Provides the basic material design visual layout structure.
- **ExtendBodyBehindAppBar:** Extends the body behind the app bar for a full screen effect.
- **AppBar:** Defines the app bar with a transparent background and no elevation.
- **Icon Button:** Adds a back button to navigate back to the previous screen.
- **Container:** Contains the main content of the screen.

- **SingleChildScrollView:** Enables scrolling if the content exceeds the screen height.
- **Column:** Arranges child widgets vertically.
- Various widgets like Padding, Image. Asset, Form, TextFieldItem, Elevated Button, etc., are used to construct the UI elements of the register screen.

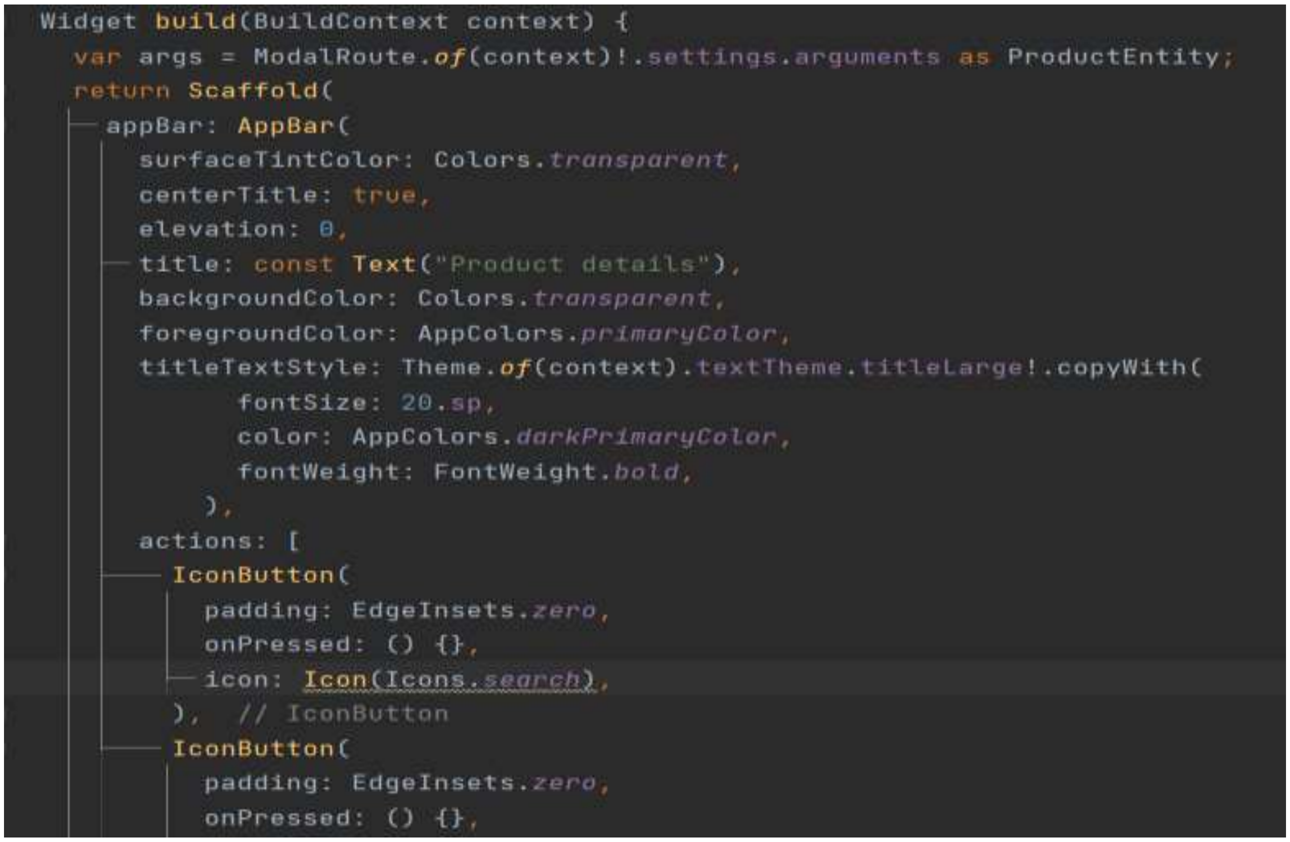
➢Home Screen



```
class HomeScreenView extends StatelessWidget {
  static const String routeName = 'home';
  final HomeScreenViewModel viewModel = HomeScreenViewModel();

  @override
  Widget build(BuildContext context) {
    return BlocConsumer<HomeScreenViewModel, HomeScreenStates>(
      bloc: viewModel,
      listener: (context, state) {},
      builder: (context, state) {
        return Scaffold(
          bottomNavigationBar: buildCustomBottomNavigationBar(
            context: context,
            selectedIndex: viewModel.selectedIndex,
            onTapFunction: (index) {
              viewModel.changeBottomNavIndex(index);
            },
          ),
          body: viewModel.tabs[viewModel.selectedIndex],
        );  // Scaffold
      },
    );  // BlocConsumer
  }
}
```

