Online Furniture Marketing System

Prepared By:

Eslam saber Elfar

Ahmed Khaled Elsharabasy

Mazen Hassan Elalfy

Mohammed Ahmed Nasr

Rowida Mohammed Abdelaal

Omnia Azmy Elqzaaz

Shrouk Sheriff Elsweesy

Sama Maged Rizk

Supervised by:

DR\ Amira sedeek Elzeiny

2022/2023

Abstract:

The world of commerce has transformed significantly in the past few decades with the introduction of E-commerce. E-commerce, or electronic commerce, refers to the buying and selling of goods and services through the internet. The emergence of E-commerce has transformed the traditional brick-and-mortar retail industry, making it more accessible, efficient, and convenient for consumers worldwide. One area where E-commerce has particularly flourished is the furniture industry, with the rise of multivendor E-commerce platforms for furniture. In this abstract, we will explore the impact of E-commerce on the furniture industry and the rise of multivendor E-commerce platforms.

E-commerce has revolutionized the furniture industry, making it more accessible and convenient for consumers to purchase furniture online. The convenience of online shopping allows consumers to browse through various options and compare prices from the comfort of their homes. Additionally, E-commerce has eliminated the geographical barriers for consumers, allowing them to purchase furniture from anywhere in the world. This has increased competition among furniture retailers, leading to more competitive prices and greater choices for consumers.

The rise of multivendor E-commerce platforms in the furniture industry has further transformed the industry. Multivendor E-commerce platforms are online marketplaces where multiple vendors can sell their products under one platform. These platforms allow vendors to access a wider audience and gain more exposure, while consumers can access a variety of products from different vendors. The benefits of multivendor E-commerce platforms are numerous, including greater product variety, competitive prices, and improved customer service. Moreover, the ease of setting up an online store on a multivendor platform has made it more accessible for small and medium-sized businesses to enter the furniture industry.

Despite the benefits, multivendor E-commerce platforms face several challenges. The first challenge is maintaining quality control across different vendors. With multiple vendors selling products on the same platform, it can be challenging to ensure consistent quality across all products. Another challenge is maintaining customer trust and loyalty. As multivendor platforms host multiple vendors, it is essential to ensure that vendors adhere to ethical business practices and provide high-quality products and services.

In conclusion, E-commerce has transformed the furniture industry, making it more accessible, efficient, and convenient for consumers worldwide. The rise of multivendor E-commerce platforms has further revolutionized the industry, providing greater product variety and competitive prices for consumers. However, multivendor E-commerce platforms also face several challenges, such as maintaining quality control and ensuring customer trust and loyalty. Overall, the emergence of E-commerce and multivendor platforms in the furniture industry has been a game-changer, providing consumers with more choices and opportunities to purchase high-quality furniture online.

Acknowledgment:

First and foremost, we owe a special dept of gratitude to our supervisor who graciously provided use with her experience and helping shape and direct our interest into a more focused and defensible work and without her this work would not have been possible. So, we would like to express our special thanks and deepest gratitude to **DR\ Amira sedeek Elzeiny.**

We would also thank each of whom brought their individual expertise and knowledge to bear and had an active role in shaping the final product, without naming all of them, we would like to single out for praise **DR\ Reda Elbarogy.**

We would like to thank our friends and family whose love and support sustained us throughout the project. This project spanned several months and required a great deal of time and energy.

Finally, we had a great time in the Faculty of Computer and Information Sciences, we learned many things, and now it's the time to use what we had learnt to produce the best of ours.

Table of Contents:

List of Figures:

List of Tables:

List of Abbreviations:

DFD: data flow diagrams.

ERD: entity-relationship diagrams.

E-commerce: electronic commerce.

CHAPTER 1

INTRODUCTION

1. Introduction:

In this chapter, we will talk about an overview of our furniture online store system, the purpose of the project, the problems it solves, list the objectives of the project to be achieved, clarify new contributions to the project, and a brief description of the content of the document and what each chapter will contain.

1.1. Overview of the system:

Our Furniture Online Store system is a comprehensive e-commerce solution that includes both a website and a mobile application. The Furniture Online Store system is a comprehensive platform consisting of three key components: seller, management, and client. The system allows clients to browse and purchase furniture products with ease and allows sellers to upload product information, manage inventory, and track orders. management platform provides administrators with tools for overseeing inventory, analyzing sales data, and managing the website and application. The Home Bazaar application and website aim to provide customers with a convenient and hassle-free shopping experience for furniture items, offering a wide range of products, easy comparison of features and prices, and the ability to shop from anywhere at any time.

The platform includes various features such as a search function to help customers find specific furniture items, and a filtering system to narrow down search results based on customer preferences. High-quality images of furniture items, along with descriptions, dimensions, and pricing are displayed to provide customers with detailed information about each product. Additionally, the app and website include user reviews, ratings, and recommendations to help customers make informed purchasing decisions.

The Furniture Online Store system is designed to offer a seamless shopping experience for customers, while also providing sellers and management with a powerful platform to manage their operations effectively.

1.2. Motivations and Problem Statement:

The furniture industry is shifting towards online shopping, and there is a demand for a multi-vendor furniture e-commerce platform.

Customers expect a wide range of furniture options, and a multi-vendor platform can provide a comprehensive selection of products.

A multi-vendor platform gives businesses a competitive advantage by tapping into a larger vendor network and offering a more extensive product catalog.

Participating in a multi-vendor platform enables vendors to expand their reach and access a larger pool of potential customers, increasing sales and business growth.

A multi-vendor platform simplifies the shopping experience by consolidating various vendors' products into a single website.

- Quality assurance standards can be established to ensure that all vendors meet certain criteria, building trust with customers.
- A multi-vendor platform can provide centralized logistics support, streamlining the process and improving efficiency for both vendors and customers.
- Effective vendor management systems and communication channels should be implemented to address the challenges of coordinating with multiple vendors.
- Mechanisms for monitoring and resolving customer complaints, handling returns and refunds, and ensuring overall customer satisfaction should be established to protect the platform's reputation.
- Implementing a multi-vendor system requires careful integration with existing infrastructure to ensure a seamless transition and compatibility with current operations.

1.3. Objectives of the system:

- Saving time and effort for the customer, as the customer can request any order at any time.
- Provide high quality service to all its customers in a professional, kind, and supportive manner.
- The purchase process is always available anytime and anywhere.
- Treat every supplier, employee, and customer with honesty, dignity, and respect.
- Provide a safe and convenient environment to shop.
- Easy to view all products to customers without moving from my place and get rid of traveling and waiting in crowds.
- Impress our customers, current and prospective, to encourage future business.
- Improve all aspects of service delivery to our customers, our employees, and our community.
- Provide a comfortable environment to know customers' opinions of products without any embarrassment and use them to improve service in the future.

1.4. Contributions:

- Unfinished furniture: If you are looking to get furniture pieces of white wood before finishing, there is no better than our site to provide this feature for you. There will be a special section for white wood that gives the customer the chance to see the details of the product in its early stages in terms of wood type User, material, and quality in detail.
- Special pieces: We will find a distinguished and unique group of furniture pieces that reflect the style and heritage of the city of Damietta, inspired by the arts of different eras. There will also be a story for each product that tells the details of the arts from which each piece was designed.
- Ask the system: the customer can give a picture of the piece he wants if he does not find it among the available products, and it will be manufactured for him.
- Evaluation: The customer can express his opinion about the experience of the site and the offered products, and this is a basic motive for developing the system.

1.5. Project Outlines:

This project consists of six chapters in addition to one appendix. These chapters are organized to reflect the scientific steps toward our main objective. A brief description about the contents of each chapter is given in the following paragraphs:

Chapter 1: Introduces an overview/background of the project, the project objectives, the motivation and Problem Statement of the project, the contributions of this project and project timeline.

Chapter 2: Provides the reader with a review of the relevant work, and the relationship between our work and the relative ones.

Chapter 3: Provides an analysis of the existing systems, our system's requirements, and the System Architecture.

Chapter 4: Introduces a main image of the system design which contains the class diagram, database design and interface design.

Chapter 5: Provides quite understandable samples of the system implementation which contains a mapping design to the implementation, samples of the code, system testing, results and the goals achieved.

Chapter 6: Introduces a very good conclusion and a part of our plan for the project in the future.

References

1.6. Project Timeline:

#	Task name	Start	Finish
1	Background research	**2-10-2022**	**23-10-2022**
2	Determine objectives	**25-10-2022**	**3-11-2022**
3	Review related work	**15-11-2022**	**22-11-2022**
4	System planning	**25-11-2022**	**8-12-2022**
5	System Analysis	**11-12-2022**	**27-12-2022**
6	System Design	**29-12-2022**	**19-1-2023**
7	Presentation for phase 1	**28-1-2023**	
8	System Implementation	**20-2-2023**	**10-6-2023**
9	Presentation for phase 2	**26-6-2023**	

Table 1.6 project timeline.

CHAPTER 2

Literature Review

2.1. Similar existing systems:

In this chapter we talk about other project that are concerned with Selling furniture online mentioning most important advantages and disadvantages, comparing between our project and the existing work.

1-Manzzeli is an online furniture and home decor store that offers a wide range of products, including sofas, chairs, tables, and decorative items. The website has an elegant design with high-quality images that showcase their products.

Pros:

1- Large variety of furniture and home decor products to choose from.

2- User-friendly website interface that makes it easy to navigate and find products.

3- High-quality product images and detailed product descriptions.

Cons:

1- Prices may be higher compared to other online furniture stores in Egypt.

2- Don't have ability to express opinion

2-Tavolo Furniture is an Egyptian online store that offers a wide range of furniture products, including sofas, beds, cabinets, and dining tables. The website has a simple design and is available in both English and Arabic.

Pros:

1- Large variety of furniture products to choose from, including modern and classic styles.

2- Provides a 10-year warranty for all furniture products.

3- Easy and secure payment methods, including cash on delivery, bank transfer, and credit/debit card.

Cons:

1- Limited customization options for furniture products.

2- The return policy is not clearly stated on the website

3-Smart furniture is a furniture and home decor store, will help you create the most comfortable and cozy living

space through innovative, European quality and yet affordable prices, available in both English and Arabic.

pros:

1- Trendy and well-designed solutions.

2- Free delivery and installation.

3- Through it, it is possible to pay in installments in different ways without interest.

cons:

1- The rooms are sold complete, non-controllable products.

2- The website does not offer international shipping.

4-Homzmart is an e-commerce platform that sells home furnishings and goods and an integrated community that connects home professionals with homeowners who can browse home designs, furniture, accessories, and products.

It is the one shop where you can buy everything from furniture to homewares and more.

pros:

1- Combines the power of technology and logistics to provide a seamless customer experience.

2- It offers convenience, predictive technology, and a customer-centric approach.

3- Has the largest selection of designing and lighting your home in one click.

cons:

1-Do not accept any claims arising from errors or omissions.

2-It does not guarantee the quality, value or marketability of the products and it is the responsibility of both the seller and the buyer to inspect all goods before they are sold.

2.2. Core elements:

- Search Function: The platform includes a search function that allows customers to search for specific furniture items by keyword or filter results based on various criteria such as price range, color, material, and more.
- Product Catalog: The platform includes a comprehensive catalog of furniture items, with high-quality images, detailed descriptions, and pricing information for each product.
- User Reviews and Ratings: Customers can leave reviews and ratings for products they have purchased, providing valuable feedback for other shoppers, and helping them make informed purchasing decisions.
- Wish List and Favorites: Customers can save items to their wish list or favorites for easy access and future reference.
- Shopping Cart: The platform includes a shopping cart feature that allows customers to add items to their cart, review their selections, and proceed to checkout.
- Order Tracking: Customers can track their orders and receive updates on the status of their delivery.
- Payment Options: The platform supports various payment options, including credit cards, debit cards, and online payment services such as PayPal.
- Customer Support: The platform provides customer support via comment to assist customers with any questions or issues they may have.
- Related Products: The platform includes a "related products" feature that suggests additional products to customers based on machine learning algorithms. This feature helps customers discover new products that may be of interest to them and encourages cross-selling and upselling opportunities for the seller.

CHAPTER 3

System Analysis

3. System Analysis:

In this chapter we provide a detailed knowledge about our system including functional requirements, non-functional requirements, system architecture, use cases, and sequence diagrams. We can divide the process of analysis into two parts, which are determining requirements and determining system architecture and development methodology.

3.1. System Requirements:

System requirements are the needed configurations for the system to operate efficiently. The next two subsections will discuss the functional requirements, Nonfunctional requirements.

3.1.1. Functional requirements:

Functional requirements refer to the specific features and functionalities that a system or software applications must be provided to meet the needs of its users. These requirements are typically expressed in terms of input and output behaviors, and they define the system's core purpose and capabilities.

Functional requirements	Description
Seller	• Register • Login • Add his personal information. • Add payment information. • Add product information. • Sending Complaints through mailbox for the management.
Management	• Manage backup for the system continuously. • Update data of users and store and analyze it.

	• Add or delete or modify or check user's account's information. • Provide Technical support for complaints and modification requests in the system from employees. • Control show or hide products advertisements. • Control permissions of users (clients and sellers). • Control and manage payments and fulfillment of purchase conditions.
Client	• Search products. • Choose product specifications from gallery filter. • Browse advertisements to see offers and ads. • Choose product from categories. • Choose payment method. • Add the number of orders of product to cart to collect orders, then buy to purchase product. • Click Remove to remove selected orders from card. • Click Choose your location to determine the delivery location. • Click Buy to purchase and deliver it. • Click number of stars of product rating. • Add review comment about product. • Send complaints to return the product if it's wrong product or it's not in good condition. • Sending Complaints through mailbox for the management.

Table 3.1.1 Functional requirements.

3.1.2. Nonfunctional requirements:

Non-functional requirements, on the other hand, are the qualities or characteristics that describe how the system should perform or behave. These requirements are not directly related to the system's core purpose, but rather they define the attributes that impact the overall user experience, such as performance, reliability, security, usability, and scalability.

Non-Functional requirements	Description
Operational	• Cross platforms. • The seller should return the product if it's the wrong product or it's not in good condition. • The system will be user friendly, consistent, intuitive, and descriptive UI. • The system should have a maximum of 2:3 clicks to reach any content. • The system should be evolvable, scalable, testable, and maintainable.
Performance	• Quick ask & response (under 2 sec). • The system should be available 24 hours a day. • 20 minutes are specified daily on a fixed date [3 am (low-intensity hours)] to maintain and update the system.
Security	• The system must protect the user's privacy. • Every user should have a user-customizable visibility policy for his/her personal information such as email and study timetable. • The system will be secure and cannot affect, harm, damage the users or their devices.

Cultural & Political	• The system should provide commonly used calendar formats. • The system should use a related time zone. • The system should use the currency of our country.
Usability	• Easy to use and understand for different users.

Table 3.1.2 Non-Functional requirements.

3.2. System Architecture:

System architecture outlines the structure and components of a system. It contains several key elements, including data flow diagrams (DFD), use cases, entity-relationship diagrams (ERD), and structure charts. A DFD represents the flow of data within a system and helps to identify inputs, processes, and outputs. Use cases, on the other hand, define the functionality of a system from a user's perspective. ERD depicts the relationships between different data entities and how they interact with each other. Lastly, structure charts provide a visual representation of the system's structure and help to identify different modules and their interconnections. Overall, these elements work together to create a comprehensive system architecture that ensures the smooth functioning of the software system.

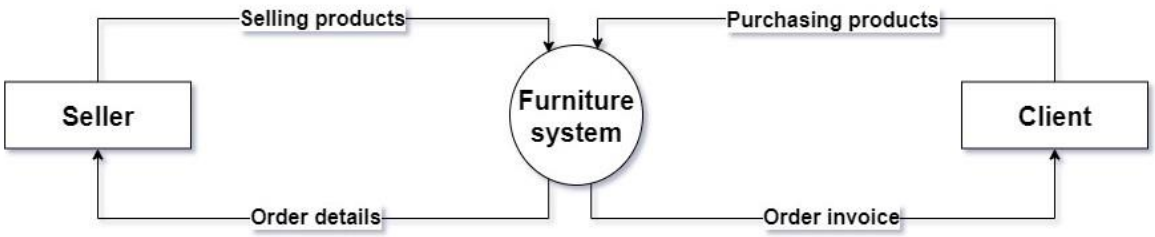


Figure 3.2.1 Context diagram of the whole system.

3.2.1 Data Flow Diagram:

The Data Flow Diagram (DFD) for the Furniture Online Store consists of three main parts: the seller, the management, and the clients. The seller enters their information into the system, including their product details, which are then stored in the product database. Clients can search for products on the website and add them to their cart, and place orders. The management team has overall control of the system and is responsible for setting constraints as well as overseeing the shipping process. The DFD shows how data flows between the different parts of the system, and how they interact with each other to ensure smooth operation of the online store.
Figure 3.2.2 illustrates the DFD.

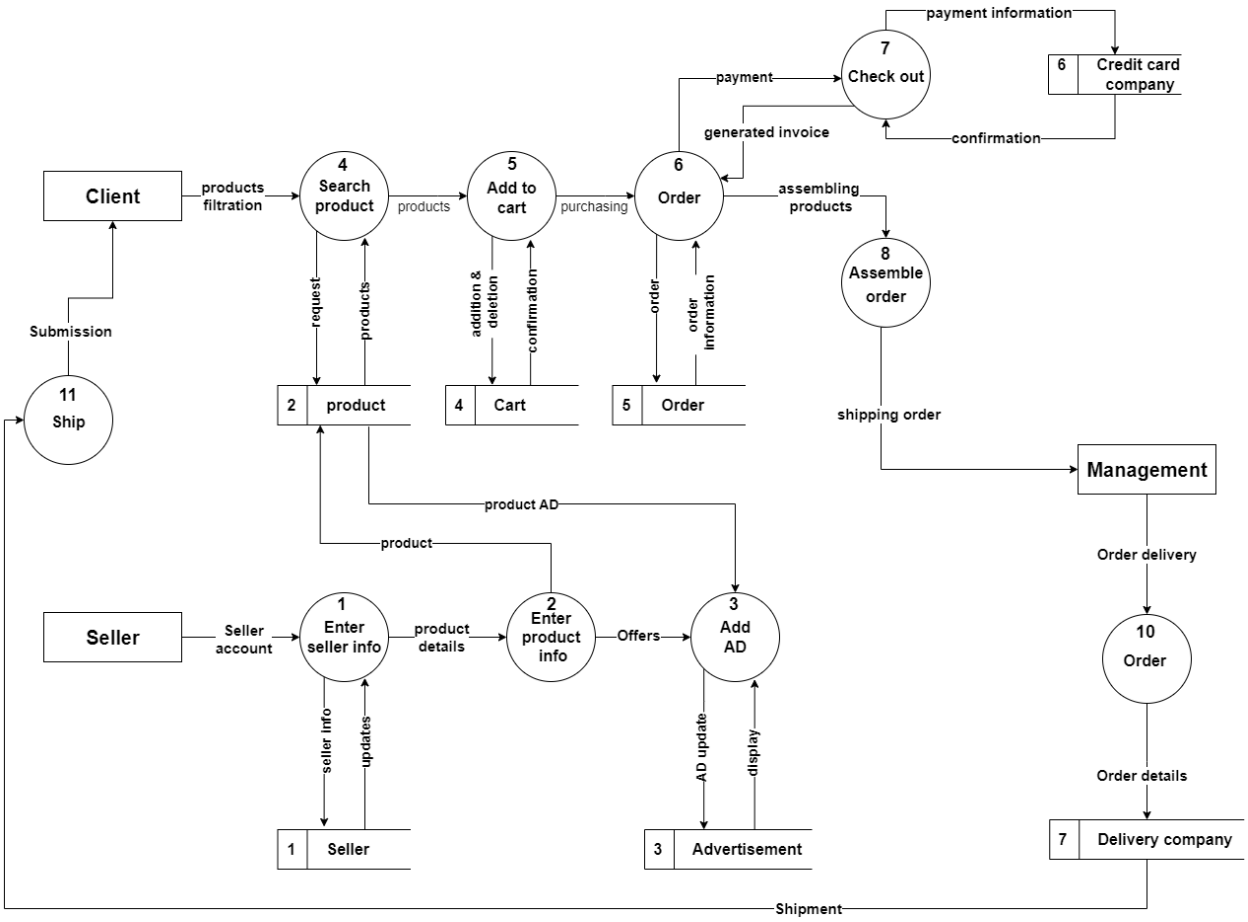


Figure 3.2.2 Data Flow Diagram of the system.

3.2.2 Use Cases:

These Use Cases describe the main actions and interactions between the different actors in the system and help to identify the key requirements and functionalities of the online store. They can be used to guide the design and development of the system, ensuring that it meets the needs and expectations of all stakeholders.

Figure 3.2.3 illustrates the Use Case Diagram of the Whole system.

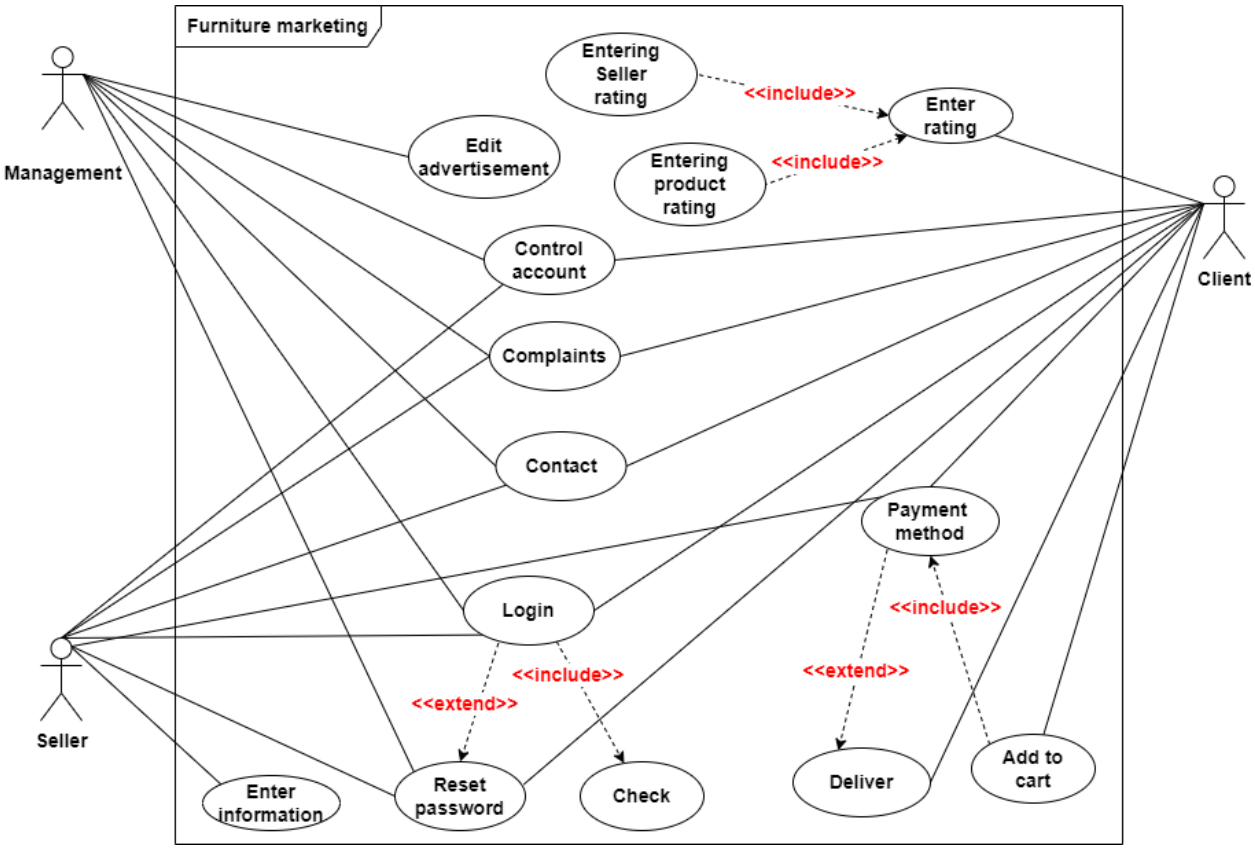


Figure 3.2.3 Use Case Diagram of the system.

The following sequence of tables describes use cases of the important functionalities of the system.

Use Case Name: Login	ID: 1	Priority: High
Actor: Manager, Seller, Client		
Description: This use case describes how a user (Manager, Seller, Client) logs into the System by using their account information before they can use system.		
Trigger: User requests to login. Type: ☑ External Temporal		
Precondition: 1. The user has not already logged into the system. 2. The user is trying to log in with their account.		
Normal Course: 1. User accesses the URL 2. The system prompts the user for their account credentials. 3. The user enters their username and password. 4. The system authenticates the login. 5. The user gains access to the systems functionality.	Information for steps: ← Email account ← Password account → Request confirmation → Control functions	
Alternative: 1. Invalid account user or password. 2. User already logged into the system before.	→ Reset password	
Postcondition: 1. The user is logged in to the system. 2. The user has access to the functions of the system.		
Exceptions: 1. Incorrect password or email. 2. Server crash. 3. Power cut off.		

Summary Inputs	Source	Outputs	Destination
• Email • Password	• User (Manager, Seller, Client).	• Request confirmation • Reset password	• User (Manager, Seller, Client).

Table 3.2.1 Login use case.

Use Case Name: Resetting Password	ID: 2	Priority: High
Actor: Manager, Seller, Client		
Description: This use case helps the user to set a new password.		
Trigger: User clicks on “FORGOT PASSWORD?!”. Type: ☑ External Temporal		
Precondition: 1. The user forgets the password. 2. The user can’t access the system using their password.		

Normal Course:	Information for steps:
1. The user writes the correct username and the incorrect password.	
2. The system shows an “invalid username or password” message error.	← Username and password.
3. The user clicks on the “FORGOT PASSWORD” link.	→ Error message.
4. The system displays the page where the user needs to answer security questions about their username and phone number(verification).	
5. The user answers the questions.	← Answers.
6. The system sends an SMS code to the phone number for verification and prompts the user to re-enter the code.	→ Code sent to the user.
7. The user enters the code.	← Users enter code.
8. The system displays whether the code is correct or not, if wrong the system generates a new code and sends it.	→ Error message.
9. If right, the system will update user new password.	→ Update new password.

Alternative:	
1. Admin creates a new password and sends it through the user email If the system is hacked.	Reset password. Send password.

Summary

Inputs	Source	Outputs	Destination
• Answers question • phone number • Verification code • New password	• User (Students and Staff info) • System code generator	• Verification • confirmation	• User (Manager, Seller, Client)

Table 3.2.2 Resetting Password use case.

Use Case Name: Enter information	ID: 10	Priority: High
Actor: Seller		
Description: This use case describes how the seller enter his information, then enter product information.		
Trigger: Seller enters his information and product information. Type: ☑ External Temporal		
Precondition: 1. The System must authenticate that the seller enters all his information before entering product information.		

Normal Course:	Information for steps:
1. The seller enters his information.	← Seller database.
2. The seller enters product information.	← Inventory database.
3. The seller updates the product information.	← Inventory database.

Alternative:	
1. The seller not complete entering information, then return to complete it	← Seller database. ← Inventory database.

Postcondition:
1. The system will send notification or email to the seller if the product has been sold.
2. The seller should be checking continuously the website for contacts or sold products.

Exceptions:
1. product issues.
2. Server crash.
3. Power cut off.

Summary Inputs	Source	Outputs	Destination
• Seller information • Product information	• Seller	• Product with a description.	• clients

Table 3.2.3 Enter information use case.

Use Case Name: Complaints	ID: 5	Priority: Low
Actor: Manager, Seller, Client		
Description: This use case describes how a user (Seller, Client) Sending Complaints through mailbox for the management.		
Trigger: problems face the user (Seller, Client). Type: ☑ External Temporal		
Precondition: 1. Problems face users. 2. Users can't solve these problems. 3. The product is not in good condition when it delivered to client.		
Normal Course: 1. The user click on "Complaints." 2. The user writes his complaint, then sends it to the management. 3. The management received the complaint message.	Information for steps: → Complaint table. ← Complaint content. → Complaint content.	
Alternative: 1. Invalid account user or password. 2. User already logged into the system before.	→ Reset password	
Postcondition: 1. Solving problems of users. 2. If it is a fraud case, the owner's account will be deleted, and the police will be reported with this fraud.		
Exceptions: 1. Incorrect password or email. 2. Server crash. 3. Power cut off. 4. Fraud process.		

Summary Inputs	Source	Outputs	Destination
• Complaint message.	• User (Seller, Client).	• Solving problem.	• Management.

Table 3.2.4 Complaints use case.

Use Case Name: Control account	ID: 3	Priority: Low
Actor: Manager, Seller, Client		
▪ Description: This use case describes how a user (Manager, Seller, Client) update his account information (ex. Address, phone, name).		
Trigger: Update user information. Type: ☑ External Temporal		
Precondition: 1. The user logged into the system.		

Normal Course:	Information for steps:
1. The user authenticates the login.	← Verification.
2. The user click on Control account update his information.	→ Show information.
3. The user update account information (ex. address).	← Update database.
4. The user click on image position to upload his picture.	← Update database.
Alternative: 1. User send complaints about problems with his information.	← Send complaints.

Postcondition:
1. The user save his changes.
2. The user receive confirmation message.

Exceptions:
1. Failed to update information.
2. Failed to upload his picture.
3. Server crash.
4. Power cut off.

Summary Inputs	Source	Outputs	Destination
• User information • User picture	• User (Manager, Seller, Client)	• Updated information • Confirmation message	• User (Manager, Seller, Client)

Table 3.2.5 Control account use case.

Use Case Name: Add to cart	ID: 7	Priority: High
Actor: Client		
Description: This use case describes how a client adds all products he wanted to purchase in a cart to collect them to buy them for one time.		
Trigger: The client collects products he wanted to purchase in a cart Type: ☑ External Temporal		
Precondition: 1. The client search product he wanted.		

Normal Course:	Information for steps:
1. The client adds products he wants to cart.	← Update cart
2. The client adds all the other products he needs.	← Update cart
3. Click Remove to remove selected orders from card.	← Update cart

Alternative:	
1. If the client wants to buy one product directly, he should Click "Buy". Which will make order directly.	← Orders

Postcondition:
1. When the client wants to buy all, he adds to cart, he should go to cart and but product.
Exceptions: 1. Server crash. 2. Power cut off. 3. Technical problem

Summary Inputs	Source	Outputs	Destination
• Add products to cart	• Client	• Products added to cart.	• Orders

Table 3.2.6 Add to cart use case.

Use Case Name: All payment method (Orders, Payments, Delivery)	ID: 8	Priority: High
Actor: Client		
Description: This use case describes how the client buy a product.		
Trigger: Client buys a product. Type: ☑ External Temporal		
Precondition: 1. the client will add Products to cart before buying it.		

Normal Course:	Information for steps:
1. The client adds the number of orders of product to cart to collect orders, then buy to purchase product.	← Update Orders
2. Client clicks Buy to check out product.	← Update Payments
3. System calculates the total price.	← Orders
4. Send order to payment to pay order.	← Payment
5. The payment department will complete the payments through credit account numbers.	← Payment
6. If the payment is completed the order will be send to delivery	← Delivery
7. Delivery will take off the ordered products from inventory.	← Inventory
8. Driver man will deliver the order to the client.	→ Delivery
Alternative:	
1. The system will delete orders if the money of the client is not enough to buy.	← Update Orders.
2. If it happens a cancellation message will appear to client.	← Cancellation message.

Postcondition:
1. The client will receive order details with a confirmation message when payment is completed.
2. The Seller will receive order details with confirmation message when payment is completed.
3. The client will receive Cancellation message if payment is failed.
4. The client will receive the order in the delivery dated time.

Exceptions:
1. money is not enough.
2. Server crash.
3. Power cut off.
4. Technical issues.
5. delivery issues.

Summary Inputs	Source	Outputs	Destination
• order	• Client	• Order details • Delivery of order	• Client

Table 3.2.7 All payment method (Orders, Payments, Delivery) use case.

Use Case Name: Enter rating of product	ID: 9	Priority: Medium
Actor: Client		
Description: This use case describes how the client enter rating of seller and product.		
Trigger: Enter seller and product rating. Type: ☑External Temporal		
Precondition: 1. The client buys the product. 2. The delivery has arrived.		

Normal Course:	Information for steps:
1. Click number of stars of product rating.	← update product table
2. Add review comment about product.	← update product table
Alternative: 1. The client doesn't want to enter the rating. 2. He will finish the payment and close the site.	

Postcondition:
1. The Client finish payment and receive order details. 2. He may contact the seller. 3. He closes the system.
Exceptions: 1. Server crash. 2. Power cut off.

Summary Inputs	Source	Outputs	Destination
• Product rating • Comment on product.	• Client	• Product stars • Comment on product.	• Product in Inventory

Table 3.2.8 Enter rating of product use case.

3.2.3 Entity-relationship diagram (ERD):

The ERD for the furniture online store consists of three main entities: seller, management, and clients. The ERD includes relationships between these entities, such as the relationship between the client entity and the order entity, indicating that clients can place orders. The ERD also includes attributes for each entity, such as the product name, price, and quantity for the product entity. ERD helps designers and developers to understand the data structure of a database, identify potential problems, and optimize the database design.

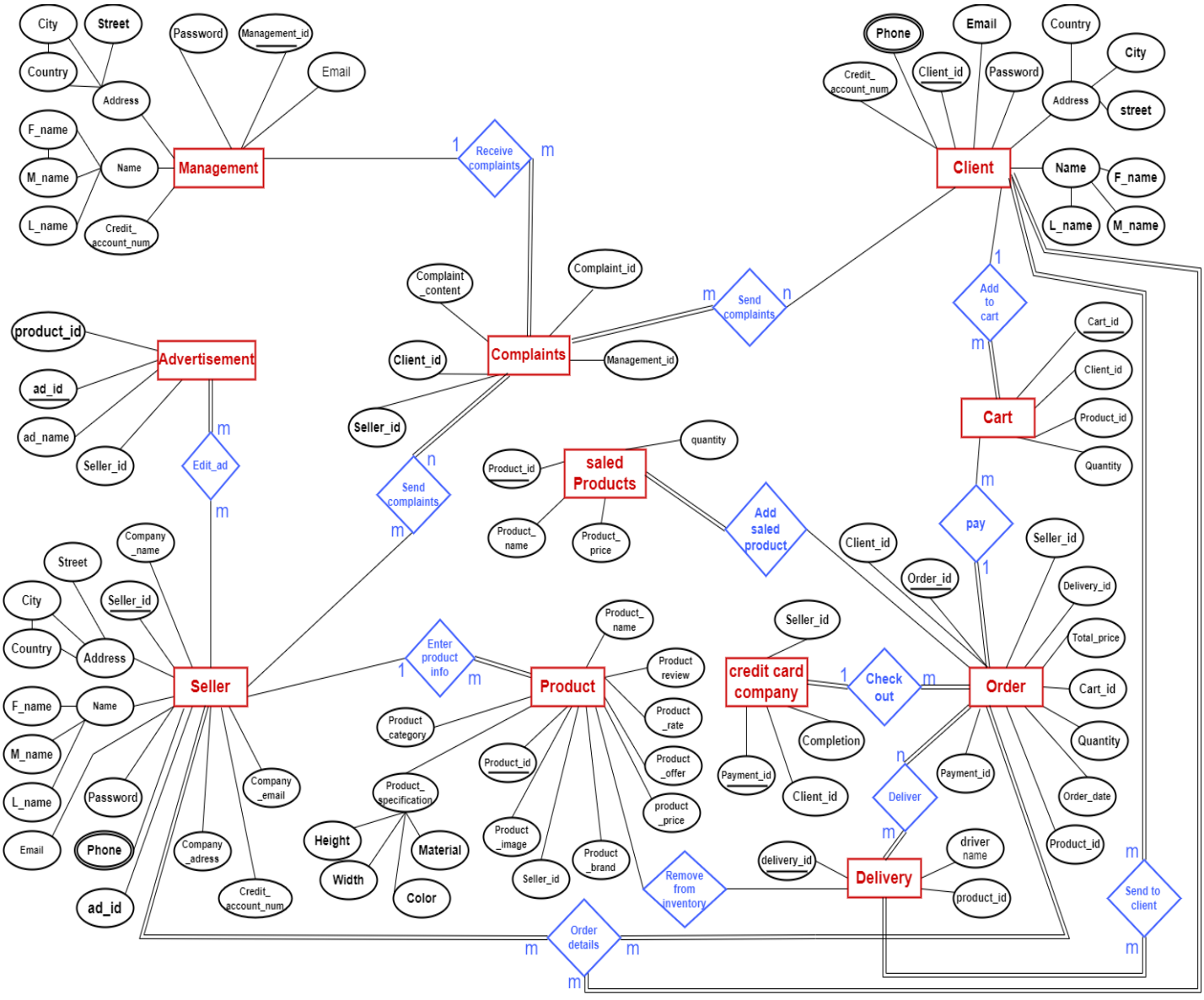


Figure 3.2.4 Entity-relationship diagram of the system.

3.2.4 Structure Charts:

Structure Chart illustrates the hierarchy and relationships between program modules or functions in a software system. It is used to represent the organization of modules in a structured programming or software engineering environment. The modules are represented as rectangles or boxes, with arrows indicating the data flow and control flow between them. It helps in visualizing the structure of a software system, identifying reusable modules, and optimizing the design of the system. The following sequence of charts illustrates structure charts of the important functionalities of the system.

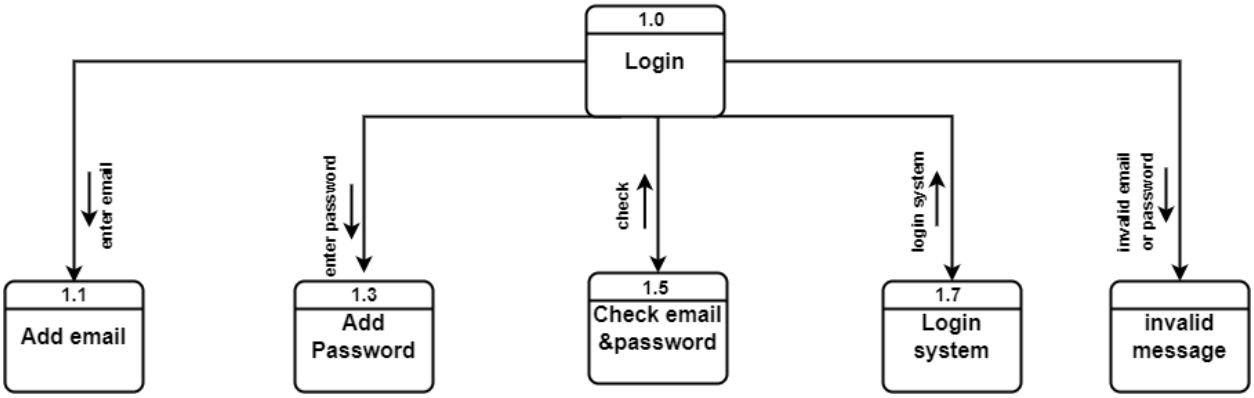


Figure 3.2.5 Login structure chart.

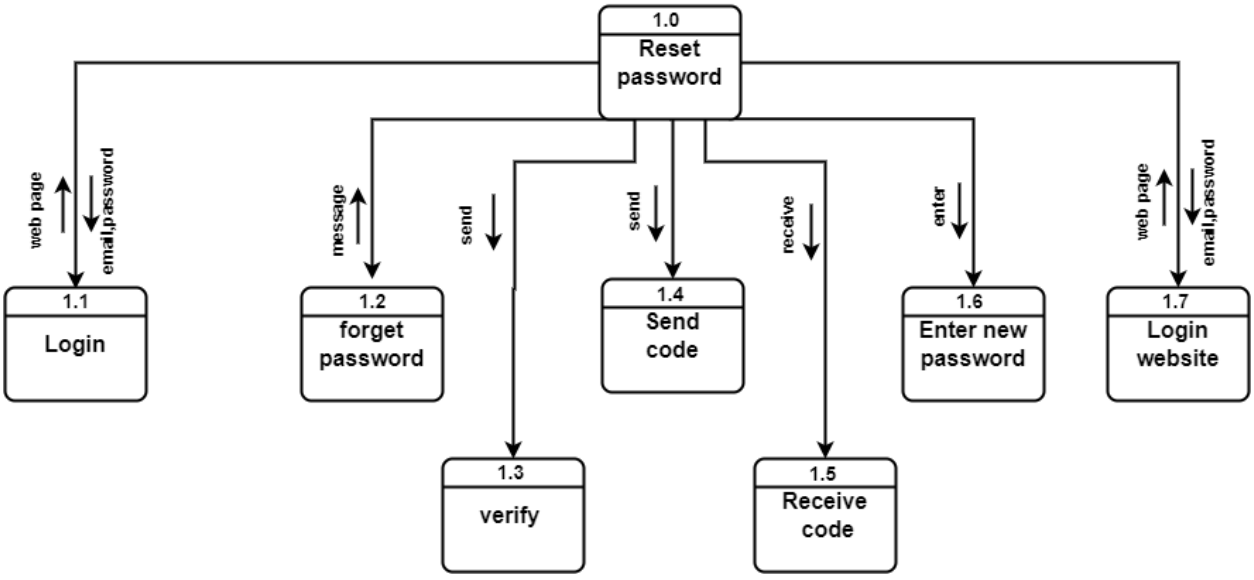


Figure 3.2.6 Reset password structure chart.

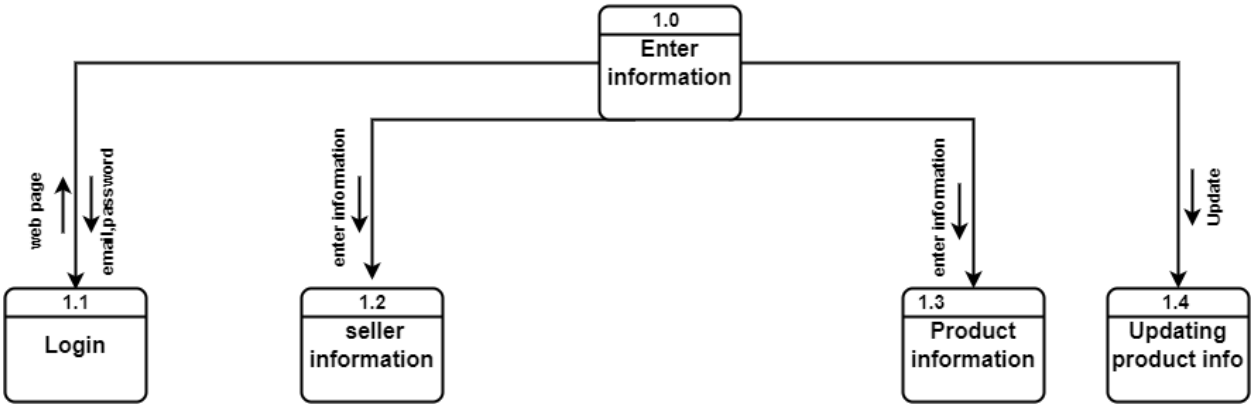


Figure 3.2.7 Enter information structure chart.

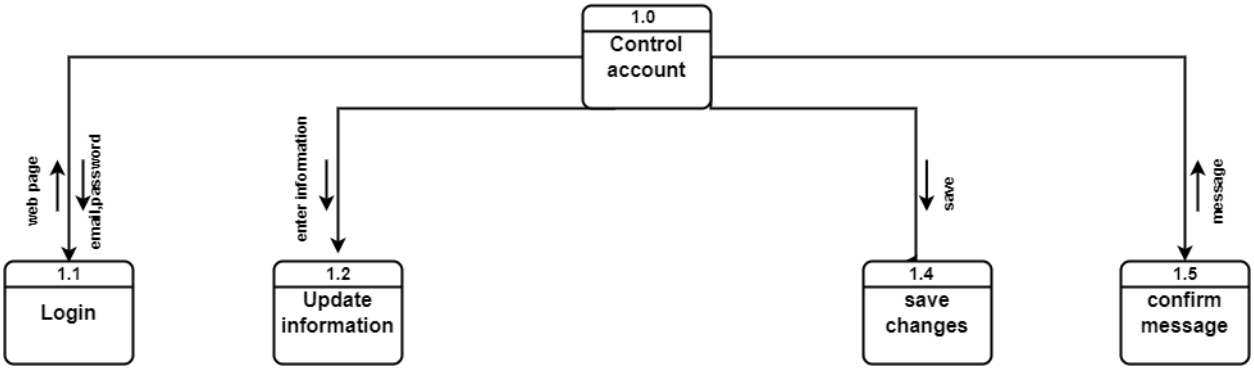


Figure 3.2.8 Control account structure chart.

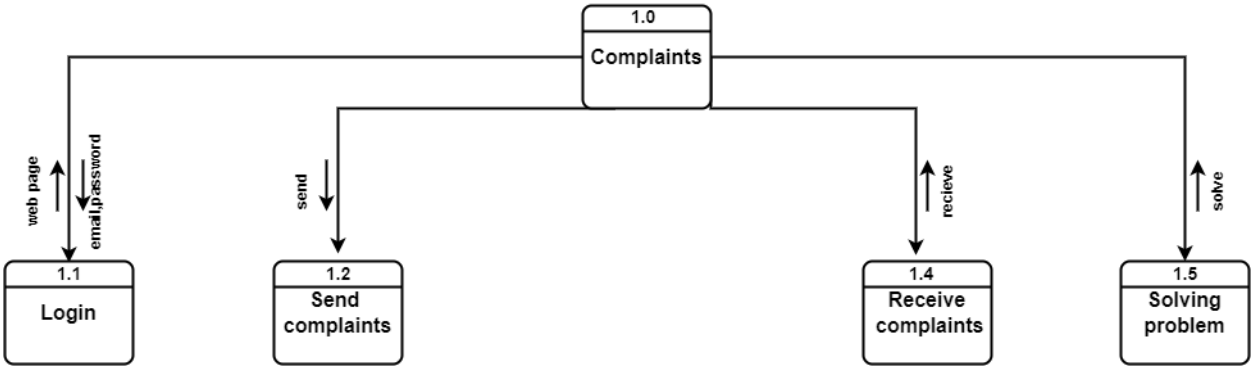


Figure 3.2.9 Complaints structure chart.