- **HomeScreenView:** Represents the view of the home screen of the application.
- **Route Name:** Represents the route name for the home screen.
- **View Model:** Instance of HomeScreenViewModel to manage the state of the home screen.
- **Build (Build Context context):** Builds the UI for the home screen.
- **Bloc Consumer:** Listens to state changes from the HomeScreenViewModel.
- **Scaffold:** Provides the basic material design visual layout structure.
- **BottomNavigationBar:** Custom bottom navigation bar widget that changes the selected index based on user interaction.
- **Body:** Displays the content of the current tab based on the selected index

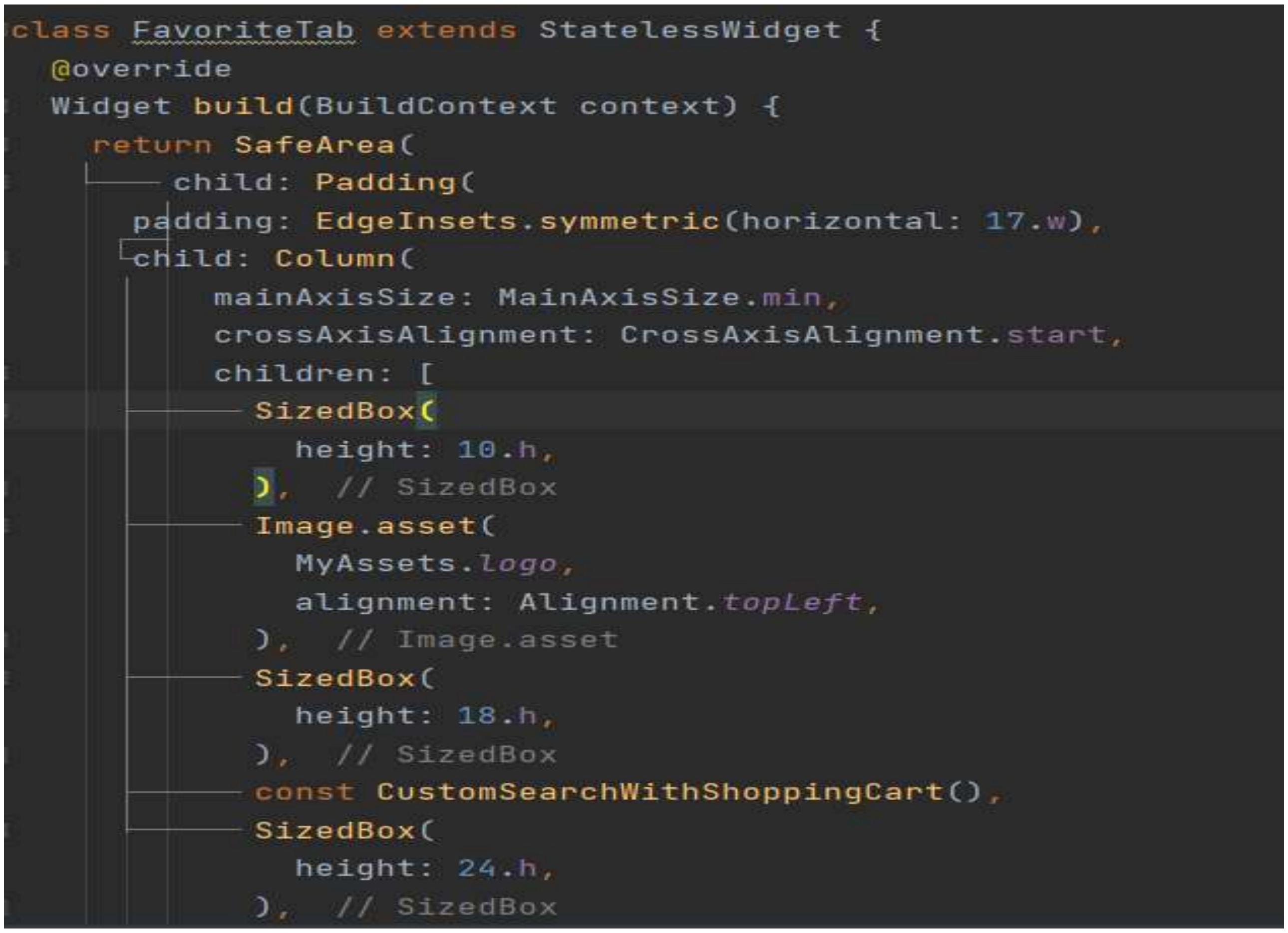
➢Product Details



```
Widget build(BuildContext context) {
  var args = ModalRoute.of(context)!.settings.arguments as ProductEntity;
  return Scaffold(
    appBar: AppBar(
      surfaceTintColor: Colors.transparent,
      centerTitle: true,
      elevation: 0,
      title: const Text("Product details"),
      backgroundColor: Colors.transparent,
      foregroundColor: AppColors.primaryColor,
      titleTextStyle: Theme.of(context).textTheme.titleLarge!.copyWith(
            fontSize: 20.sp,
            color: AppColors.darkPrimaryColor,
            fontWeight: FontWeight.bold,
          ),
      actions: [
        IconButton(
          padding: EdgeInsets.zero,
          onPressed: () {},
          icon: Icon(Icons.search),
        ),  // IconButton
        IconButton(
          padding: EdgeInsets.zero,
          onPressed: () {},
```

- **Scaffold:** Provides the basic material design visual layout structure.
- **AppBar:** Configures the app bar with title and actions.
- **Padding:** Ads padding around the main content.
- **SingleChildScrollView:** Enables scrolling if the content exceeds the screen height.
- **Column:** Arranges child widgets vertically.
- **RoductDetailsView:** Represents the view for displaying product details.
- **Route Name:** Represents the route name for the product details view.
- **RoductDetailsView:** Represents the view for displaying product details.
- **Route Name:** Represents the route name for the product details view.