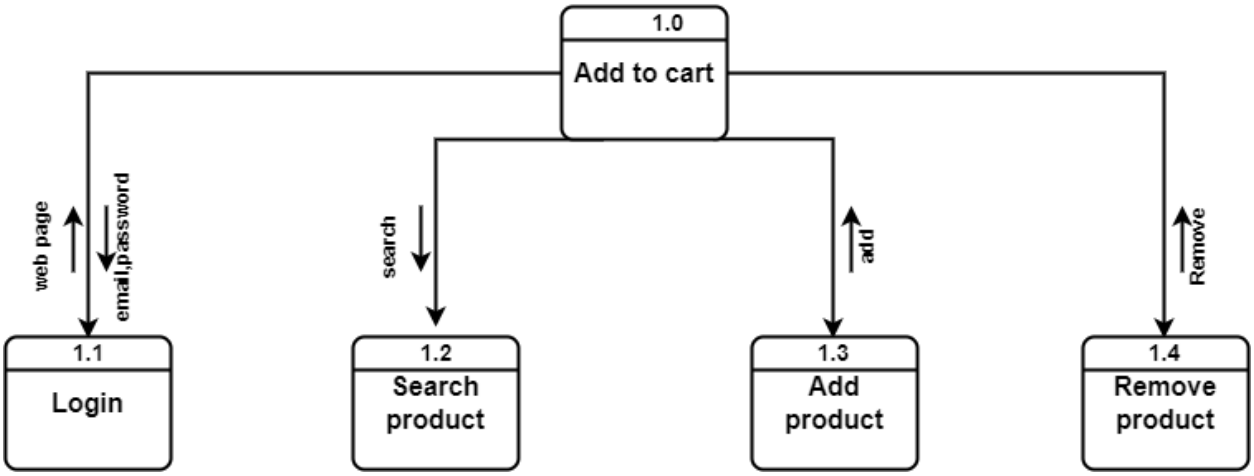


Figure 3.2.10 Add to cart structure chart.

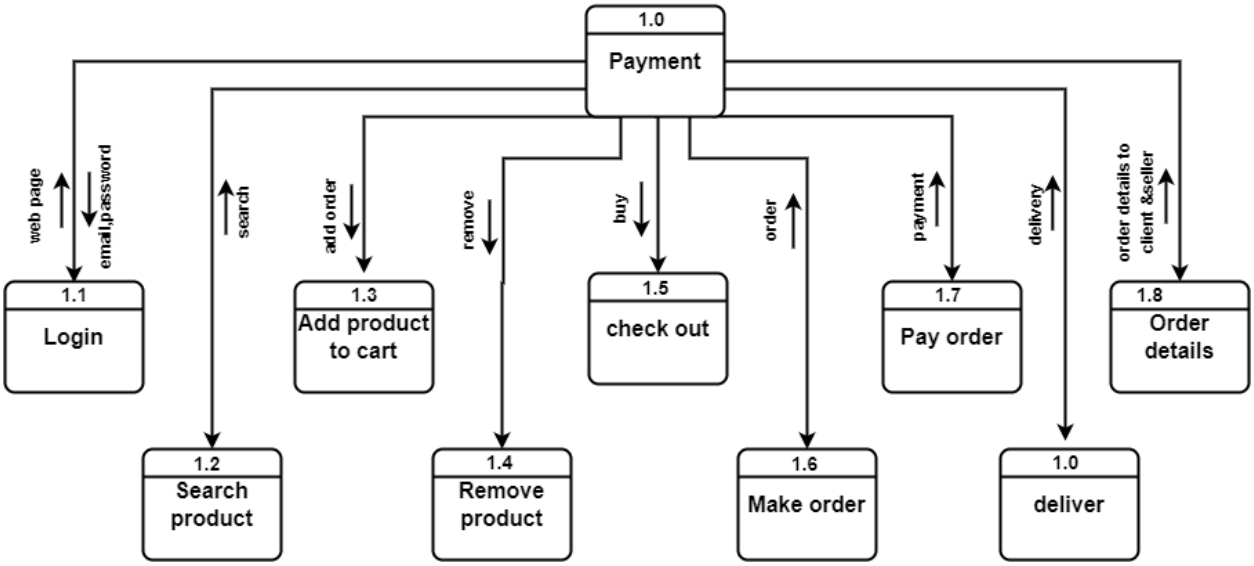


Figure 3.2.11 Payment structure chart.

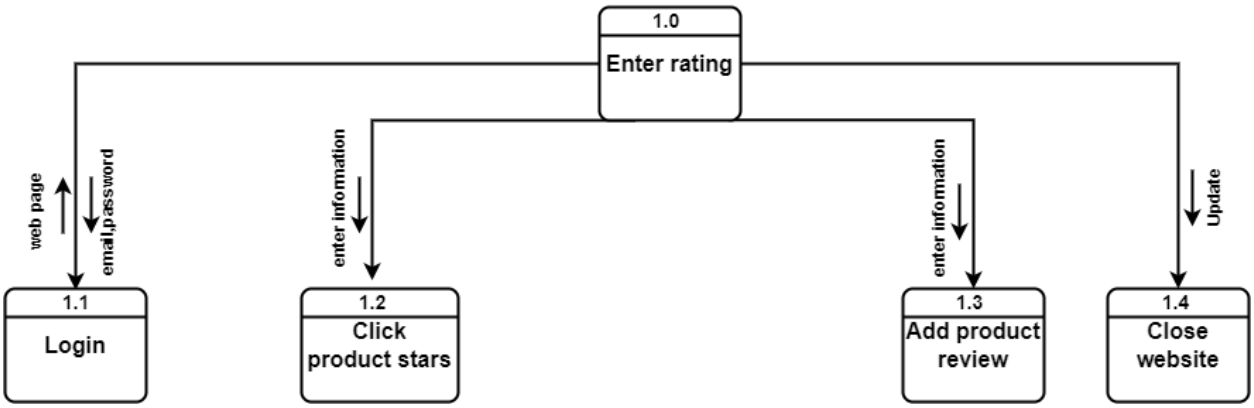


Figure 3.2.12 Enter rating structure chart.

3.3. System tools and technologies:

Our project encompasses both a website and mobile application, and to create an effective and engaging user experience, we have utilized a range of tools and technologies across frontend, backend, and mobile app development.

3.3.1. System analysis:

We have performed a thorough system analysis using diagrams.net to identify and address potential issues and optimize the performance of our website and mobile app and to create system architecture diagrams and visualize the components and interactions of our system.

3.3.2. User Interface and User Experience (UI/UX):

To design the user interface and user experience, we have used Adobe XD, Adobe Photoshop, and Adobe Illustrator. These tools have allowed us to create wireframes, mockups, and prototypes for testing and refining the design before implementation.

3.3.3. Frontend Development:

For frontend development, we have used HTML, CSS, and JavaScript to create the visual and interactive elements of our website. Additionally, we have leveraged the Bootstrap framework and Swiper.js library to enhance the styling and functionality of our interfaces. The React.js library has also been used to build complex user interfaces.

3.3.4. Mobile App Development:

To develop our mobile app, we have utilized Flutter, an open-source mobile application development framework created by Google. Flutter allows us to build native-looking interfaces for both iOS and Android platforms. We have also used the Material and Cupertino design systems to create a consistent and seamless experience across the two platforms.

3.3.5. Backend Development:

For server-side scripting, we have used PHP and the Laravel framework to manage data and perform server-side processing. MySQL has been used for database management.

CHAPTER 4

System Design

4. System Design:

The system design for the furniture online store consists of three main parts: the seller interface, the client interface, and the management interface. The seller interface is designed to allow sellers to enter their personal information, add product details, and manage their inventory. The client interface is designed to allow clients to search for products, add them to their cart, and place orders. The management interface is designed to allow the system administrators to manage the entire system, set constraints, and handle the shipping process. The system design also includes a database that stores information about sellers, clients, products, orders, and other important data. The database is designed to be scalable and efficient, allowing for easy management and retrieval of information. The system design also includes the interface design refers to the process of designing the user interface of a software application or system. It involves creating the visual and interactive elements of the user interface. The overall system design is user-friendly, intuitive, and efficient, providing a seamless experience for sellers, clients, and system administrators.

4.1. Physical ERD:

A Physical ERD is a diagram that shows how a database is structured at a physical level, including tables, columns, and data types. It is used to implement and build a database and to ensure that it meets the users' requirements. It can also include indexes, keys, and other database objects to optimize the performance of the database.

Figure 4.1.1 illustrates the physical ERD of the Whole system.

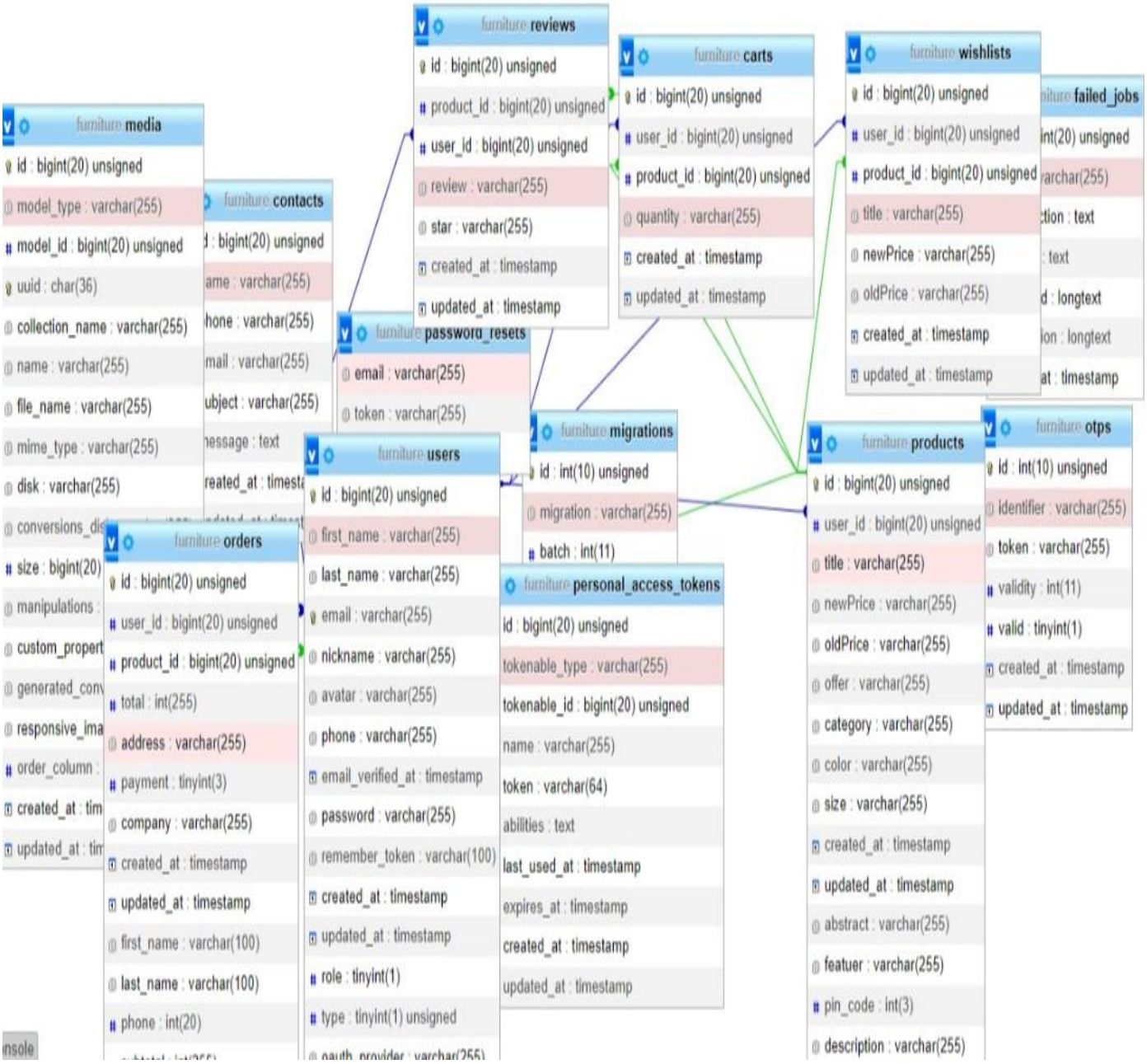


Figure 4.1.1 Physical ERD.

4.2. System LOGO:

Figure 4.2.1 System LOGO.

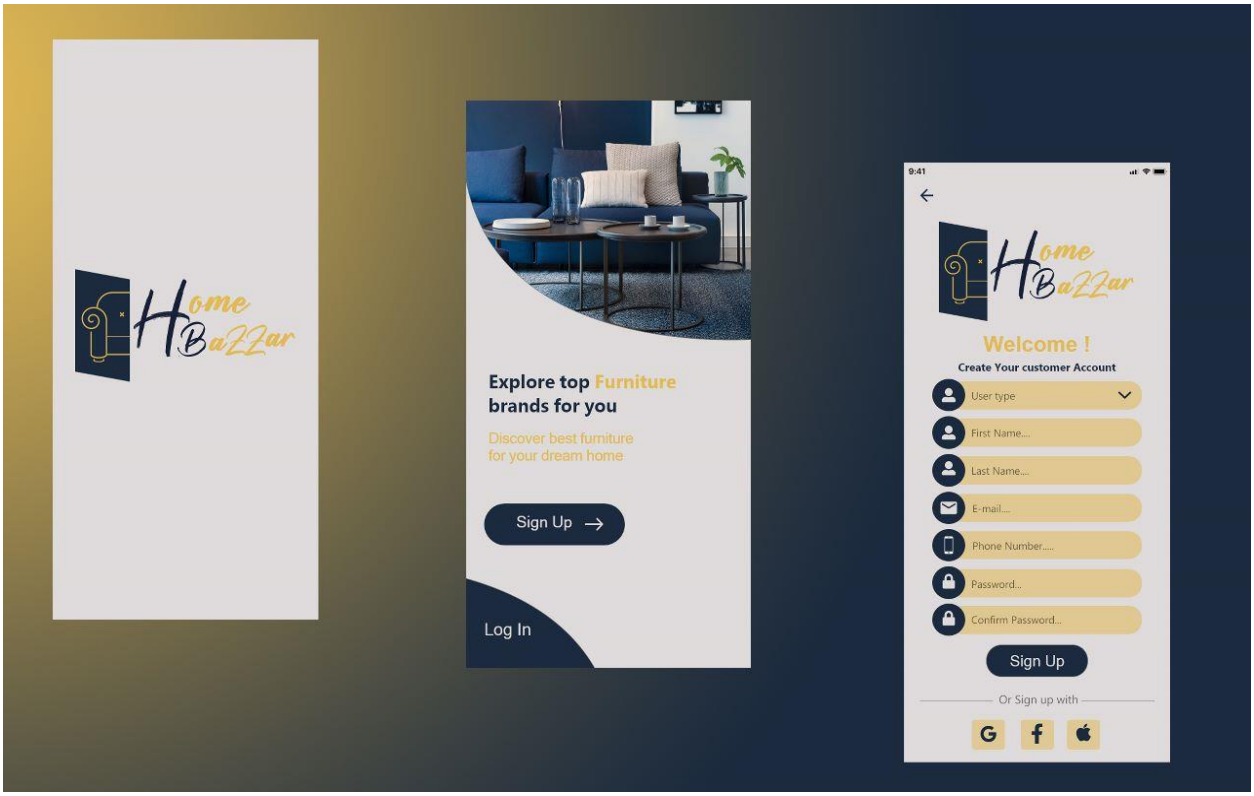
4.3. UIUX of the mobile application:

The UI/UX (User Interface/User Experience) of a mobile app refers to how the app looks and feels, as well as how easy it is to use and navigate. It includes the design of the visual elements of the app, such as the layout, typography, color scheme, and images, as well as the interaction design, such as the navigation, gestures, and feedback. A good UI/UX design aims to provide a seamless and enjoyable experience for the user, by anticipating their needs, simplifying tasks, and providing clear and helpful guidance. It should be intuitive, responsive, and accessible, and take into consideration the limitations and capabilities of mobile devices.

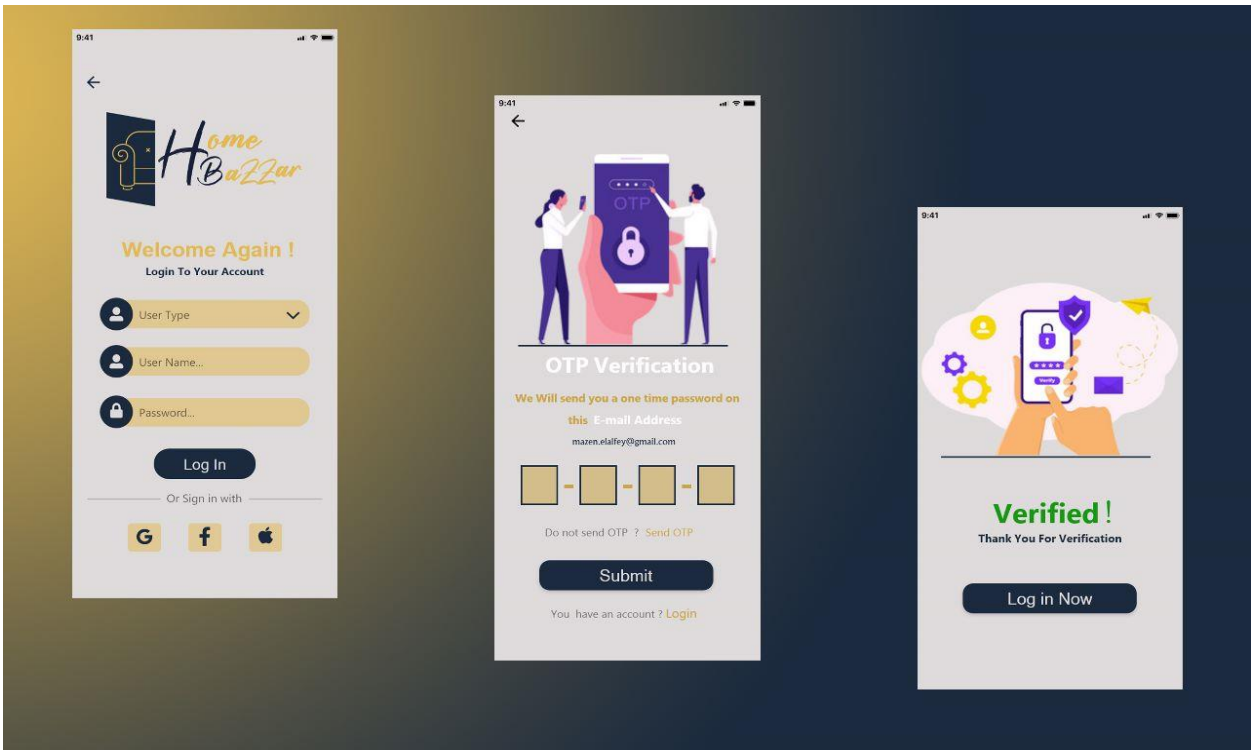
The following sequence of Figures illustrates the UIUX design of the mobile application.

Figures 4.3.1 UIUX of the mobile application.

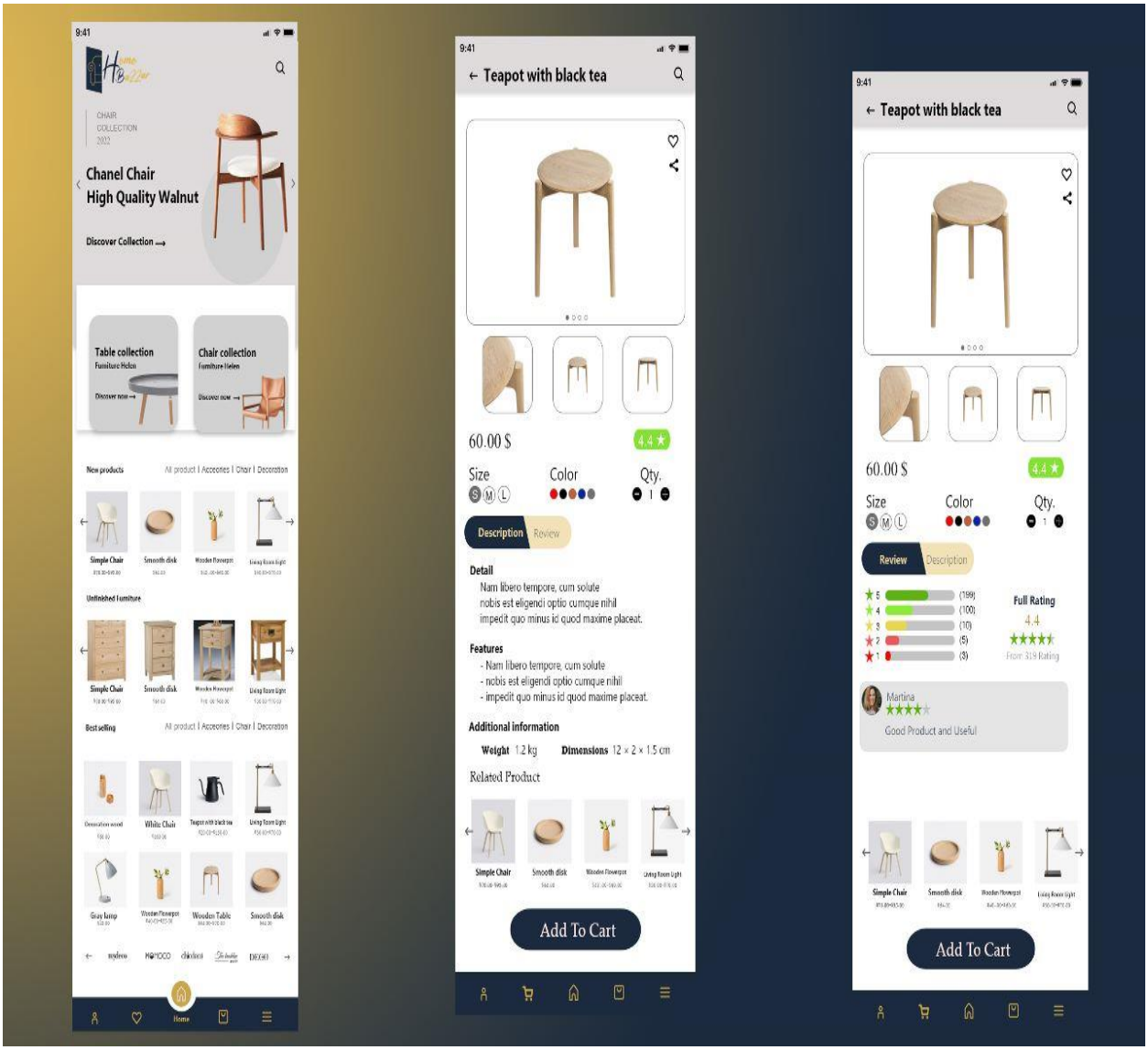
Sign Up:



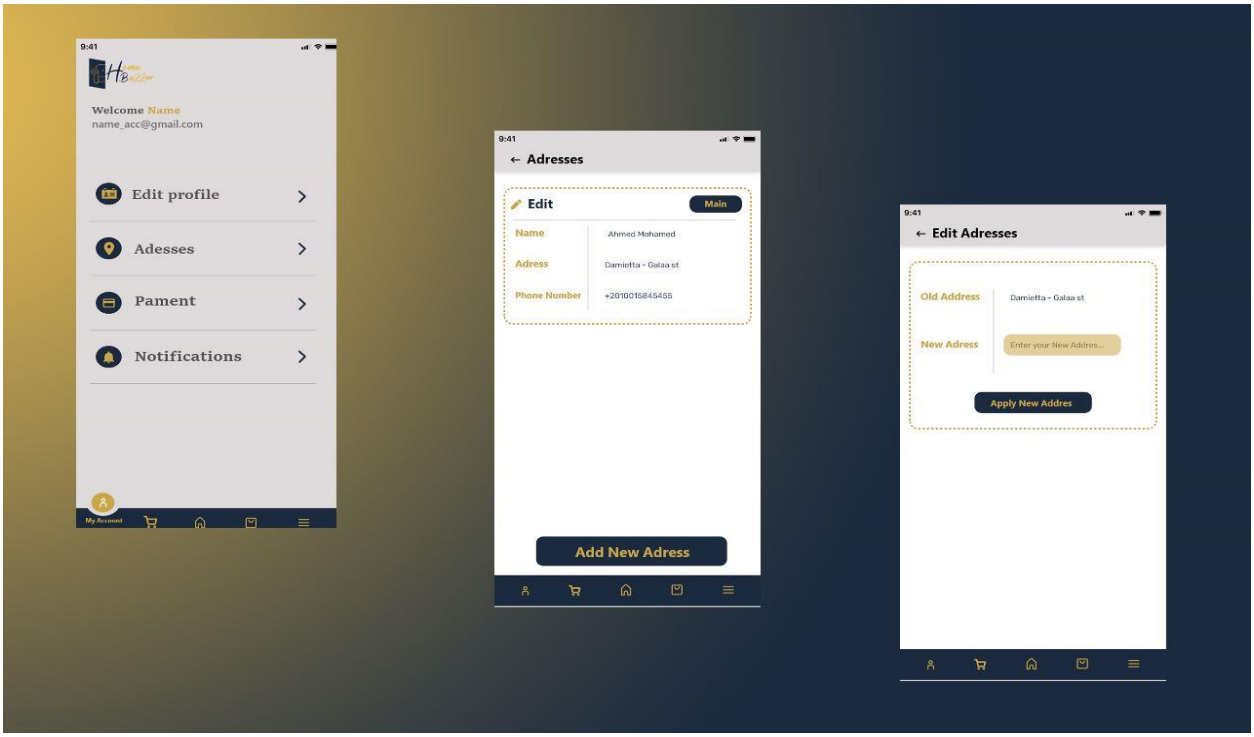
Account:



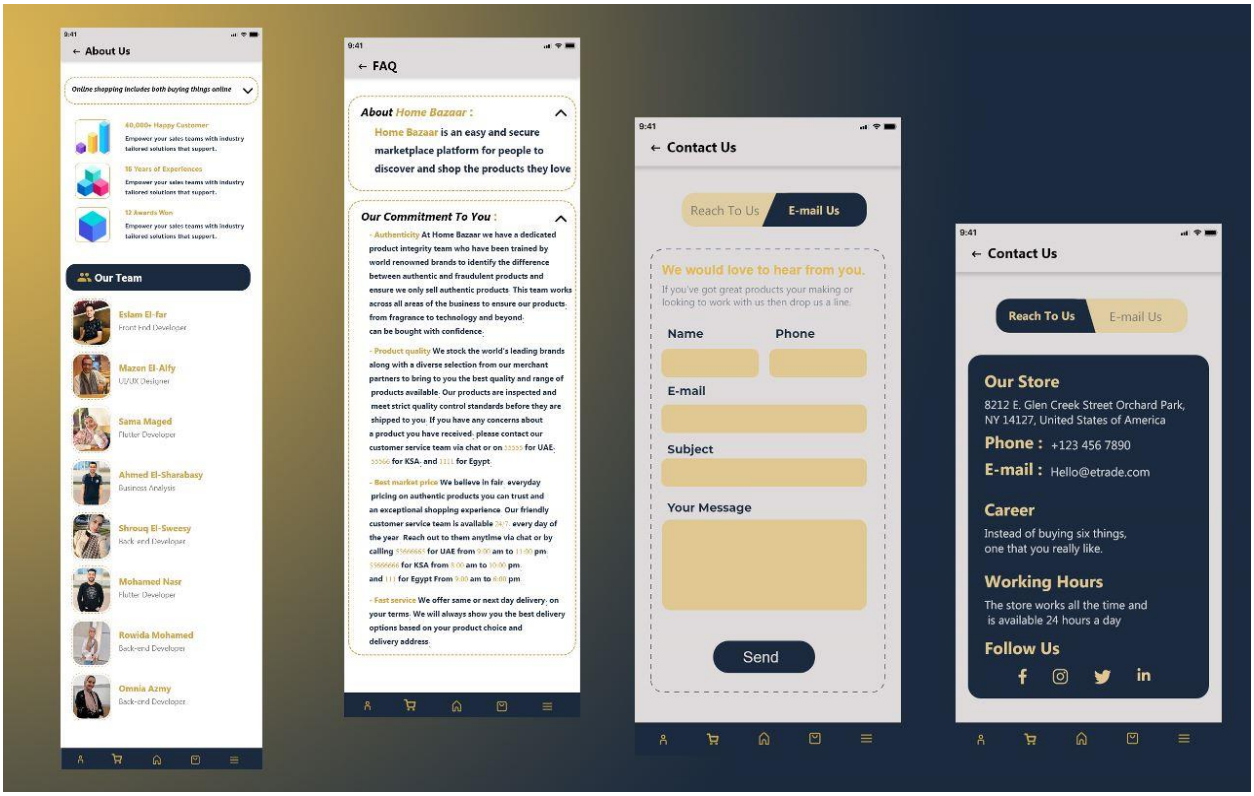
Home page:


Chanel Chair
High Quality Walnut
Discover Collection
Table collection
Chair collection
New products
Best selling
Unfinished furniture
Teapot with black tea
60.00 $
4.4
Size
Color
Qty.
Description
Review
Detail
Features
Additional information
Weight 1.2 kg
Dimensions 12 × 2 × 1.5 cm
Related Product
Add To Cart
Full Rating
Good Product and Useful

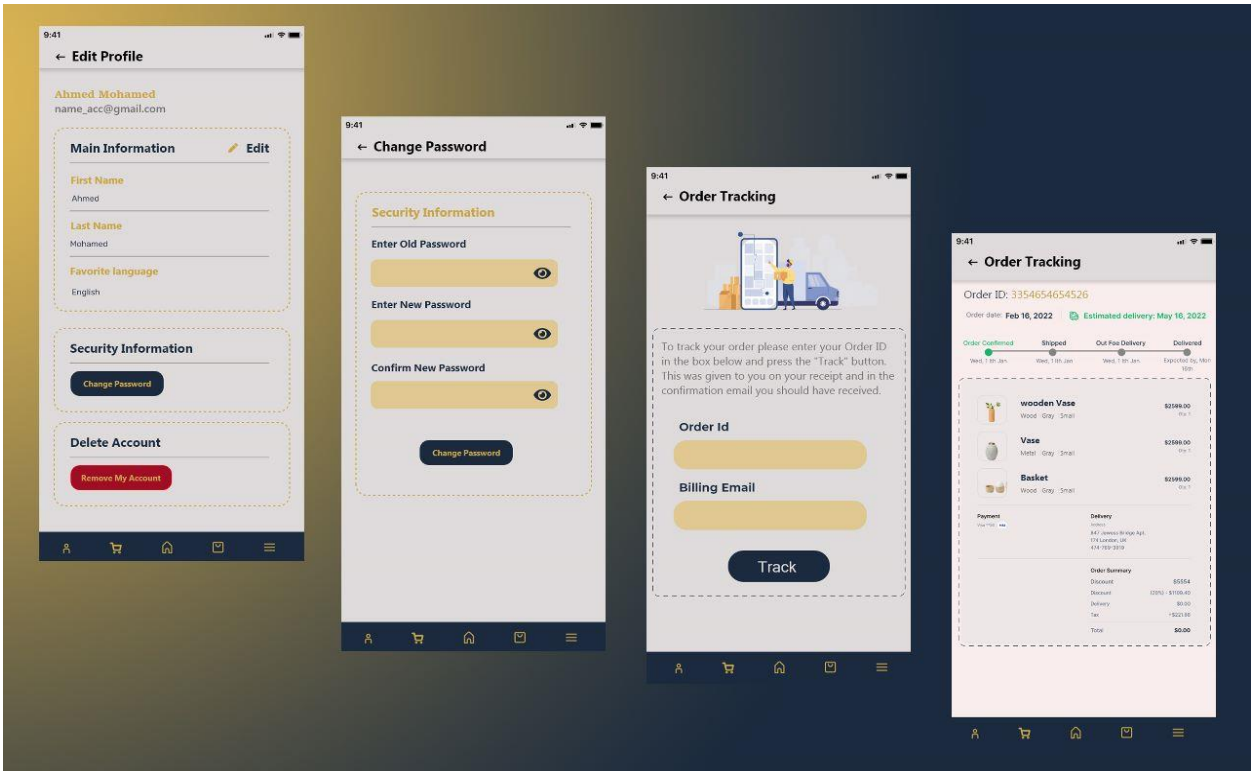
Account Address:



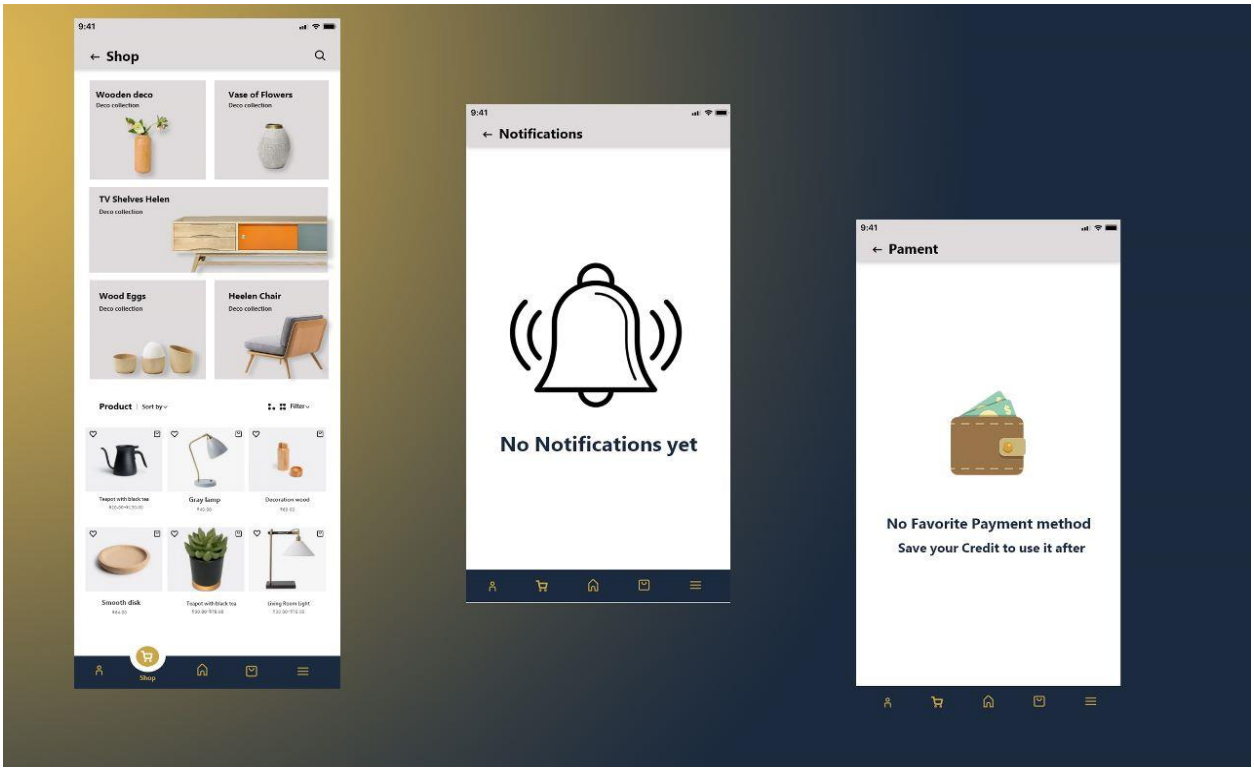
About Us / FAQ / Contact Us:



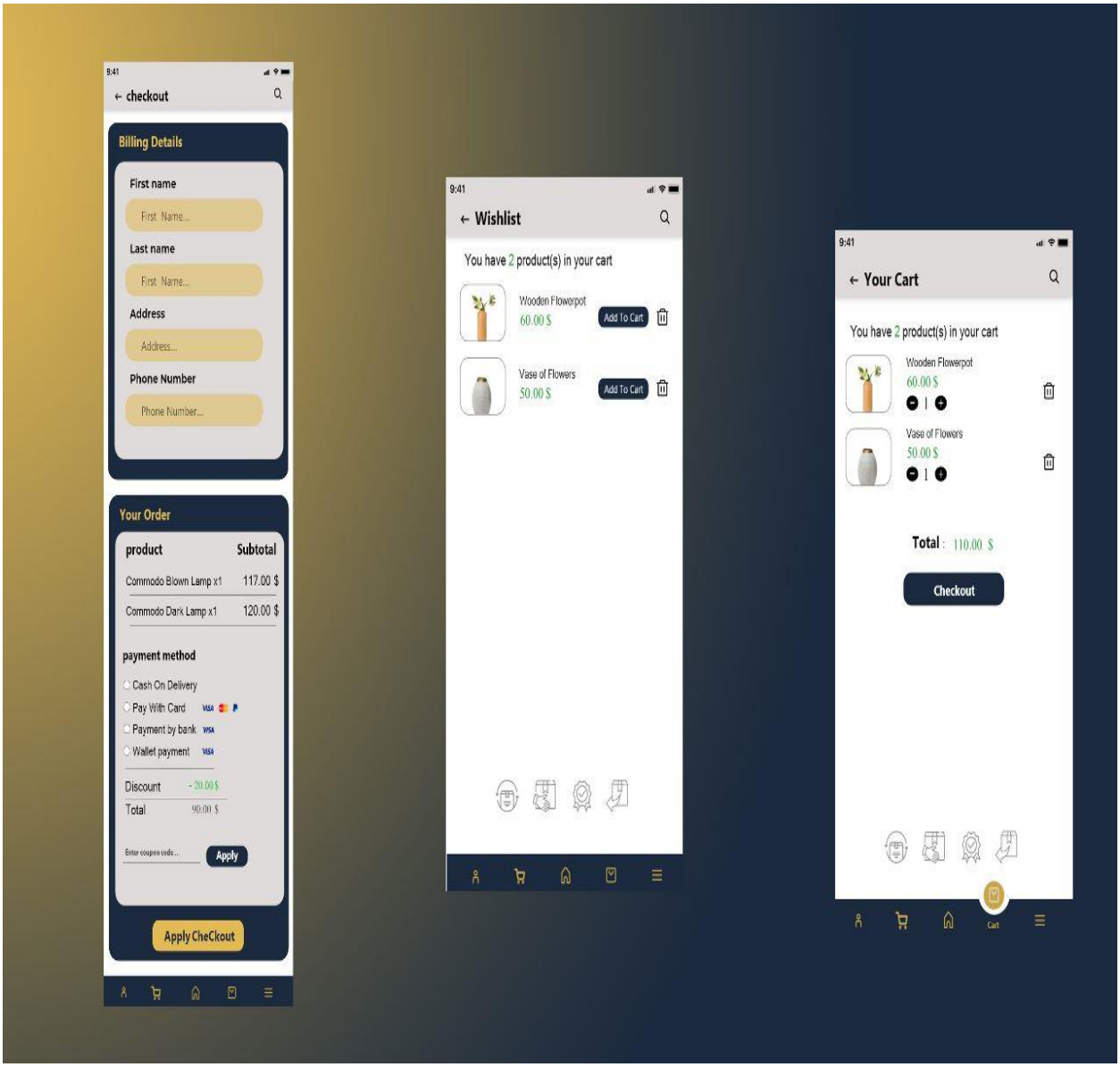
Edit Profile / Password / Order Tracking:



Shop / Notification / Payment:



Checkout / wishlist / Your Cart:



CHAPTER 5

System Implementation

5. System Implementation:

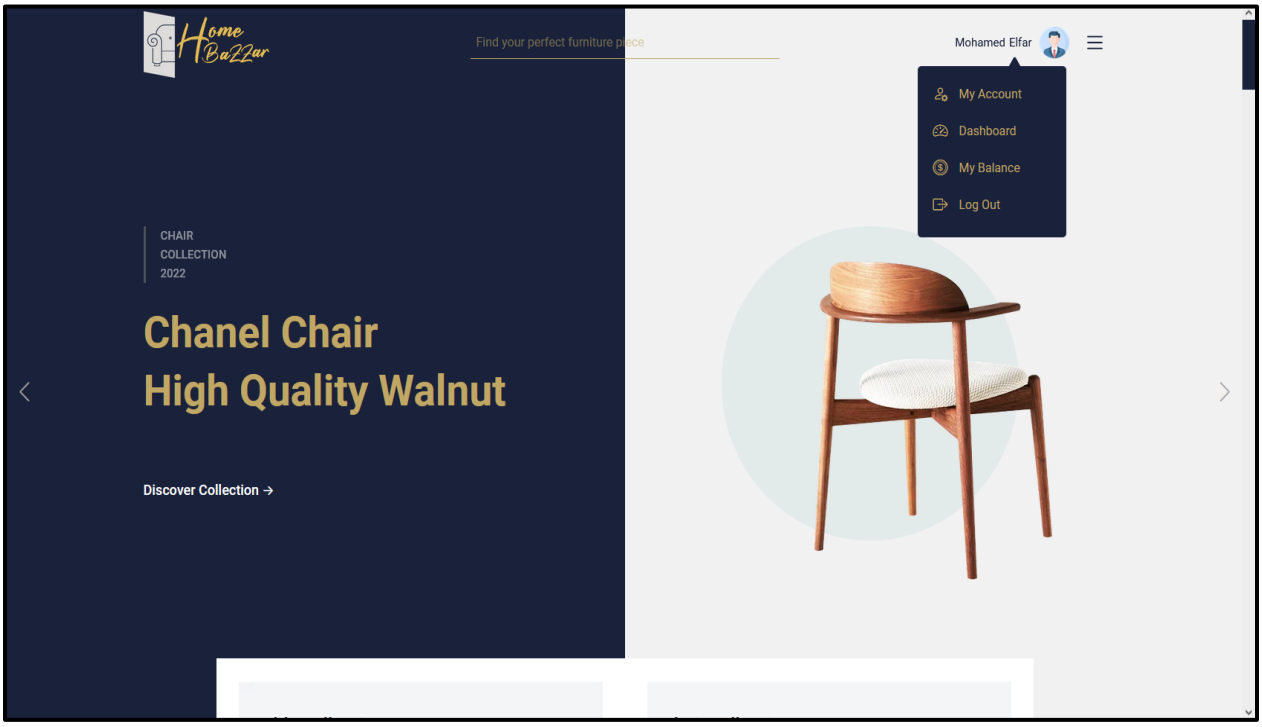
In system implementation we turned the system design into a functional system. For our Furniture Online Store project, this involves developing both the website and mobile application. The implementation phase includes coding, testing, debugging, and integrating the various components of the system. The interface design of the website and application is also an essential part of the implementation phase, ensuring that the visual and interactive elements of the user interface are designed to be user-friendly, intuitive, and aesthetically pleasing.

5.1. Interface Design of the website:

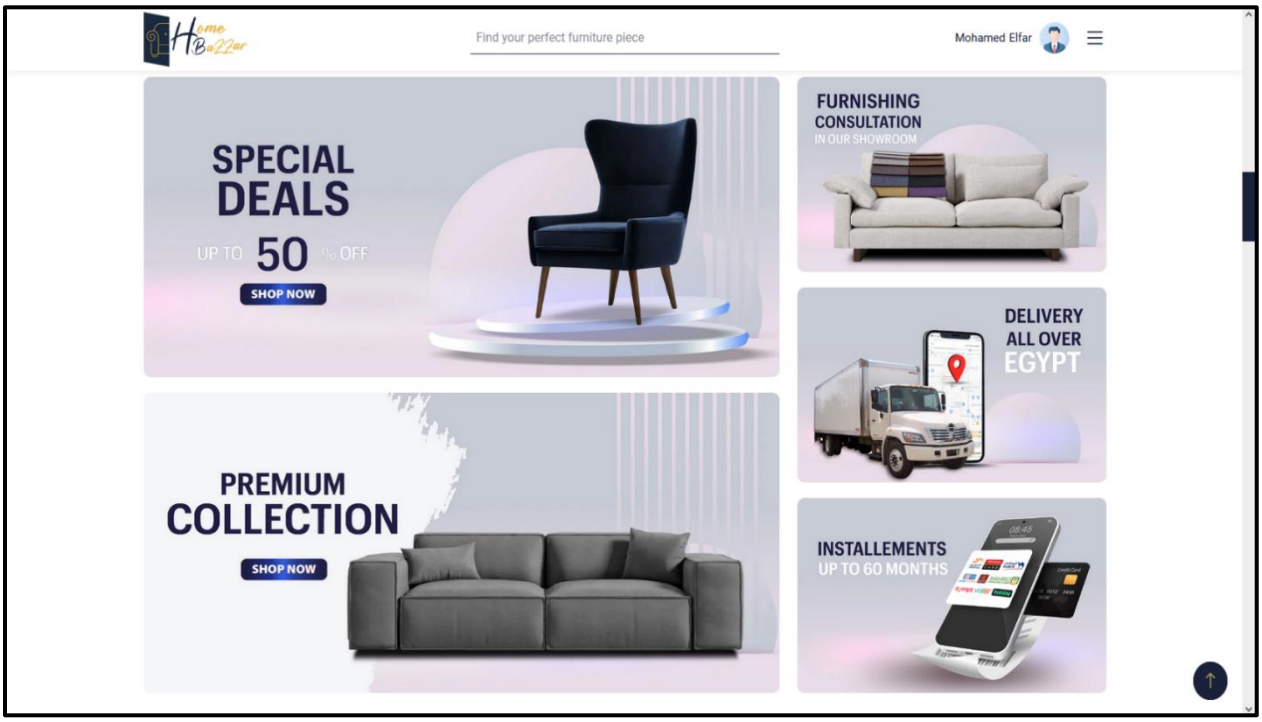
The following sequence of Figures illustrates the interface design.

Figures 4.4.1 Interface Design of the website.