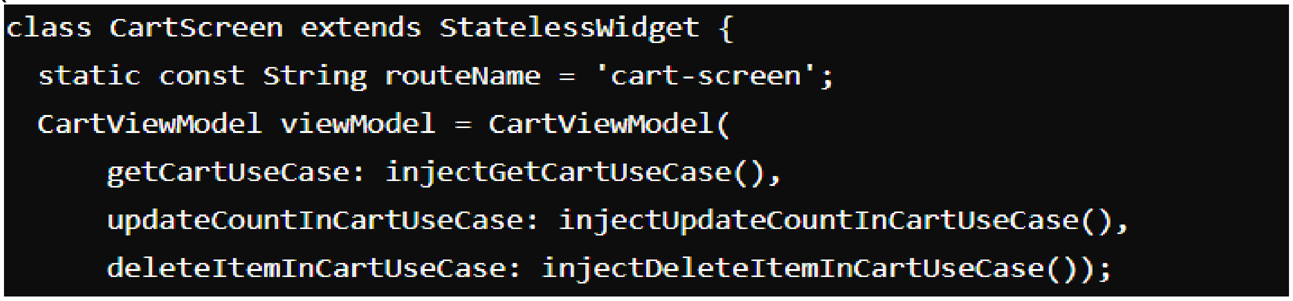
➢ Favorite Screen



```
class FavoriteTab extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return SafeArea(
        child: Padding(
      padding: EdgeInsets.symmetric(horizontal: 17.w),
      child: Column(
          mainAxisSize: MainAxisSize.min,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            SizedBox(
              height: 10.h,
            ),  // SizedBox
            Image.asset(
              MyAssets.logo,
              alignment: Alignment.topLeft,
            ),  // Image.asset
            SizedBox(
              height: 18.h,
            ),  // SizedBox
            const CustomSearchWithShoppingCart(),
            SizedBox(
              height: 24.h,
            ),  // SizedBox
```

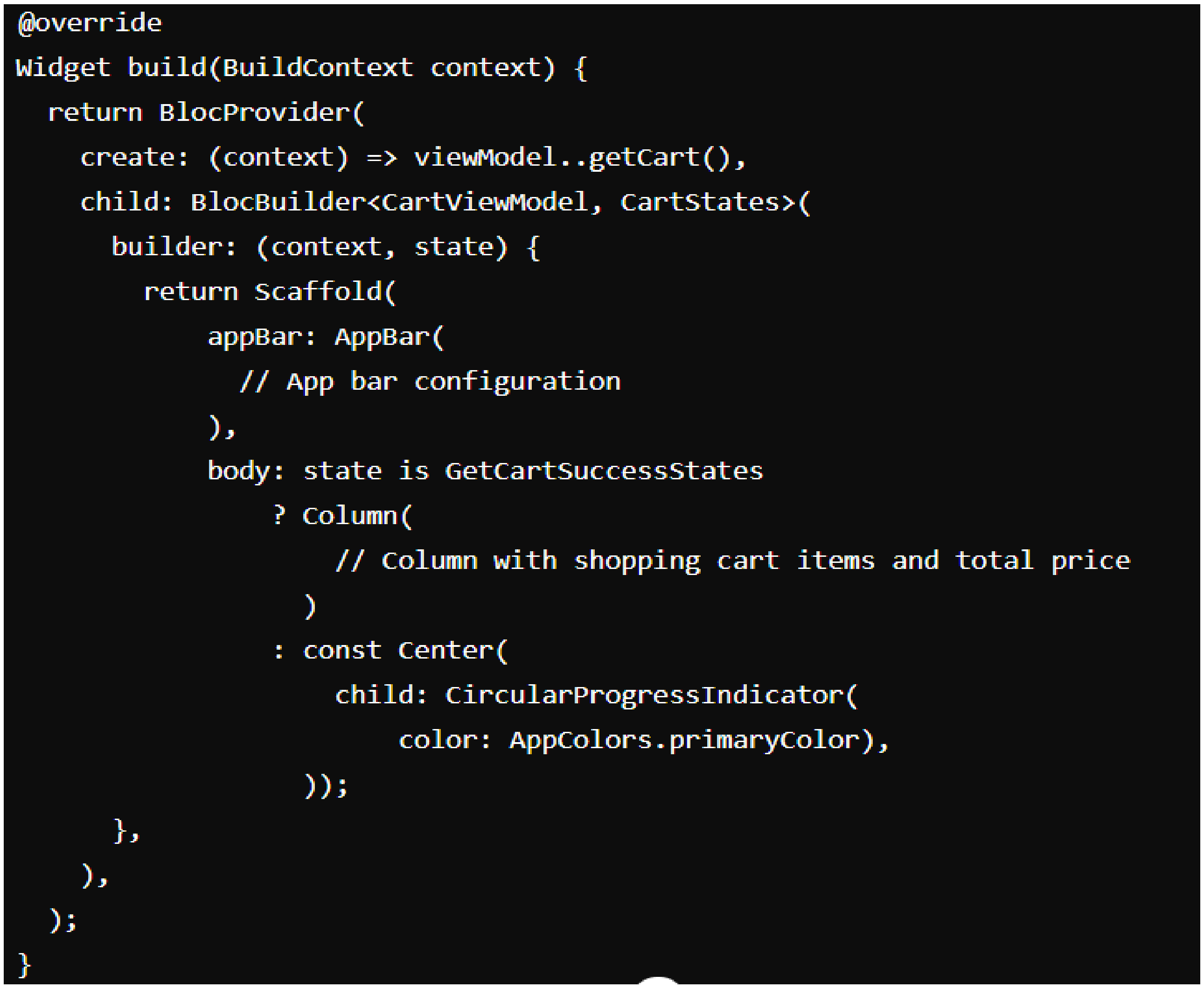
- **Favorite Tab:** Represents the tab for displaying favorite items.
- **Build (Build Context context):** Builds the UI for the favorite tab.
- **Safe Area:** Widget to ensure content is displayed within safe areas of the screen.
- **Padding:** Adds padding around the content.
- **Column:** Arranges child widgets vertically.
- **Sized Box:** Provides spacing between widgets.
- **Image. Asset:** Displays an image from assets.
- **MyAssets.logo:** Asset path for the logo image.
- **CustomSearchWithShoppingCart:** Custom widget for search with shopping cart.