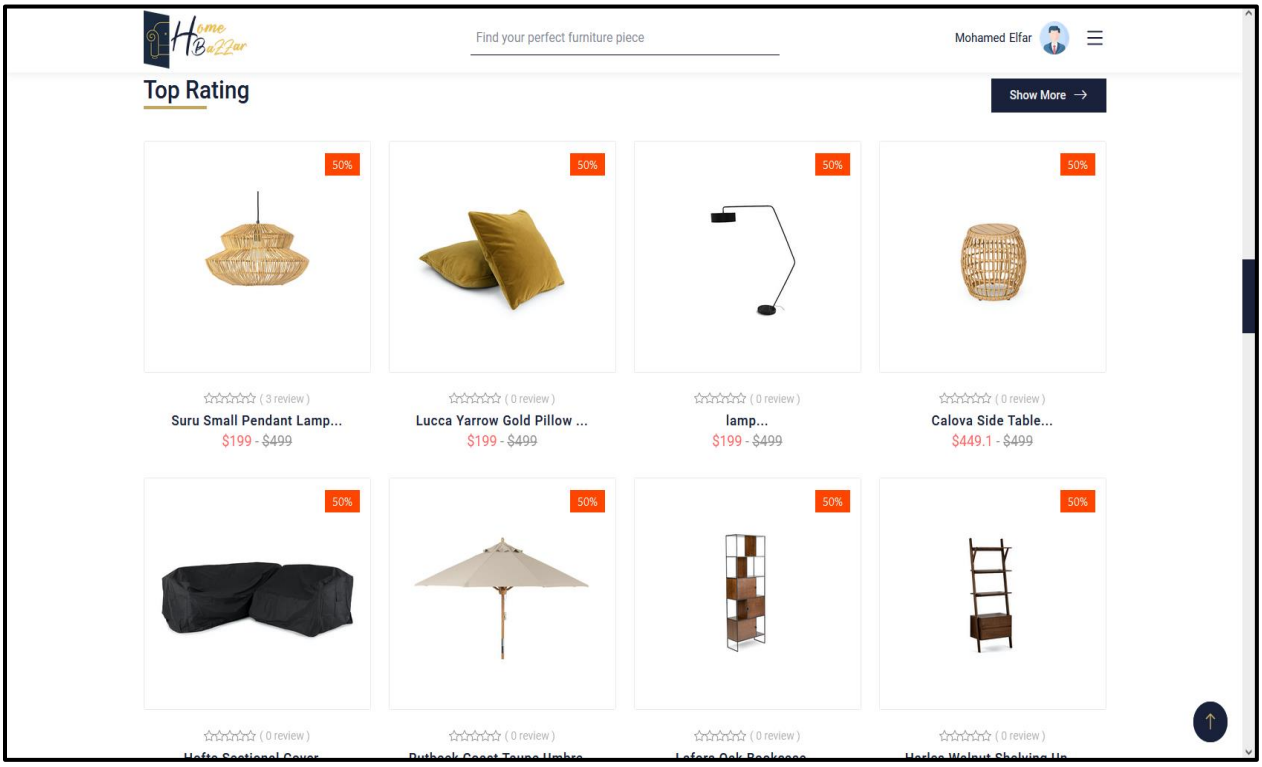
Home page:



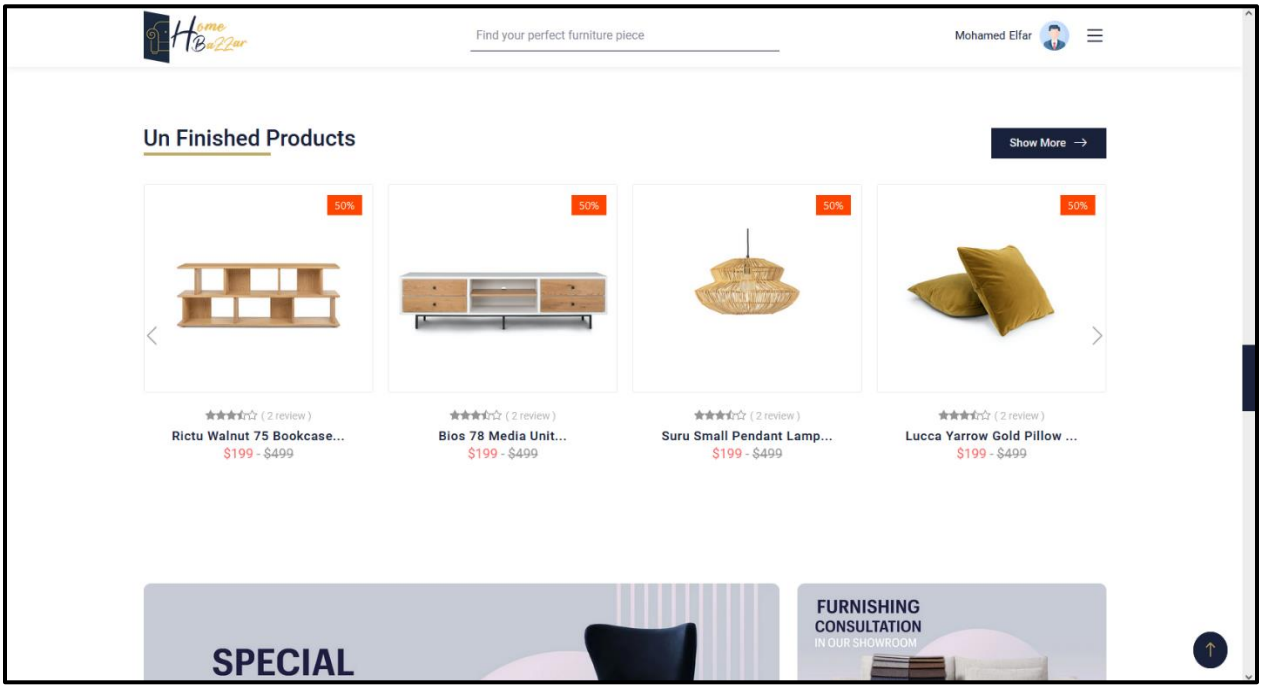
Offers:



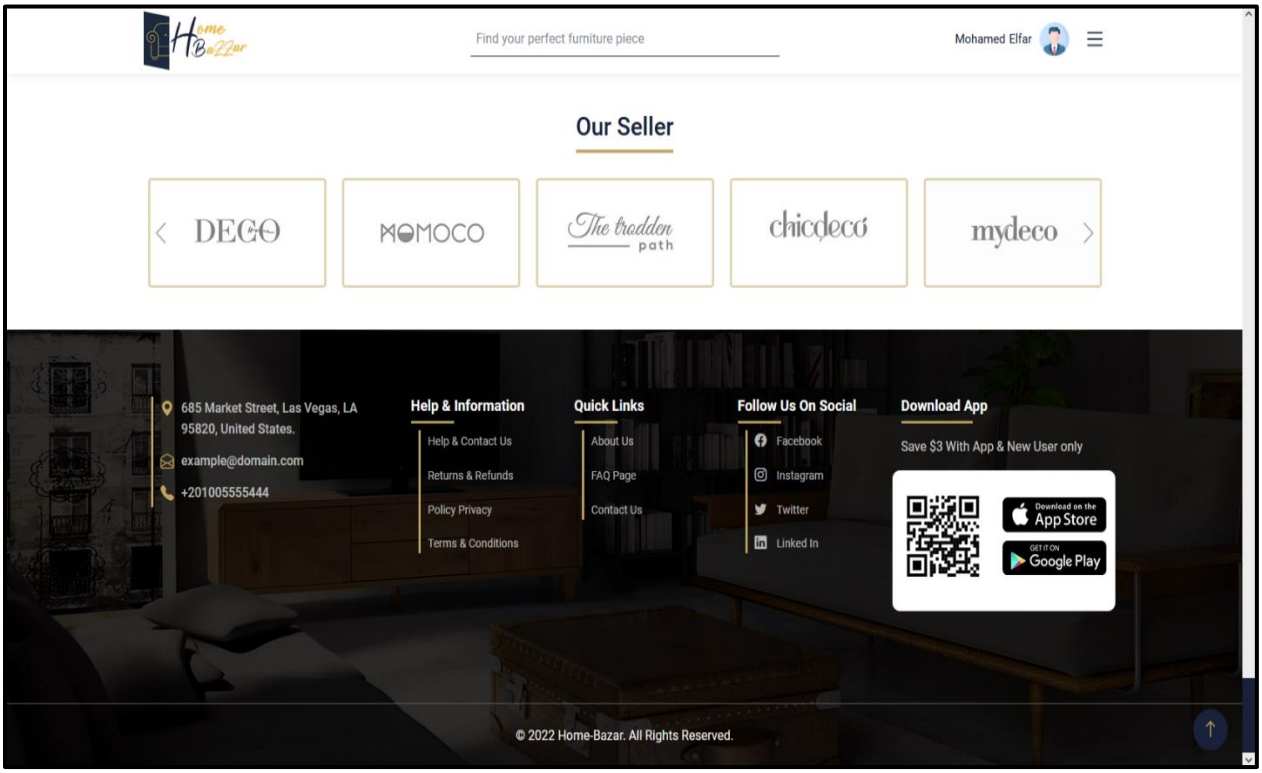
Top Rating:



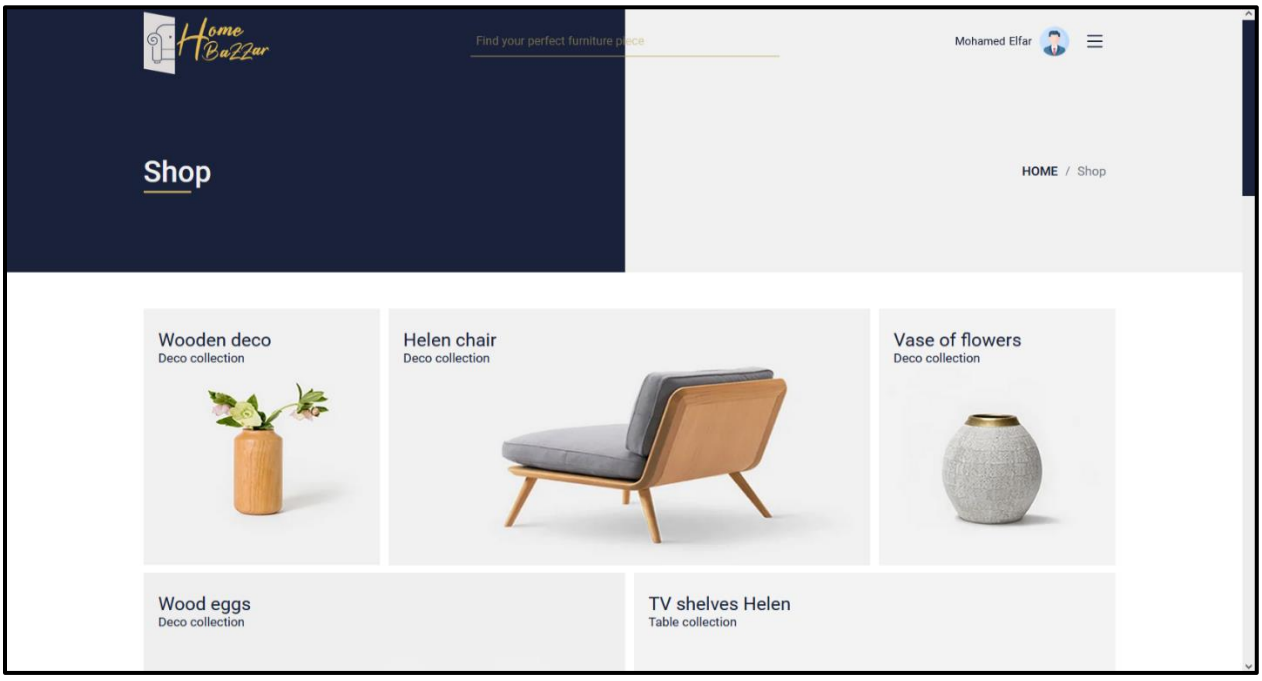
Unfinished Products:



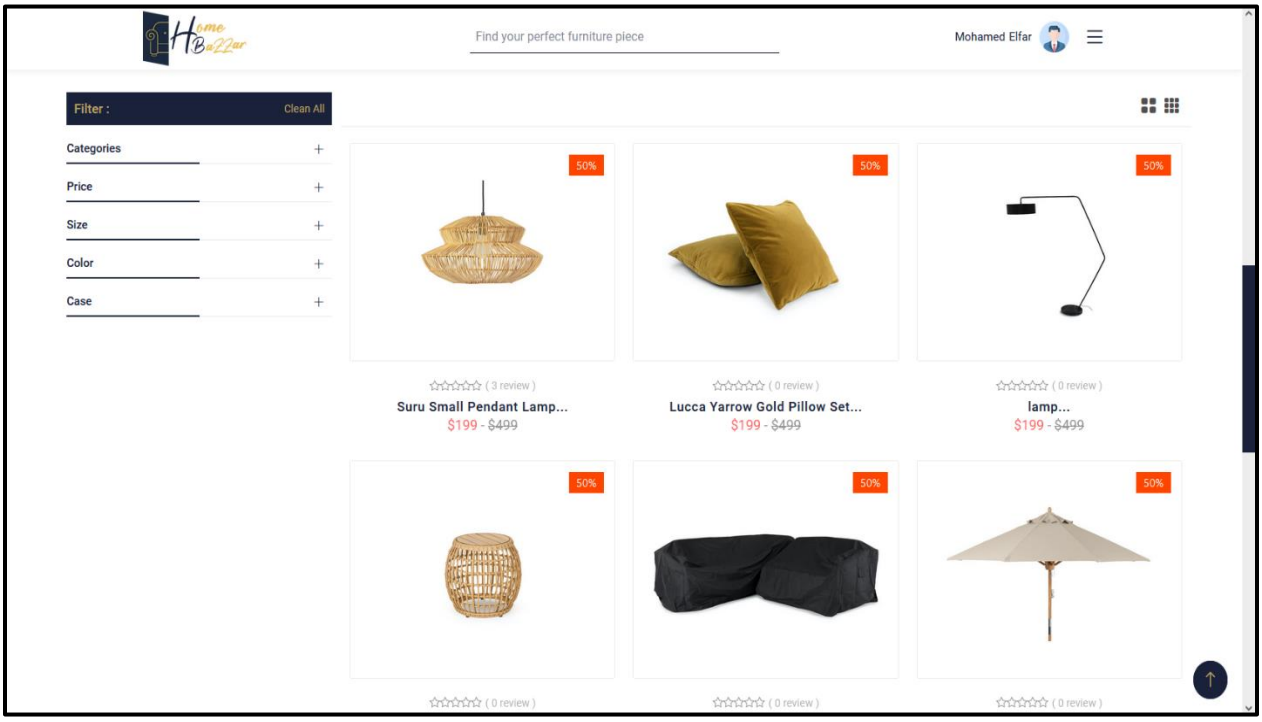
About Us:



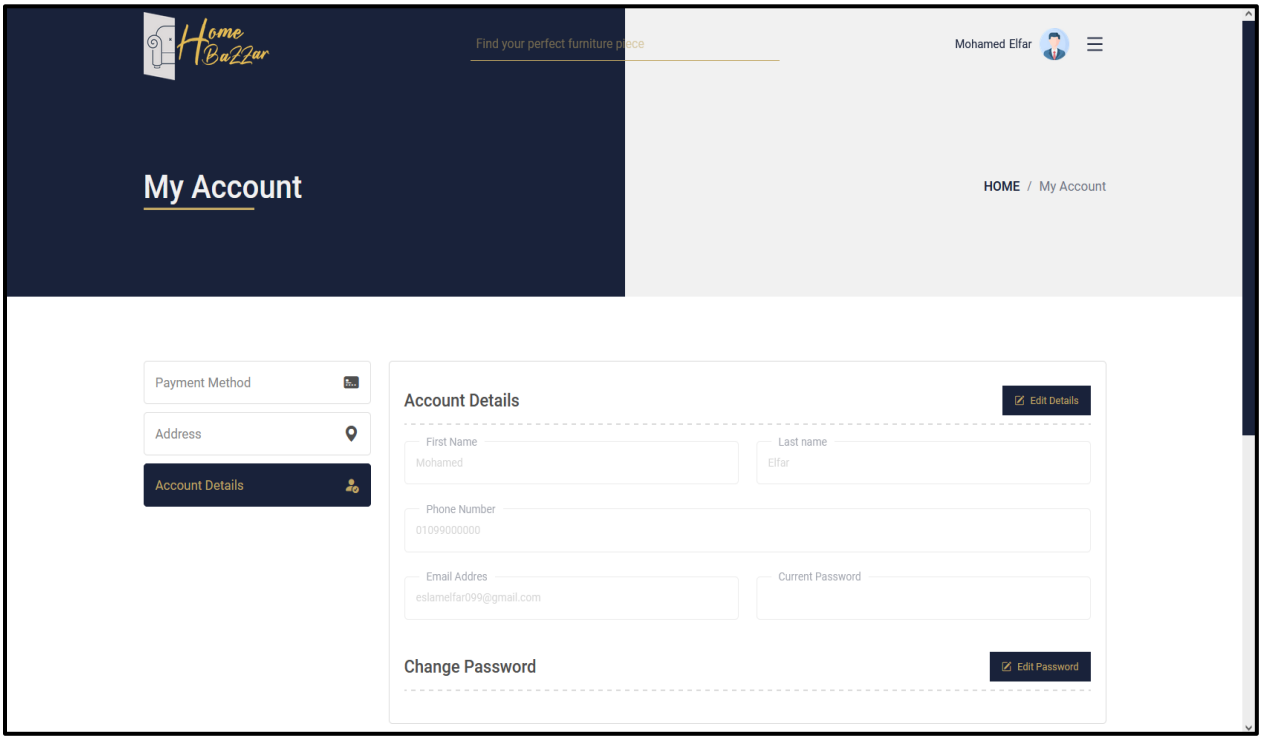
Shop:



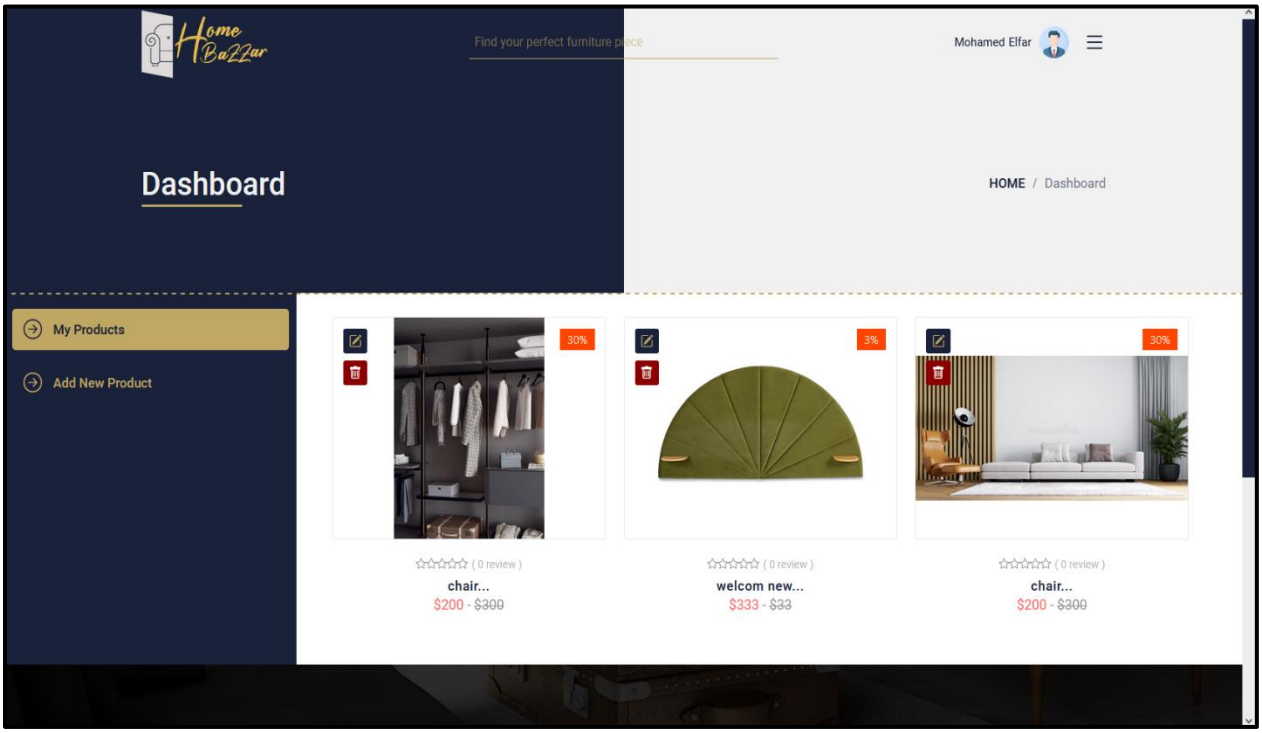
Filter:

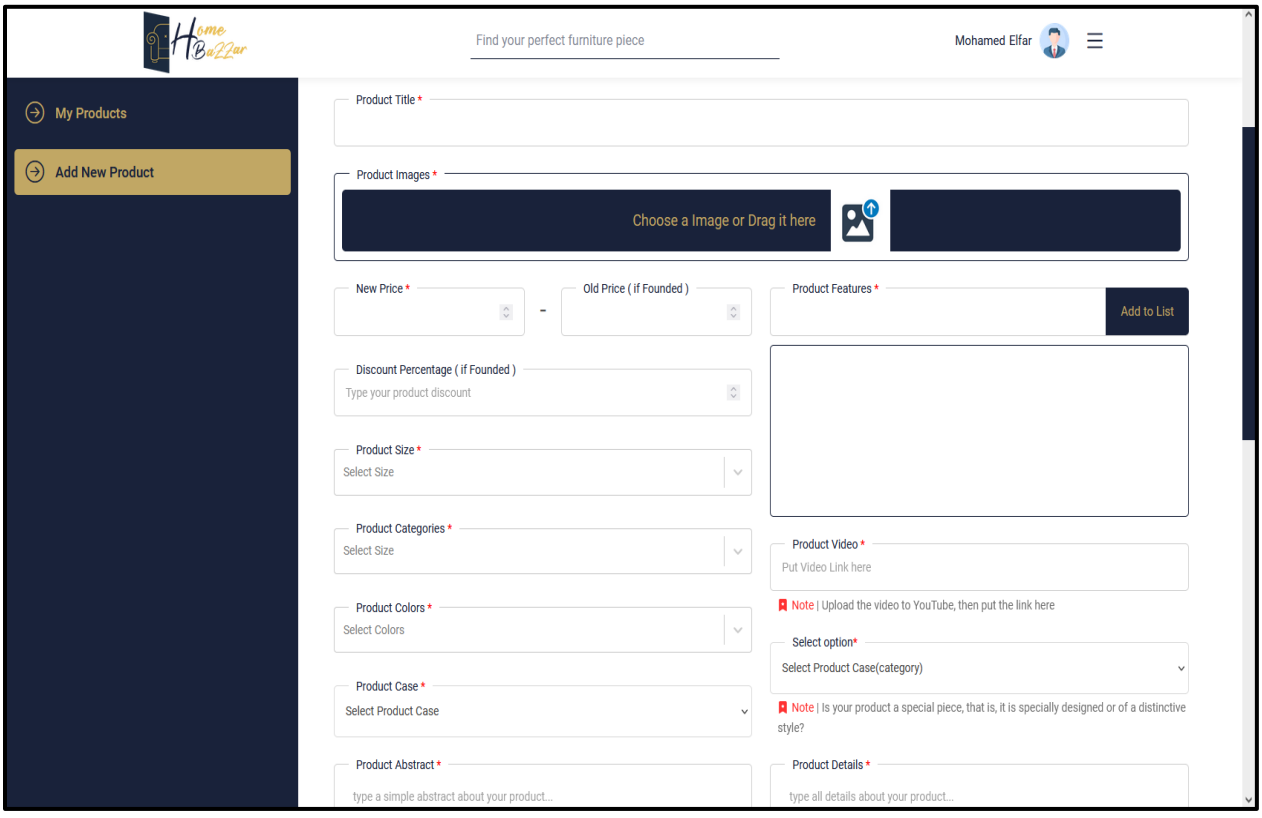


My Account:

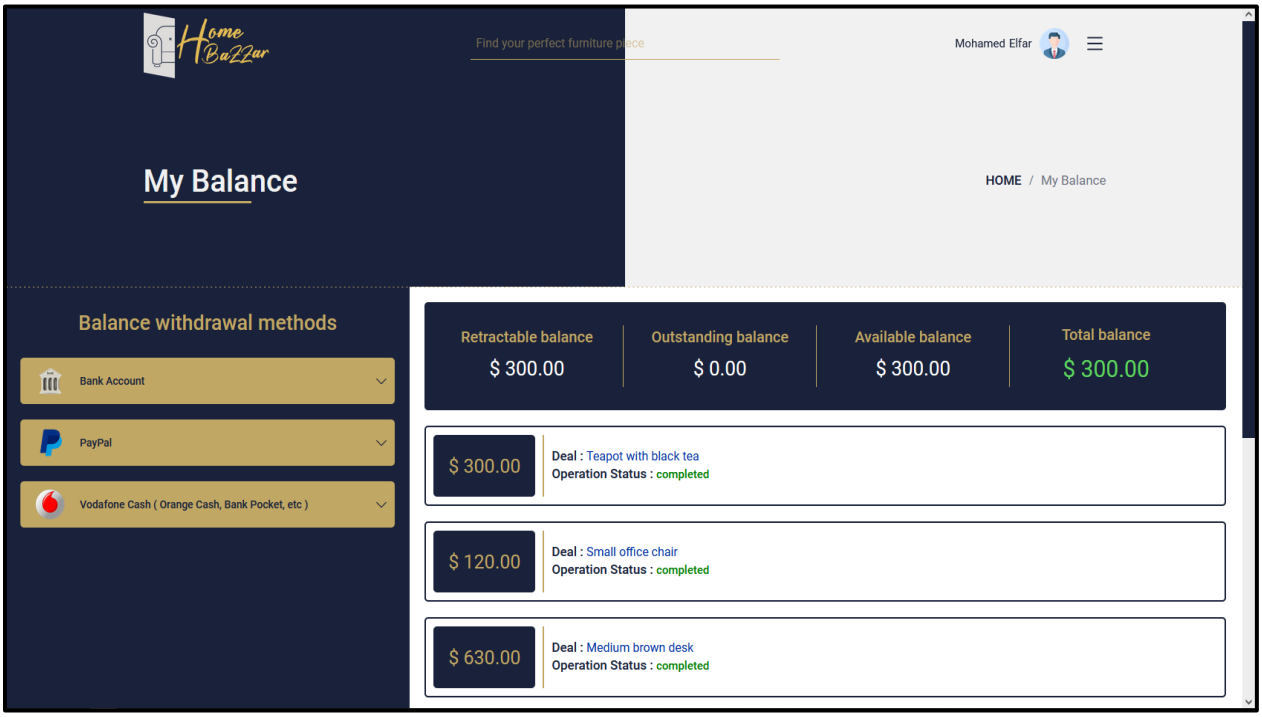


Dashboard:

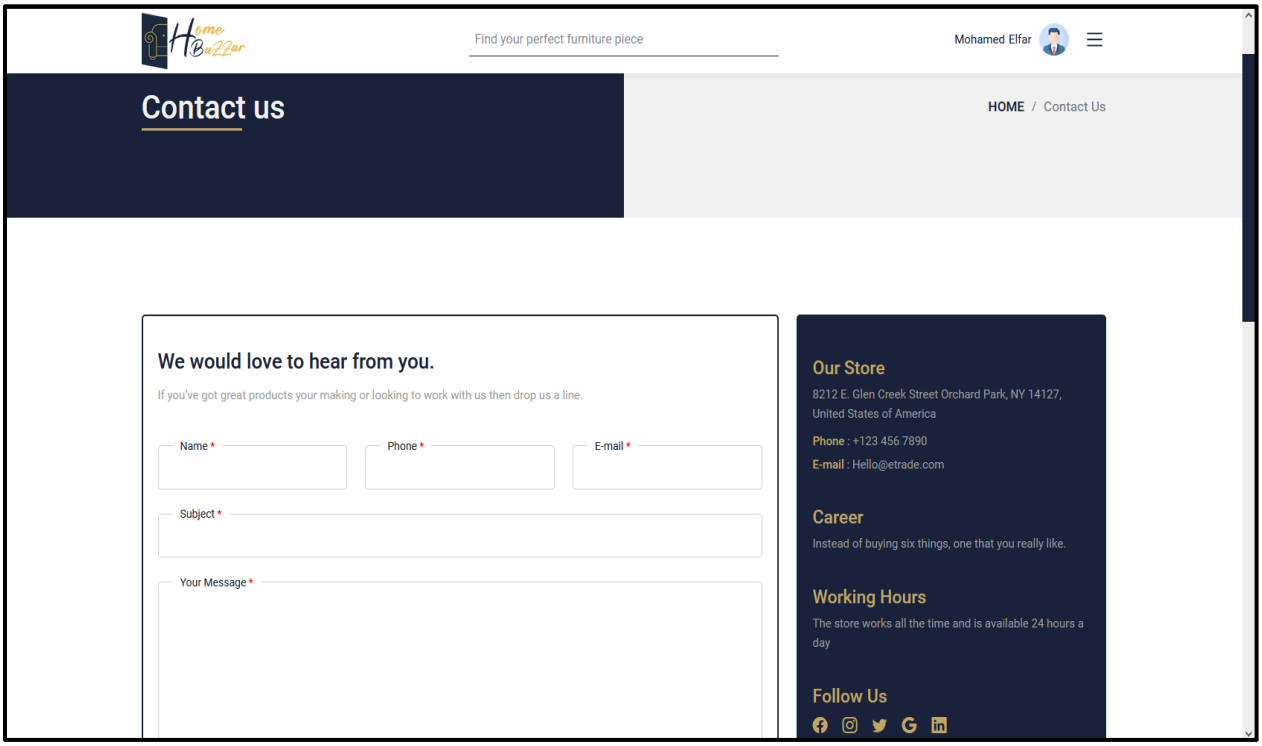

Home BaZZar
Find your perfect furniture piece
Mohamed Elfar
Dashboard
HOME / Dashboard
My Products
Add New Product
30%
3%
30%
(0 review)
chair...
$200 - $300
(0 review)
welcom new...
$333 - $33
(0 review)
chair...
$200 - $300


Home BaZZar
Find your perfect furniture piece
Mohamed Elfar
My Products
Add New Product
Product Title *
Product Images *
Choose a Image or Drag it here
New Price *
Old Price (if Founded)
Product Features *
Add to List
Discount Percentage (if Founded)
Type your product discount
Product Size *
Select Size
Product Categories *
Select Size
Product Colors *
Select Colors
Product Case *
Select Product Case
Product Abstract *
type a simple abstract about your product...
Product Video *
Put Video Link here
Note | Upload the video to YouTube, then put the link here
Select option*
Select Product Case(category)
Note | Is your product a special piece, that is, it is specially designed or of a distinctive style?
Product Details *
type all details about your product...

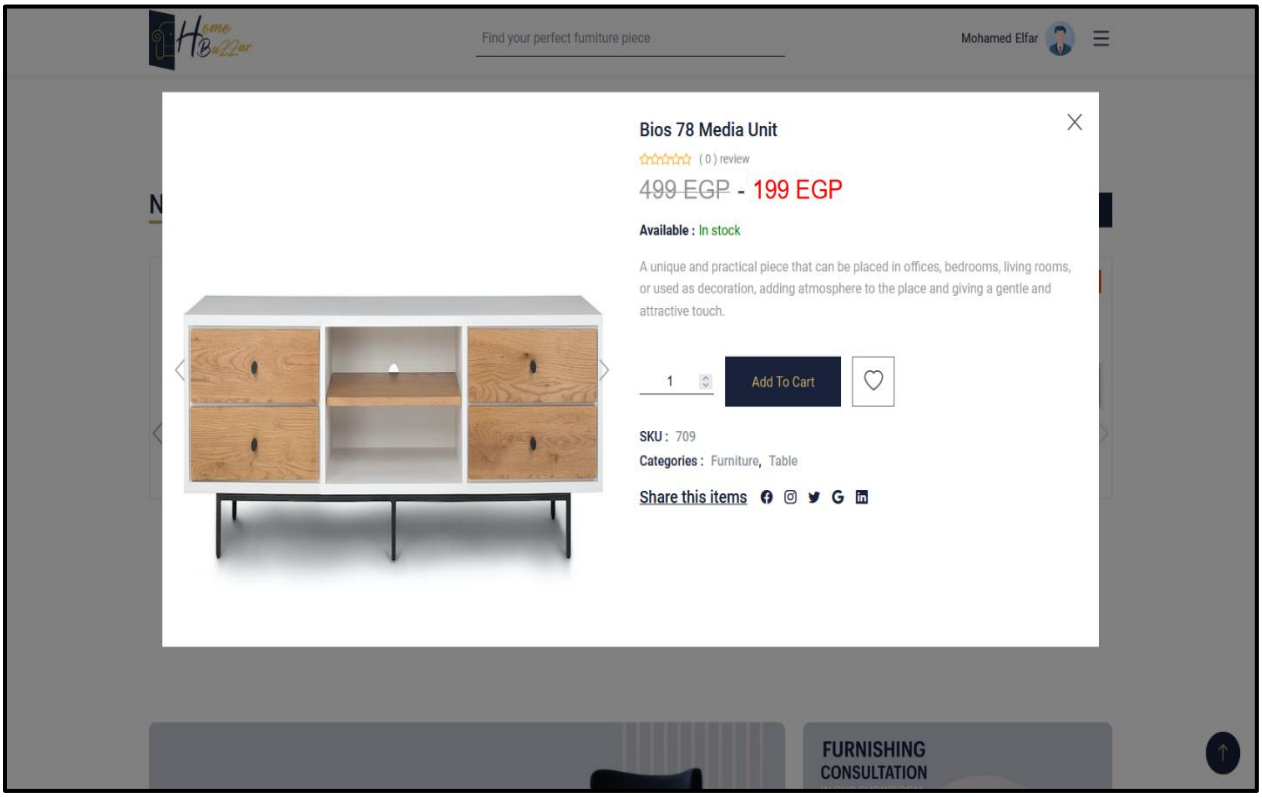
My Balance:

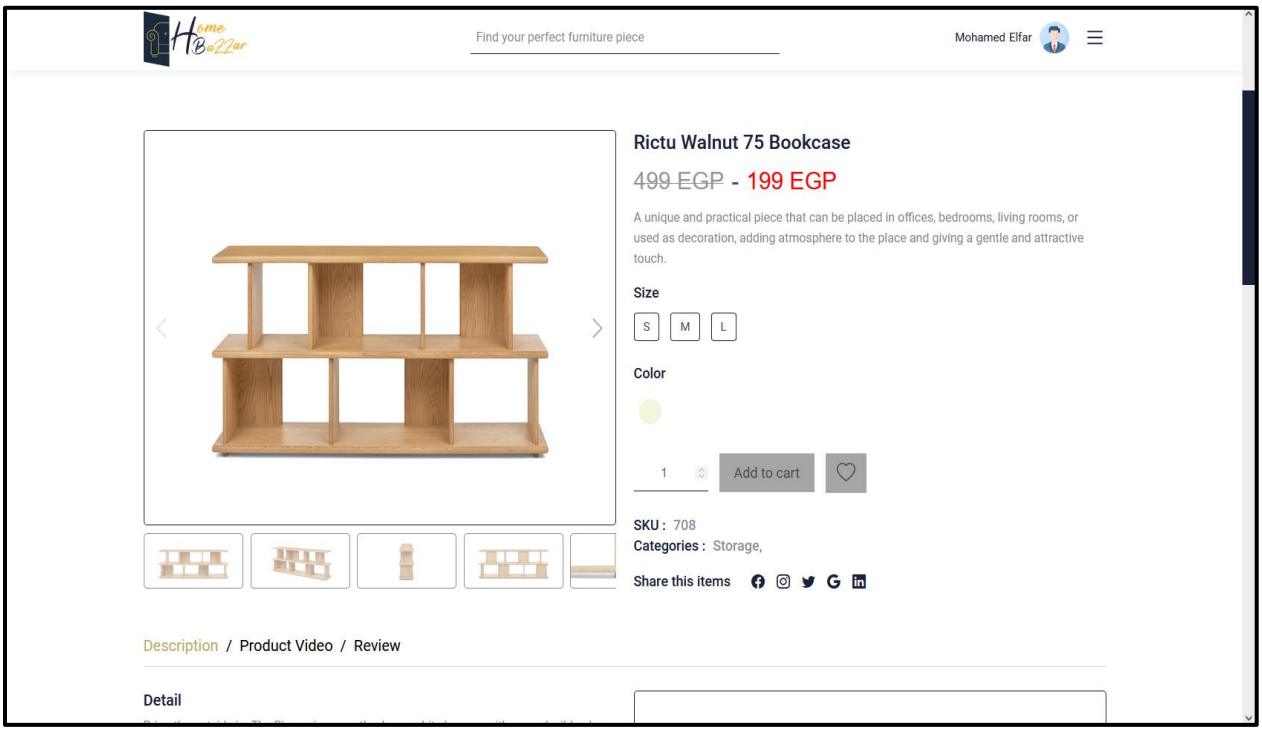


Contact Us:

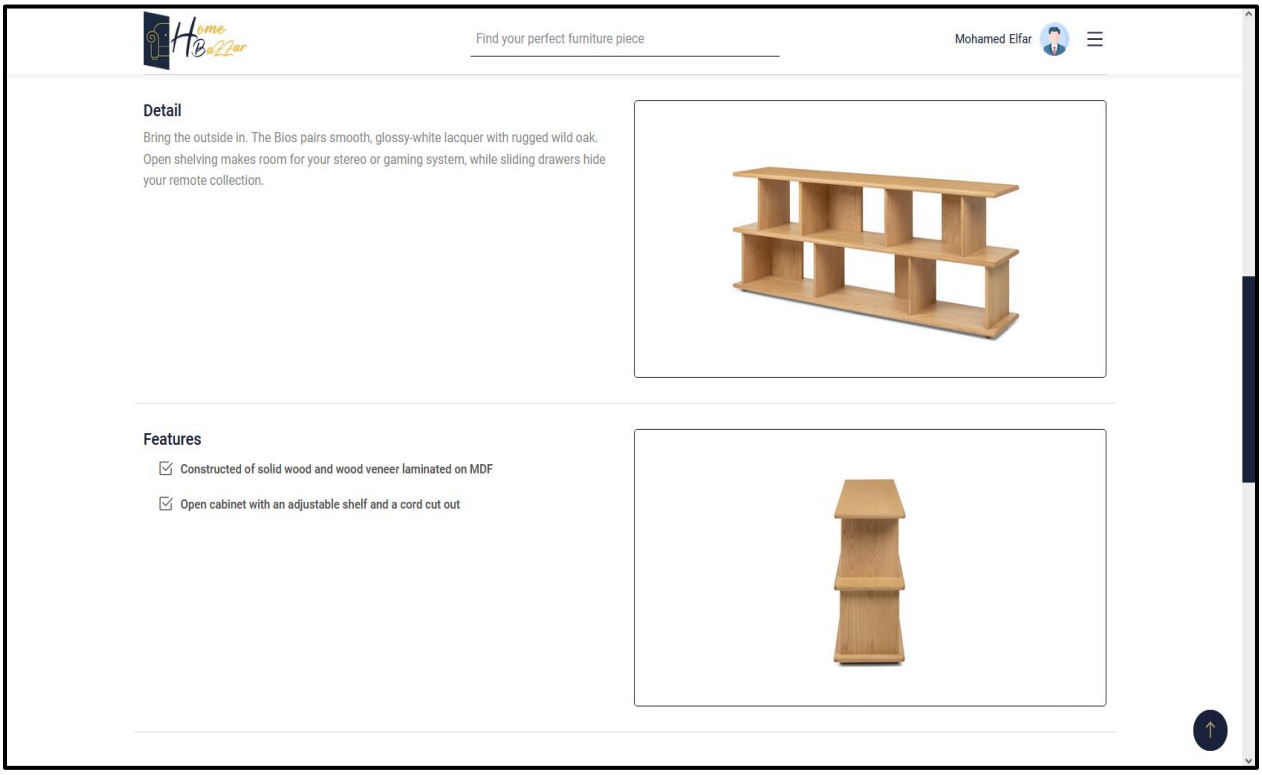


Ordering / Add to Cart:

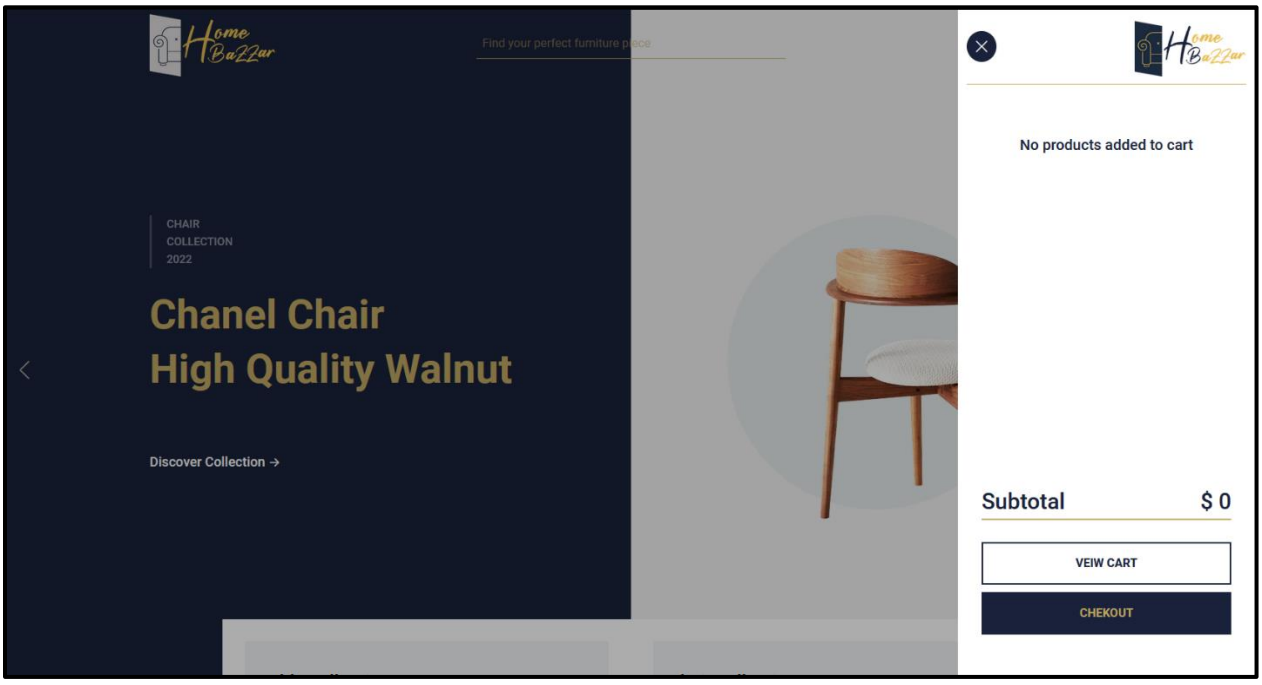




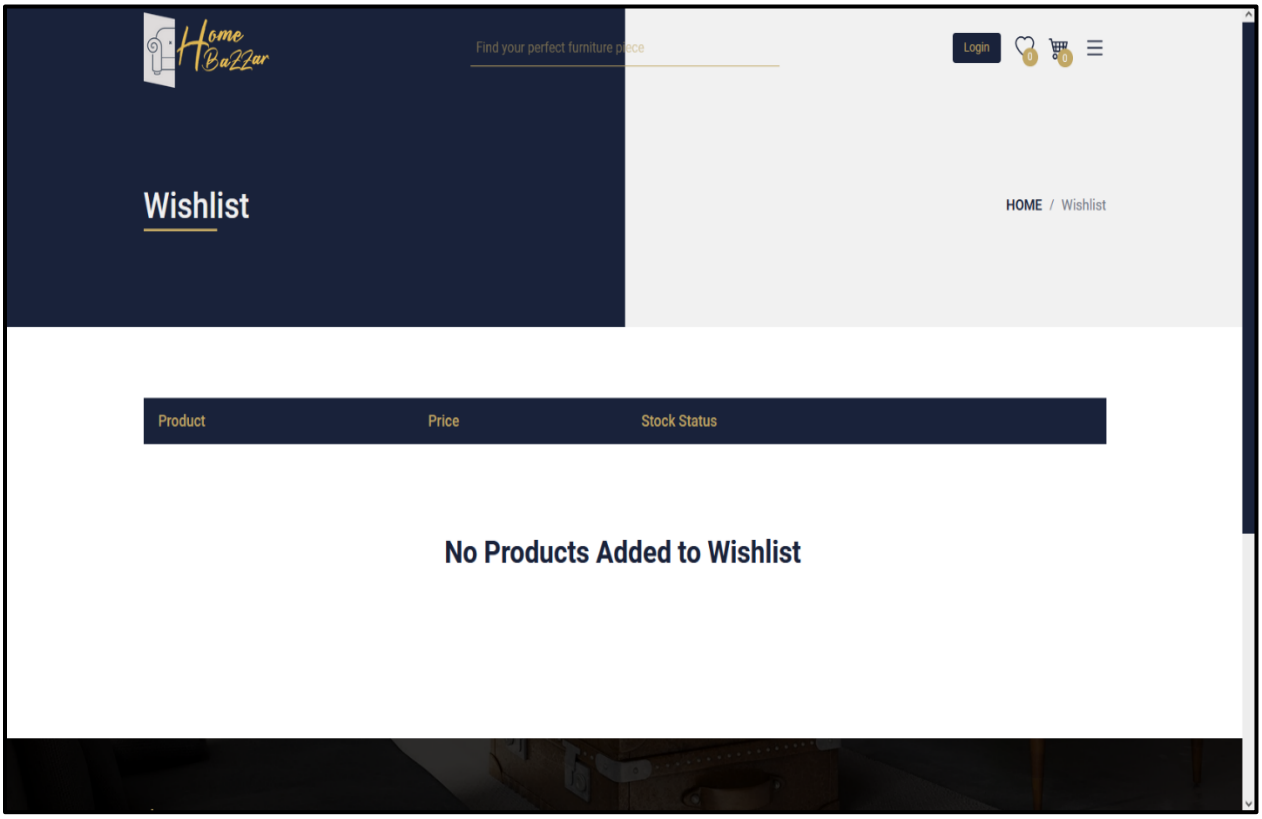
Exploring product:

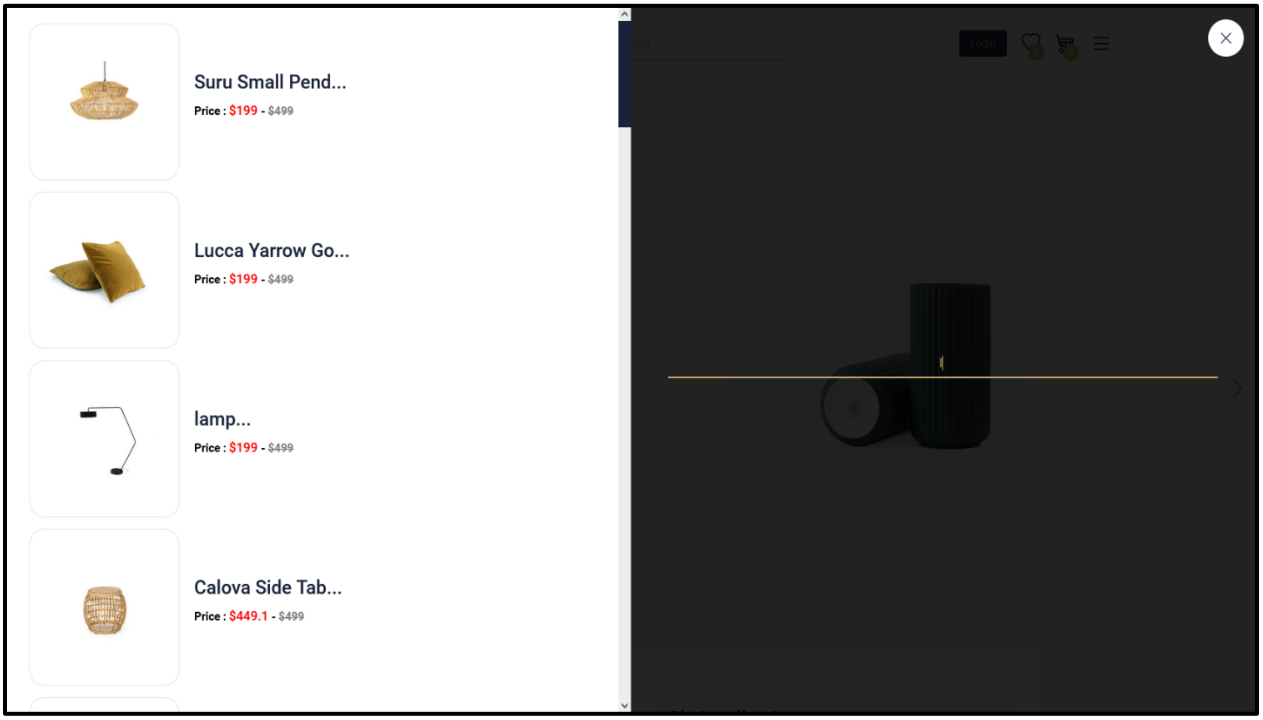


Checkout:

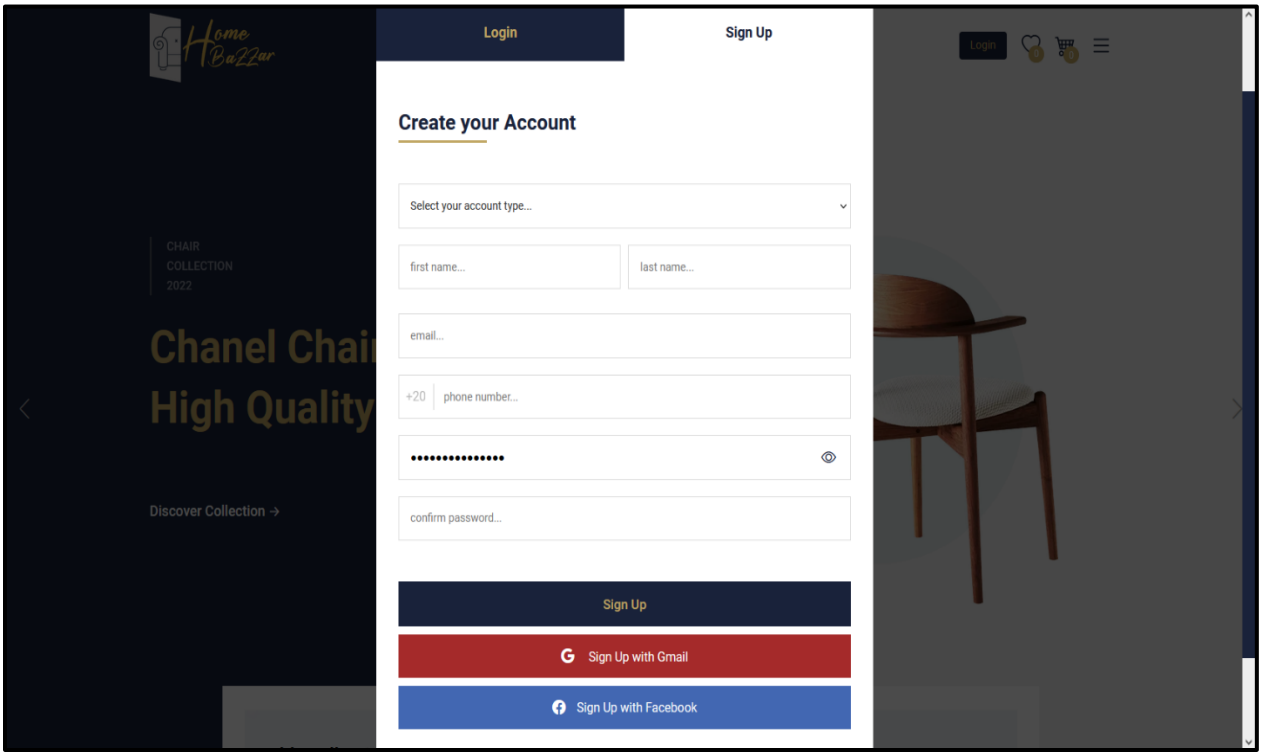


Wishlist:

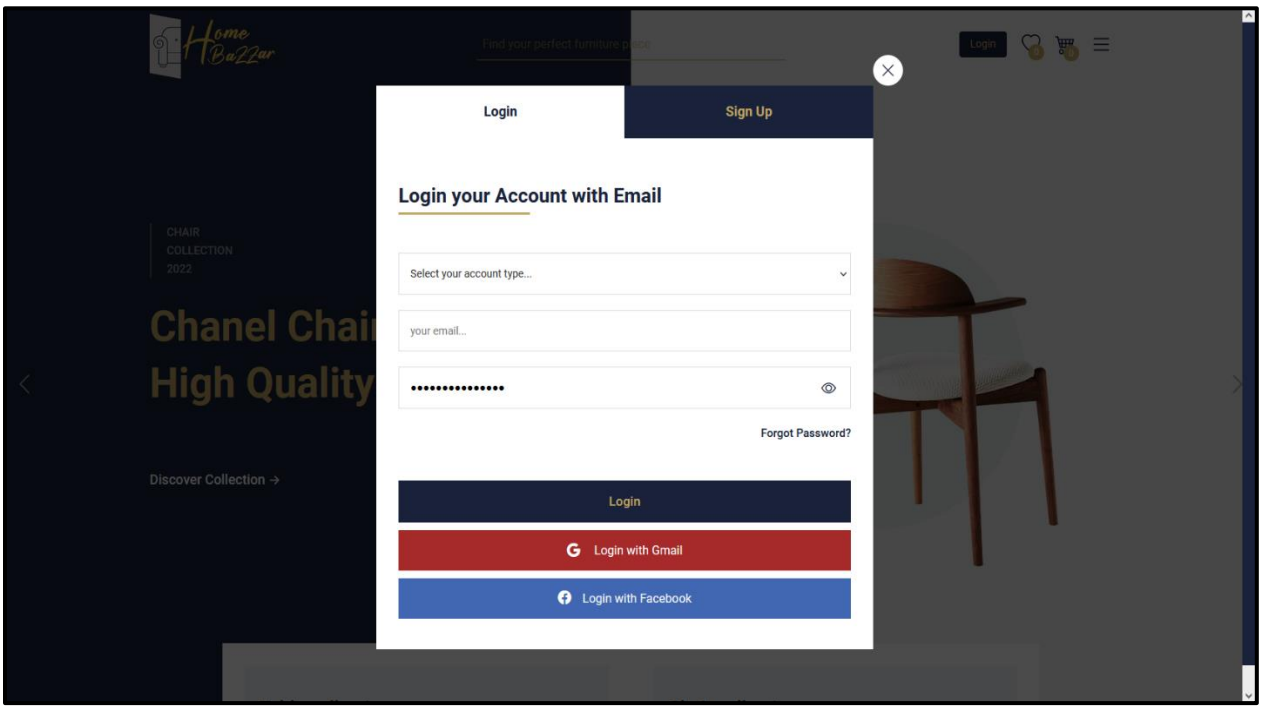

Home BaZZar
Find your perfect furniture piece
Login
Wishlist
HOME / Wishlist
Product
Price
Stock Status
No Products Added to Wishlist


Suru Small Pend...
Price : $199 - $499
Lucca Yarrow Go...
Price : $199 - $499
lamp...
Price : $199 - $499
Calova Side Tab...
Price : $449.1 - $499

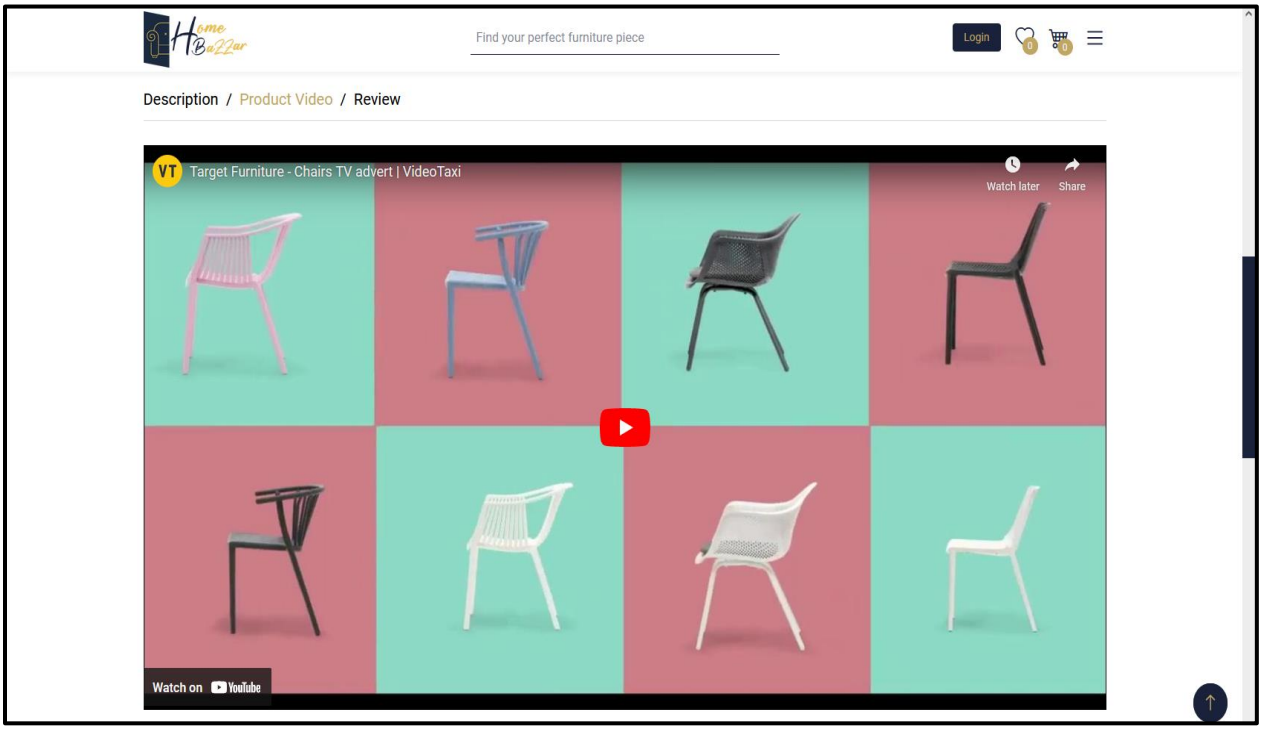
Create Account:

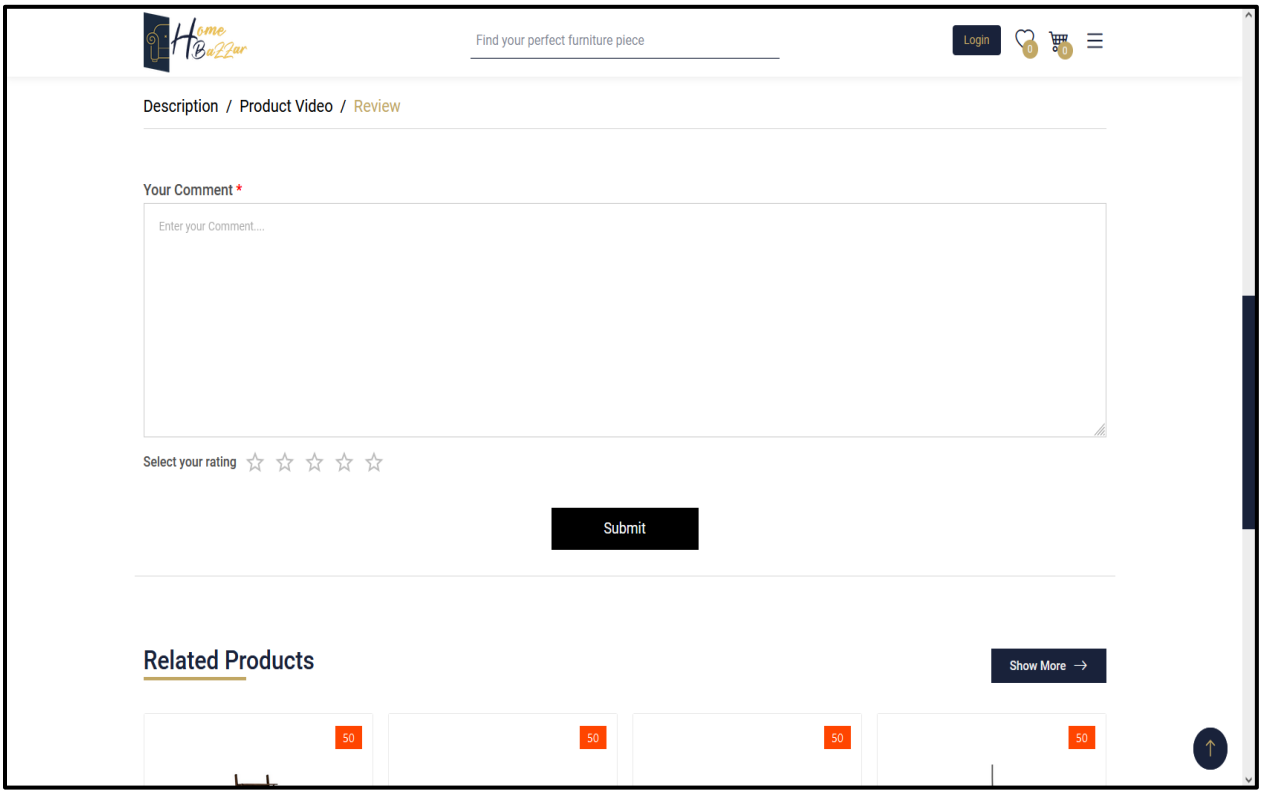


Sign Up / Login:

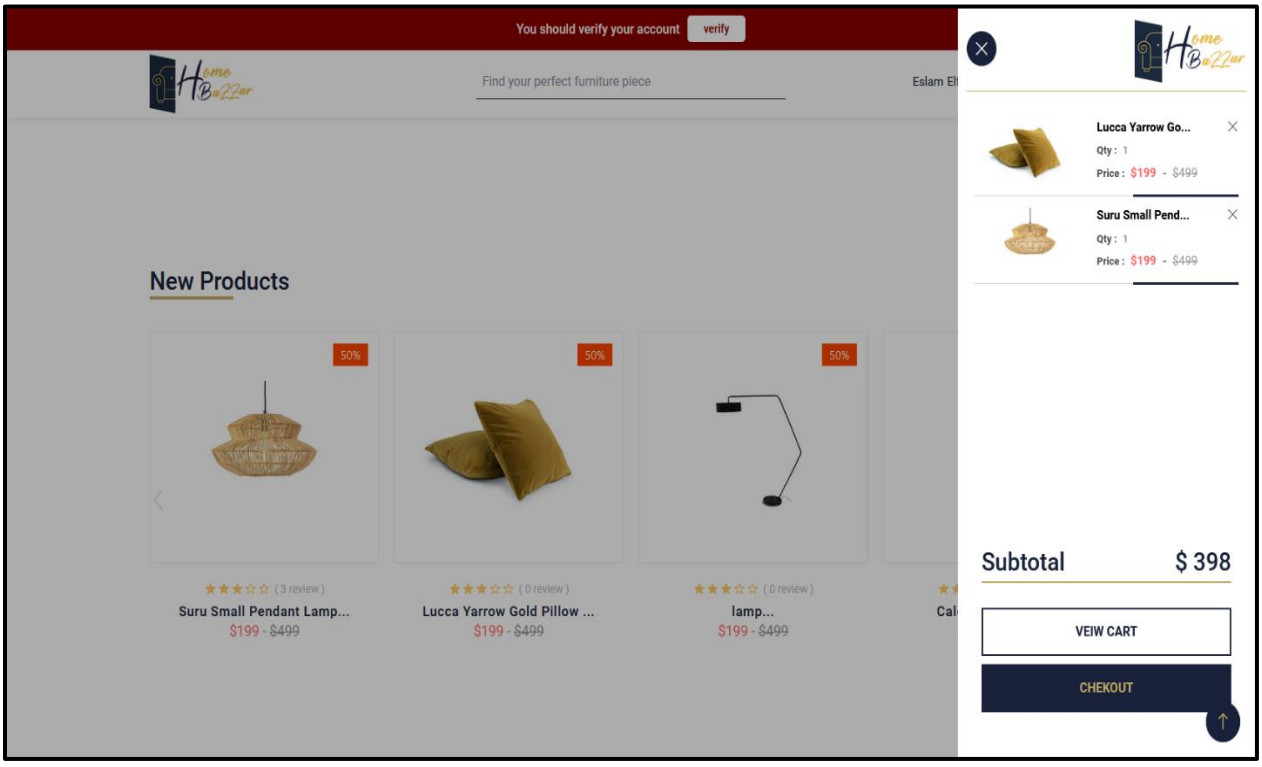


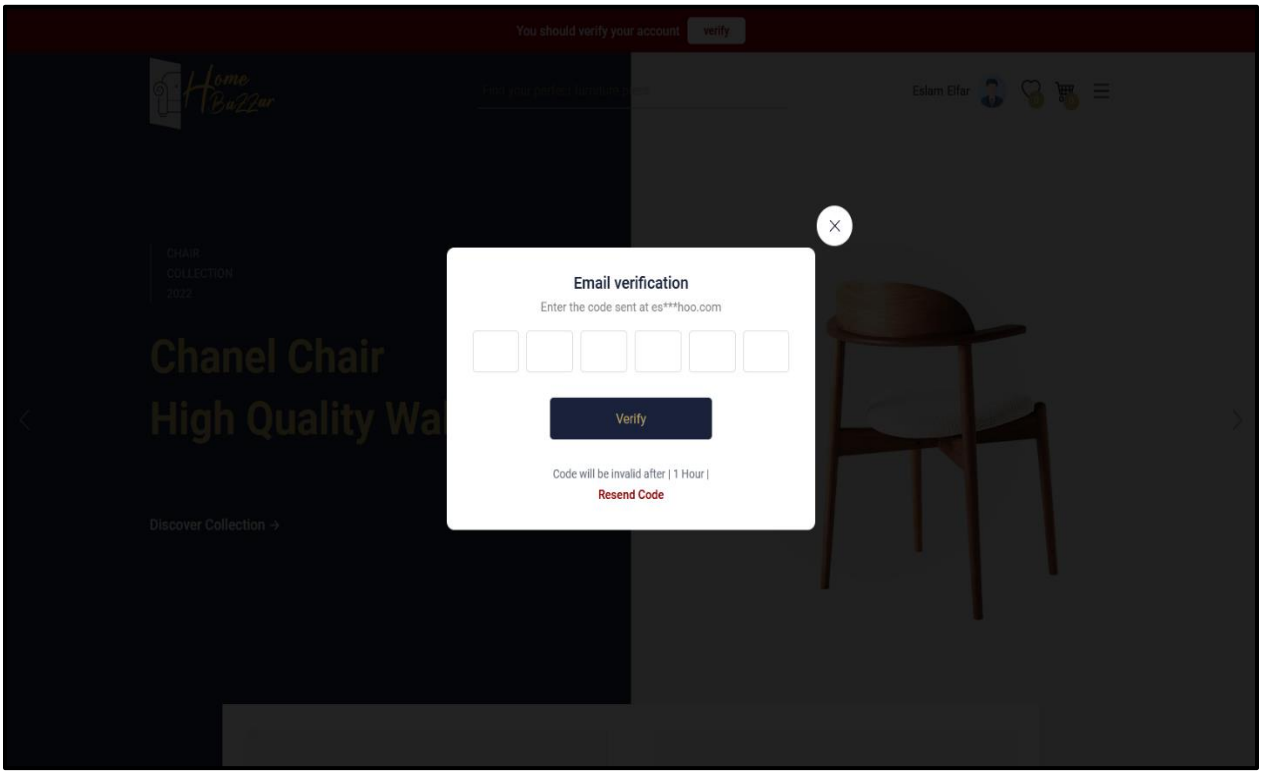
Product Video / Review / Related Products





Email Verification:



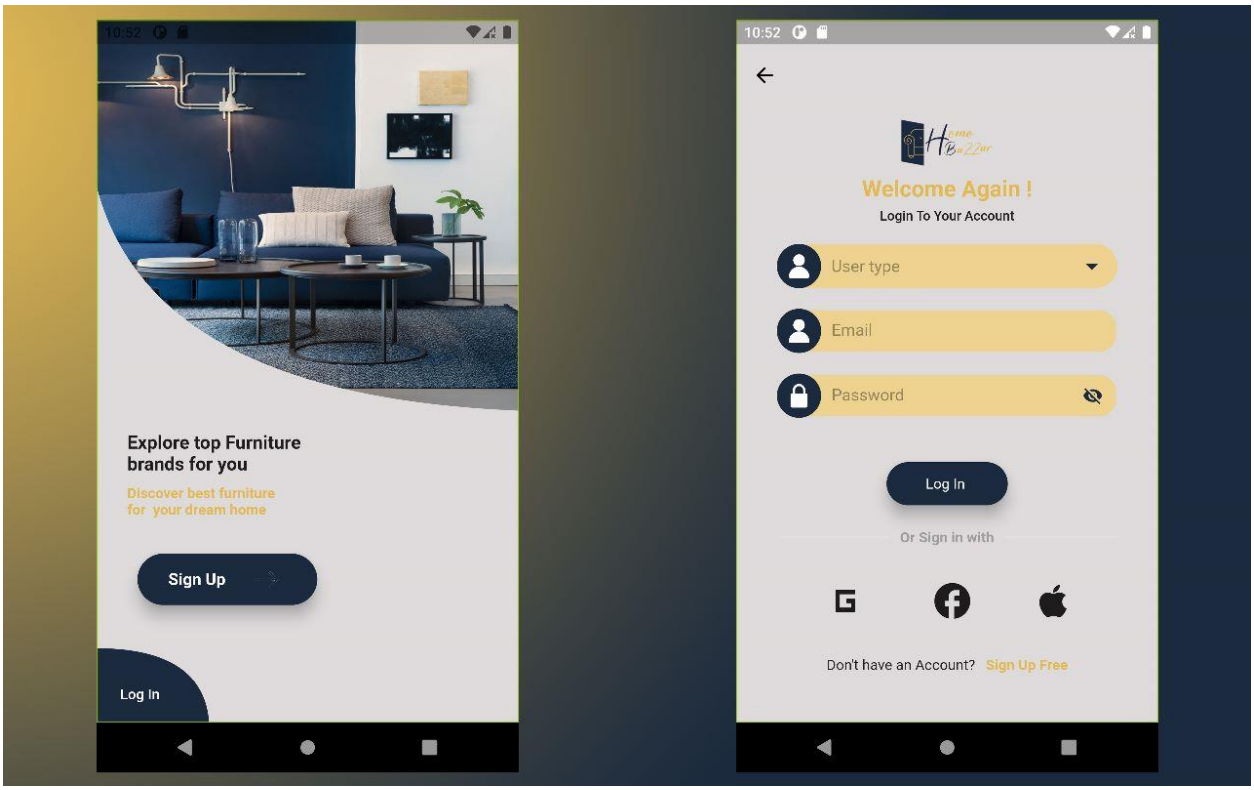


5.2. Interface Design of the mobile application:

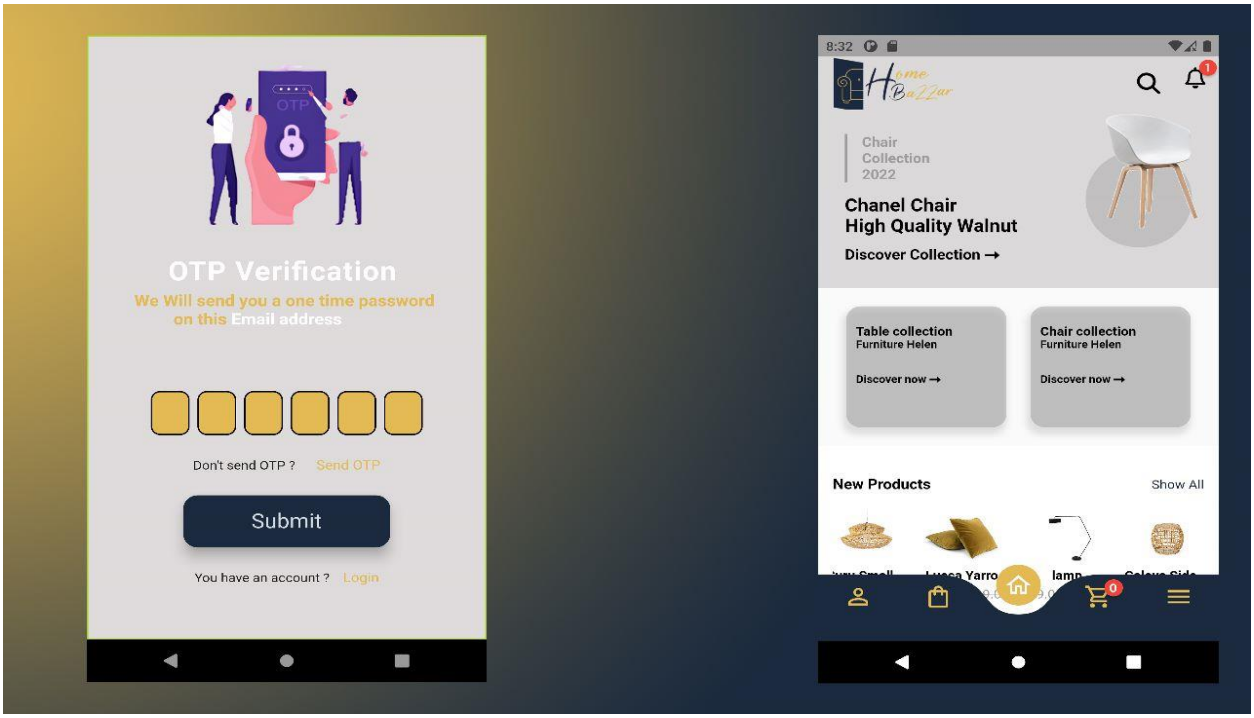
The following sequence of Figures illustrates the interface design.