➢ Cart Screen



```
class CartScreen extends StatelessWidget {
  static const String routeName = 'cart-screen';
  CartViewModel viewModel = CartViewModel(
      getCartUseCase: injectGetCartUseCase(),
      updateCountInCartUseCase: injectUpdateCountInCartUseCase(),
      deleteItemInCartUseCase: injectDeleteItemInCartUseCase());
```

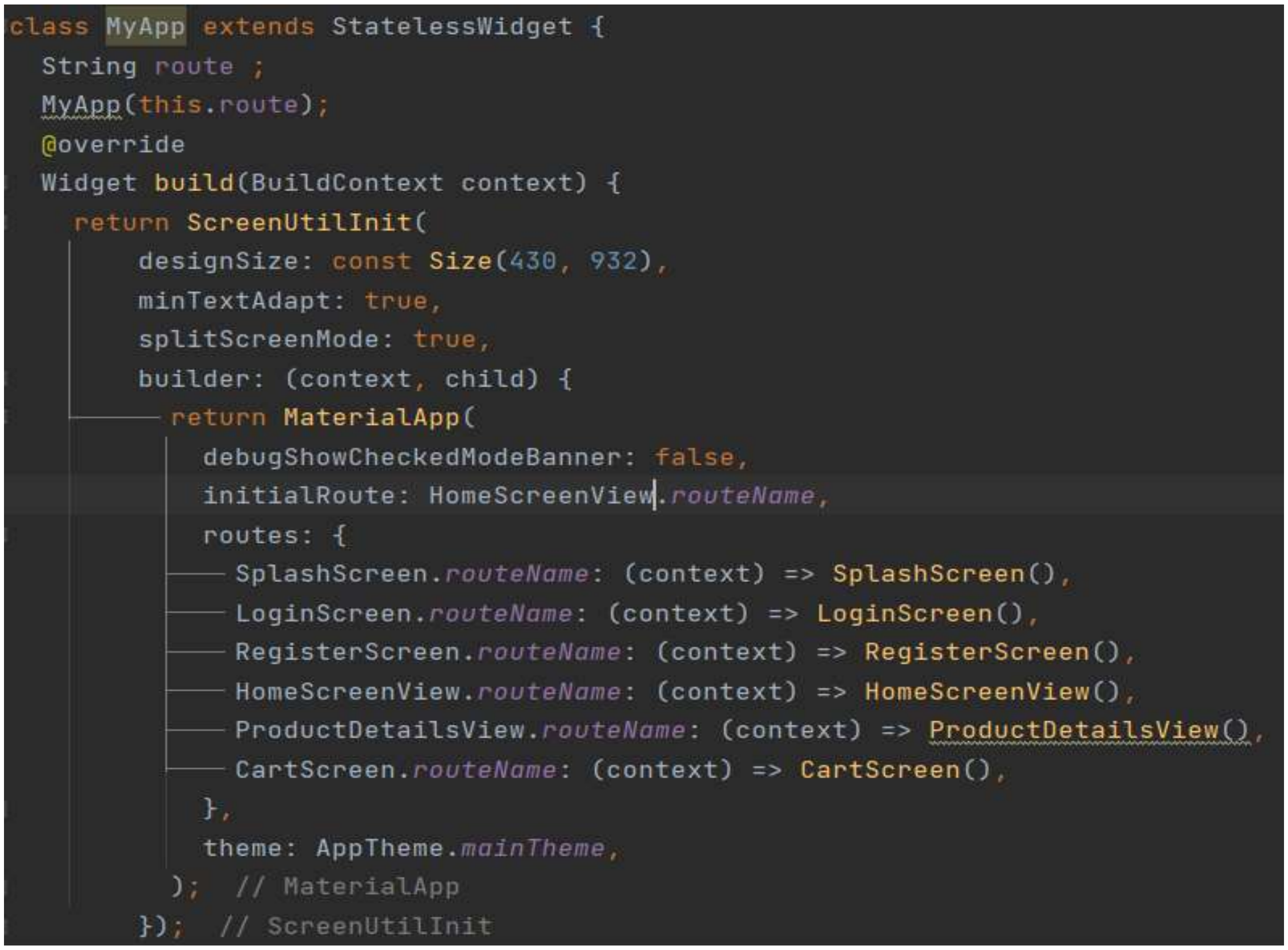
- **Cart Screen:** Represents the screen for displaying the shopping cart.
- **RouteName:** Represents the route name for the cart screen.
- **CartViewModel:** Creates an instance of the cart view model with injected dependencies.



```
@override
Widget build(BuildContext context) {
  return BlocProvider(
    create: (context) => viewModel..getCart(),
    child: BlocBuilder<CartViewModel, CartStates>(
      builder: (context, state) {
        return Scaffold(
            appBar: AppBar(
              // App bar configuration
            ),
            body: state is GetCartSuccessStates
                ? Column(
                  // Column with shopping cart items and total price
                )
                : const Center(
                  child: CircularProgressIndicator(
                      color: AppColors.primaryColor),
                ));
      },
    ),
  );
}
```

- **Bloc Provider:** Provides the cart view model to its descendants.
- **Create:** Creates an instance of the cart view model and initializes the cart data.
- **Bloc Builder:** Builds the UI based on the state changes in the cart view model.
- **Scaffold:** Provides the basic material design visual layout structure.
- **AppBar:** Configures the app bar with title and actions.
- **Body:** Displays different content based on the state of the cart.

➢Main Screen



```
class MyApp extends StatelessWidget {
  String route ;
  MyApp(this.route);
  @override
  Widget build(BuildContext context) {
    return ScreenUtilInit(
        designSize: const Size(430, 932),
        minTextAdapt: true,
        splitScreenMode: true,
        builder: (context, child) {
          return MaterialApp(
            debugShowCheckedModeBanner: false,
            initialRoute: HomeScreenView.routeName,
            routes: {
              SplashScreen.routeName: (context) => SplashScreen(),
              LoginScreen.routeName: (context) => LoginScreen(),
              RegisterScreen.routeName: (context) => RegisterScreen(),
              HomeScreenView.routeName: (context) => HomeScreenView(),
              ProductDetailsView.routeName: (context) => ProductDetailsView(),
              CartScreen.routeName: (context) => CartScreen(),
            },
            theme: AppTheme.mainTheme,
          );  // MaterialApp
        });  // ScreenUtilInit
```

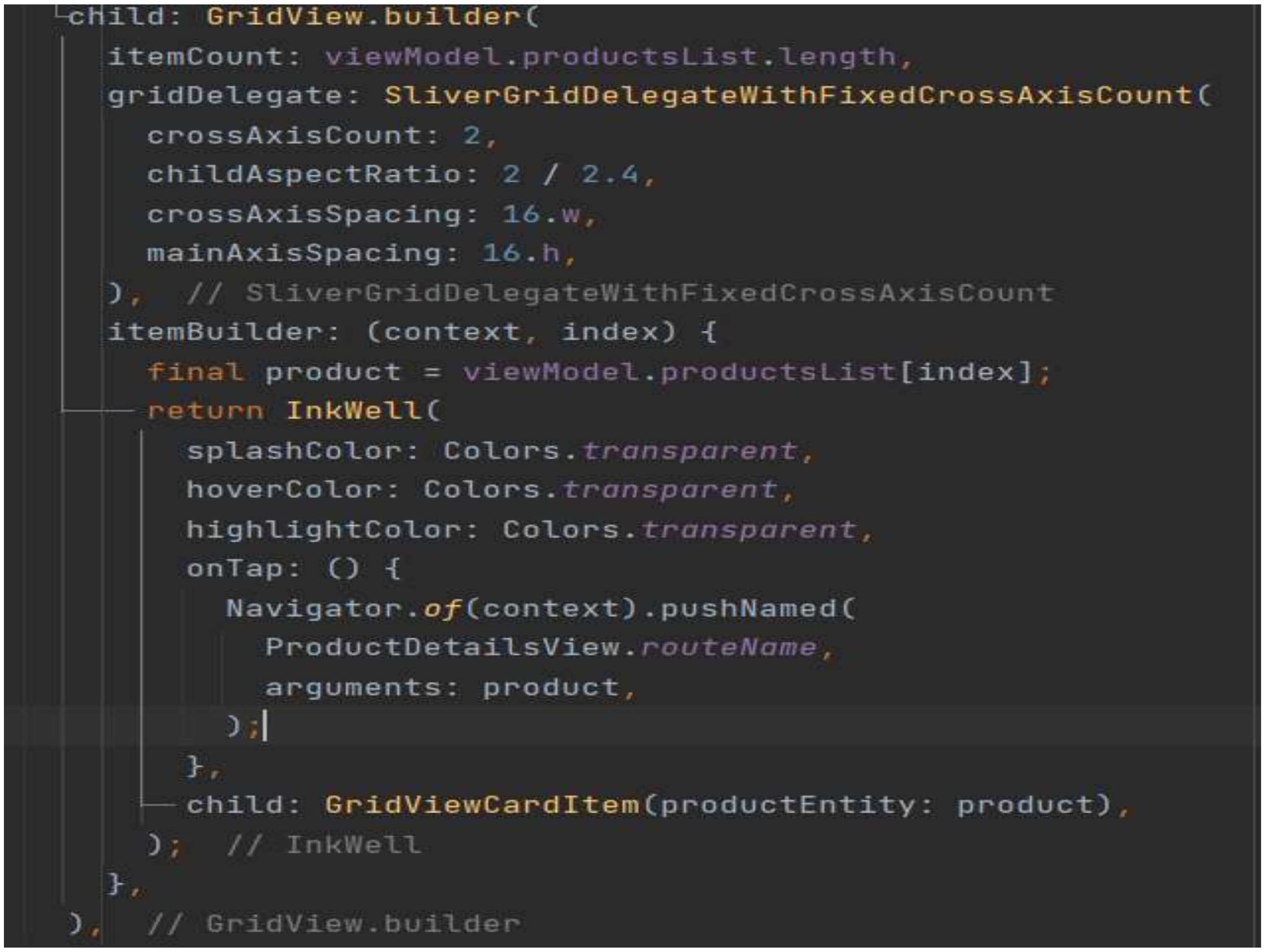
- **Main ():** Entry point of the Flutter application.
- **WidgetsFlutterBinding.ensureInitialized ():** Ensures that Flutter binding is initialized before executing further code.
- **Bloc. Observer = MyBlocObserver ():** Sets a custom observer for Bloc to observe state changes.
- **Await SharedPreferenceUtils.init ():** Initializes shared preferences for storing user data.

- Determines the initial route based on whether a user is logged in or not.
- **RunApp (MyApp (route)):** Runs the application with the specified initial route.

➢Product List Screen

```
class ProductListTab extends StatelessWidget {
  final ProductListTabViewModel viewModel = ProductListTabViewModel(
    addToCartUseCase: injectAddToCartUseCase(),
    getAllProductsUseCase: injectGetAllProductsUseCase(),
  );
```

- **ProductListTab:** Represents a tab that displays a list of products.
- **View Model:** Instance of the view model responsible for managing state and business logic.



```
  child: GridView.builder(
    itemCount: viewModel.productsList.length,
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 2,
      childAspectRatio: 2 / 2.4,
      crossAxisSpacing: 16.w,
      mainAxisSpacing: 16.h,
    ), // SliverGridDelegateWithFixedCrossAxisCount
    itemBuilder: (context, index) {
      final product = viewModel.productsList[index];
      return InkWell(
        splashColor: Colors.transparent,
        hoverColor: Colors.transparent,
        highlightColor: Colors.transparent,
        onTap: () {
          Navigator.of(context).pushNamed(
            ProductDetailsView.routeName,
            arguments: product,
          );
        },
        child: GridViewCardItem(productEntity: product),
      ); // InkWell
    },
  ), // GridView.builder
```

- **Build ():** Builds the widget tree for the product list tab.
- **Bloc Provider:** Provides the product list tab view model to its descendants.
- **Safe Area:** Ensures that content is displayed within the safe area of the screen.
- **CustomTextField:** Custom widget for search functionality.
- **Badge:** Widget for displaying a badge with the number of items in the cart.
- Displays a loading indicator if the state is ProductListTabLoadingStates.
- Displays a grid view of products when the loading state is finished.
- **GridView.builder:** Builds a grid view of product items using a builder function.
- **Inkwell:** Provides the ability to tap on product items to navigate to their details.
- **GridViewCardItem:** Widget representing an item in the grid view of products.

Chapter 6

Future Work

Future Work

- **Security Enhancements**: Strengthen security measures by implementing multi-factor authentication, encryption of sensitive data, and regular security audits to protect user information and prevent data breaches.

- **Customer Support and Feedback Mechanisms**: Integrate live chat support, Chabot, or help centers within the app to provide immediate assistance to users and gather feedback for continuous improvement.

- **Multiple Payment Options**: Offer a variety of payment methods to cater to different user preferences. This may include credit/debit cards, digital wallets (e.g., PayPal, Google Pay, and Apple Pay), bank transfers, cash on delivery (COD), and cryptocurrency payments.

- **One-Click Payment**: Implement one-click payment functionality to allow returning customers to complete purchases quickly without having to re-enter payment details, enhancing convenience and reducing friction in the checkout process.

Chapter 7

References

API Used

References

Dataset for laptops:
https://github.com/37Degrees/DataSets/blob/master/laptops.csv

Data set for cameras:
https://www.kaggle.com/datasets/crawford/1000-cameras-dataset

Data set For Adidas clothes:
https://data.world/data-hut/product-data-from-adidas

Data set For Nike Products:
https://www.kaggle.com/datasets/adwaitkesharwani/nike-product-descriptions/

API Used

- Category:
  https://mazenrepository1.onrender.com/api/v1/category
- Brand:
  https://mazenrepository1.onrender.com/api/v1/brand
- Subcategory:
  https://mazenrepository-1.onrender.com/api/v1/subCategory
- Product:
  https://mazenrepository-1.onrender.com/api/v1/product