Figures 4.4.2 Interface Design of the mobile application.

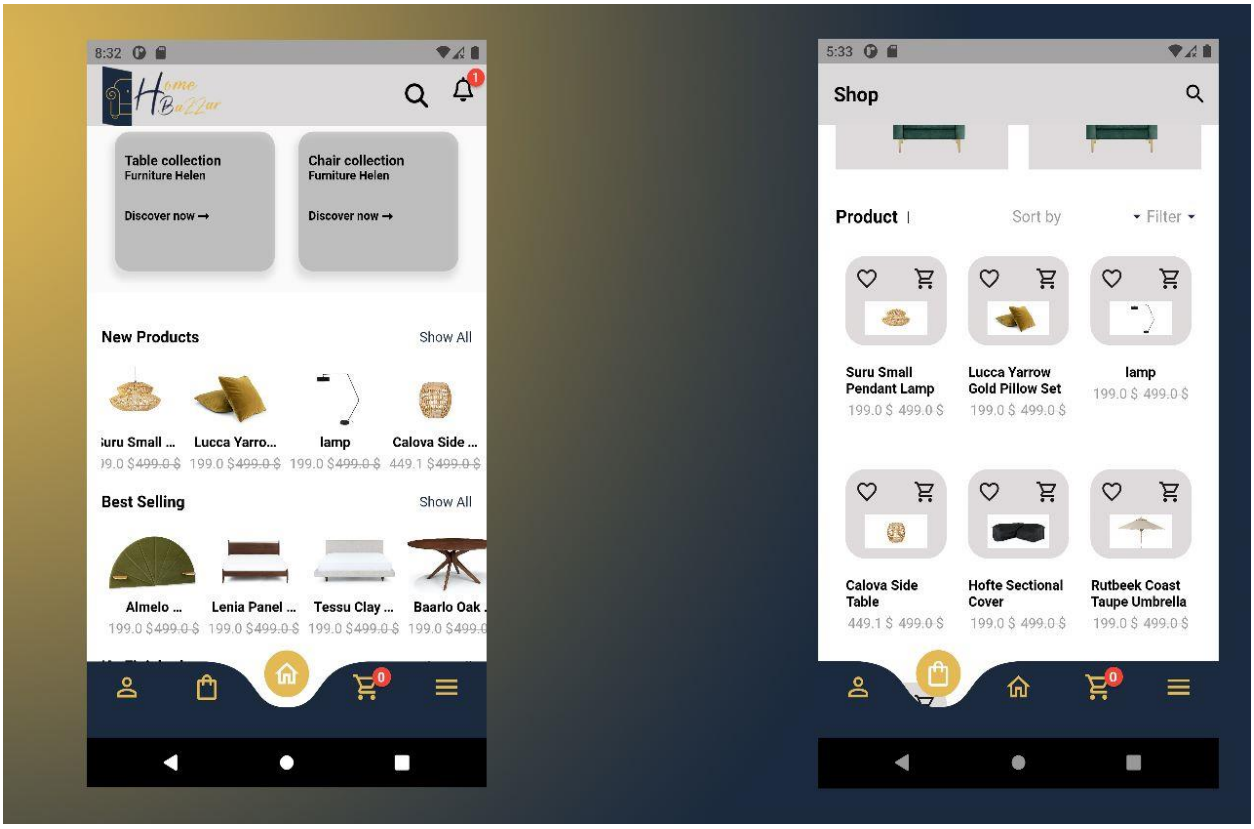
Sign Up / Login:



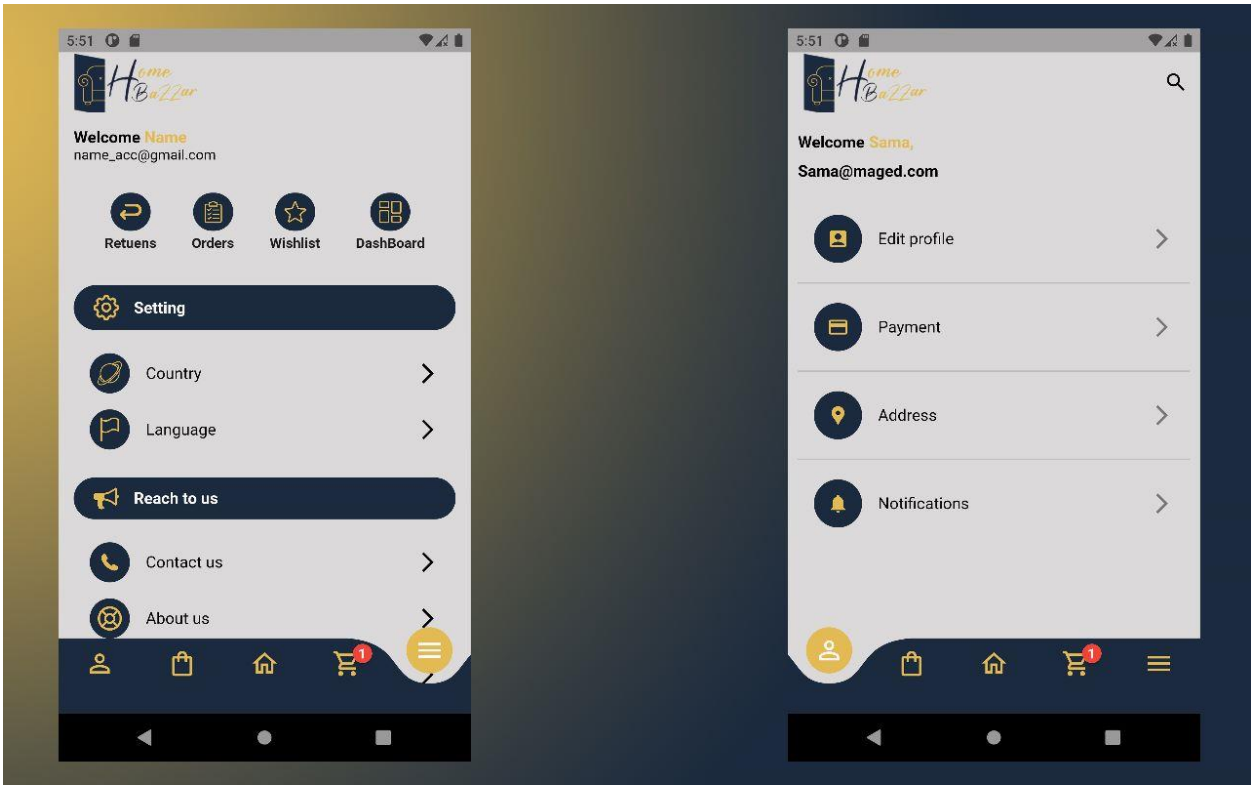
Verification / Home:



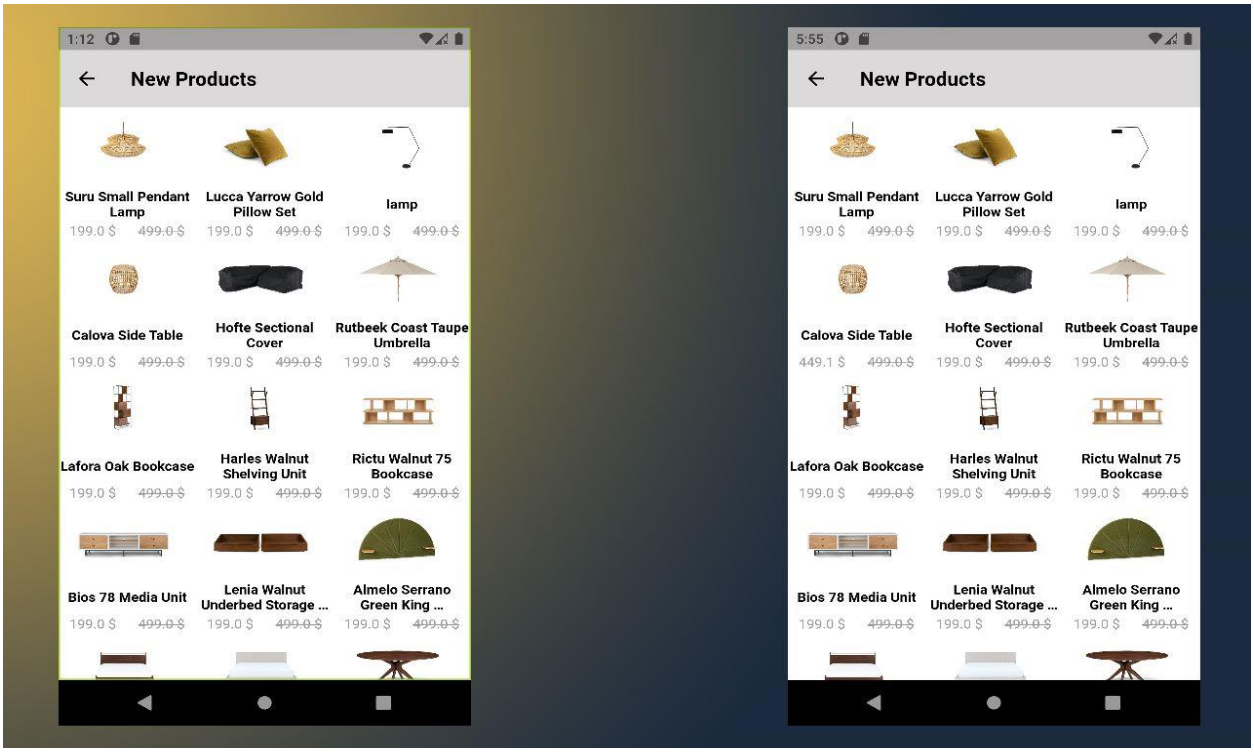
New Products / Best Selling / Shop:



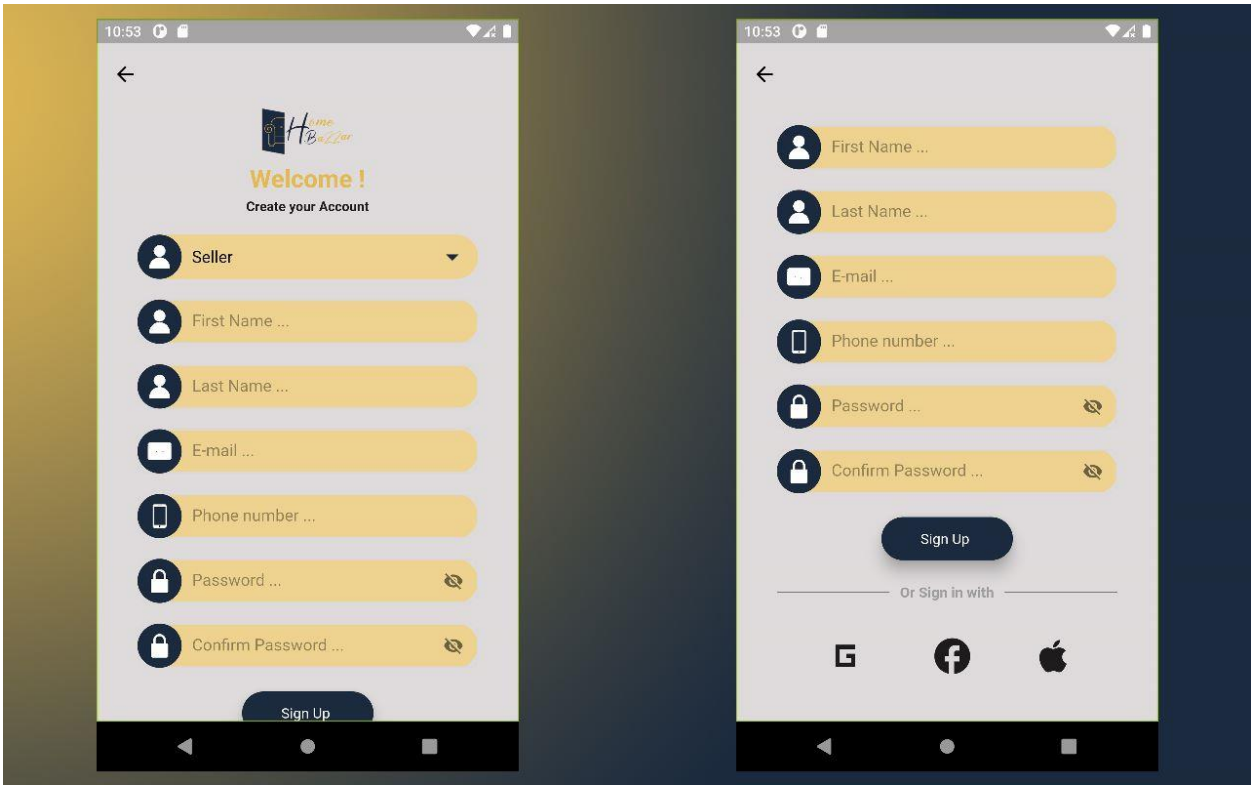
Setting / Reach To Us:



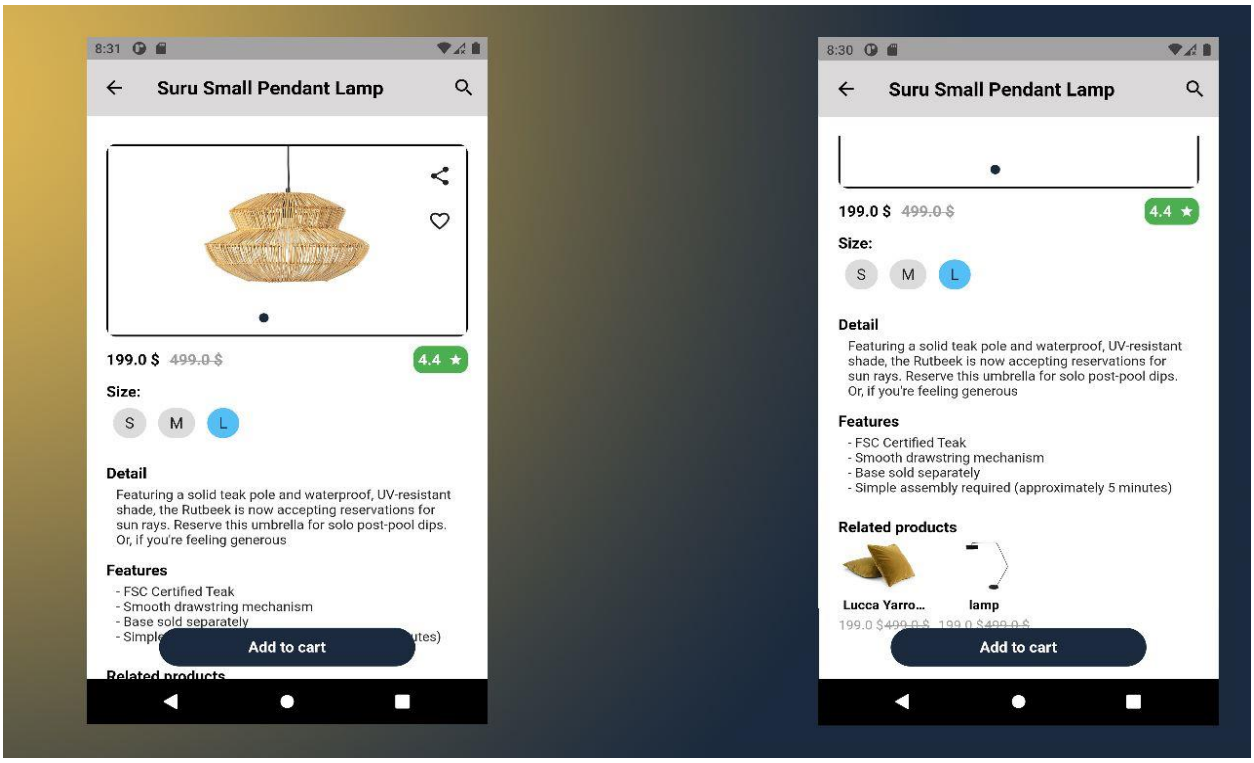
New Products:



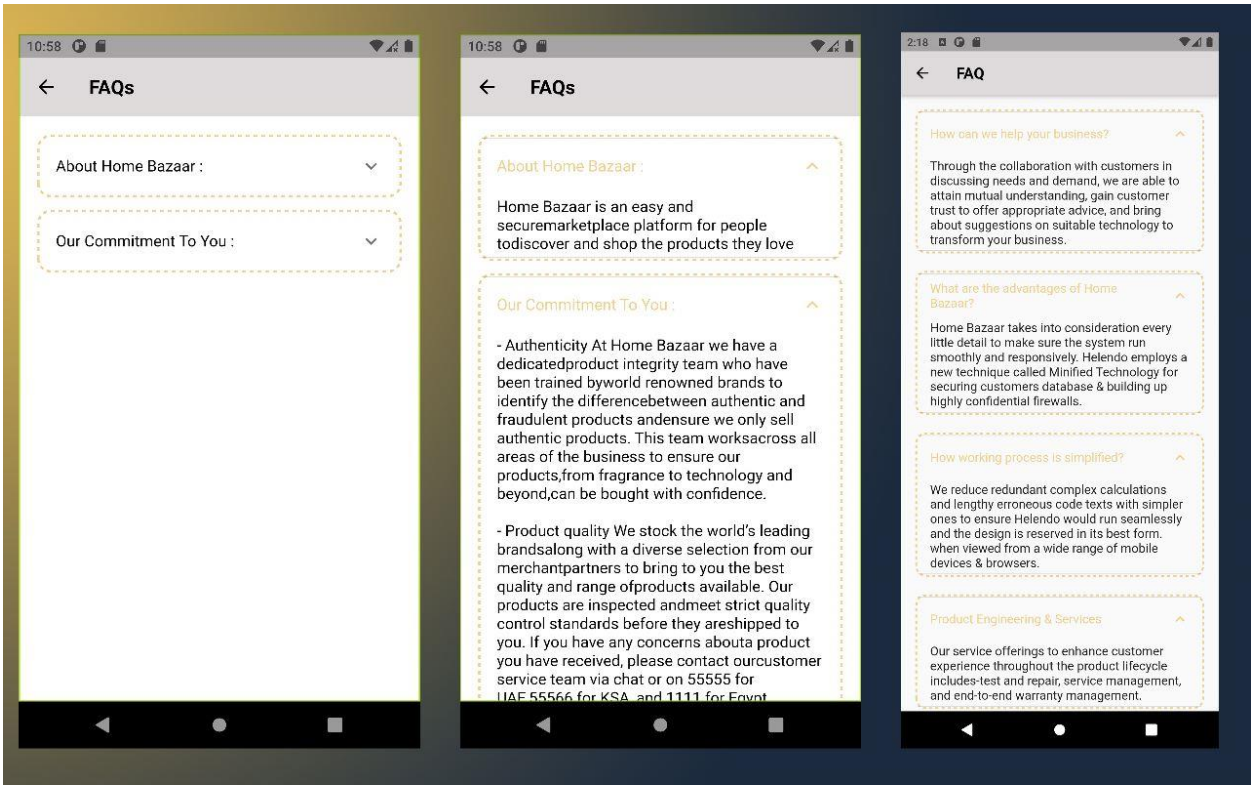
Account:



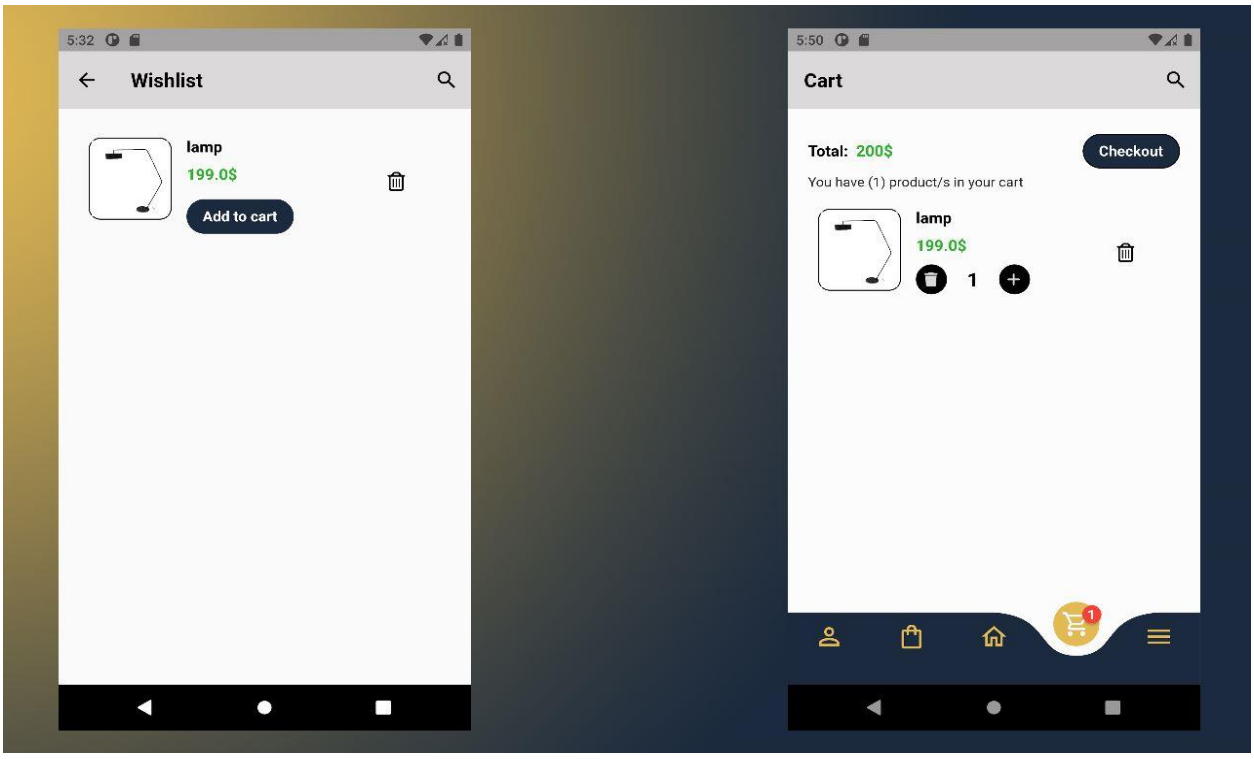
Product:



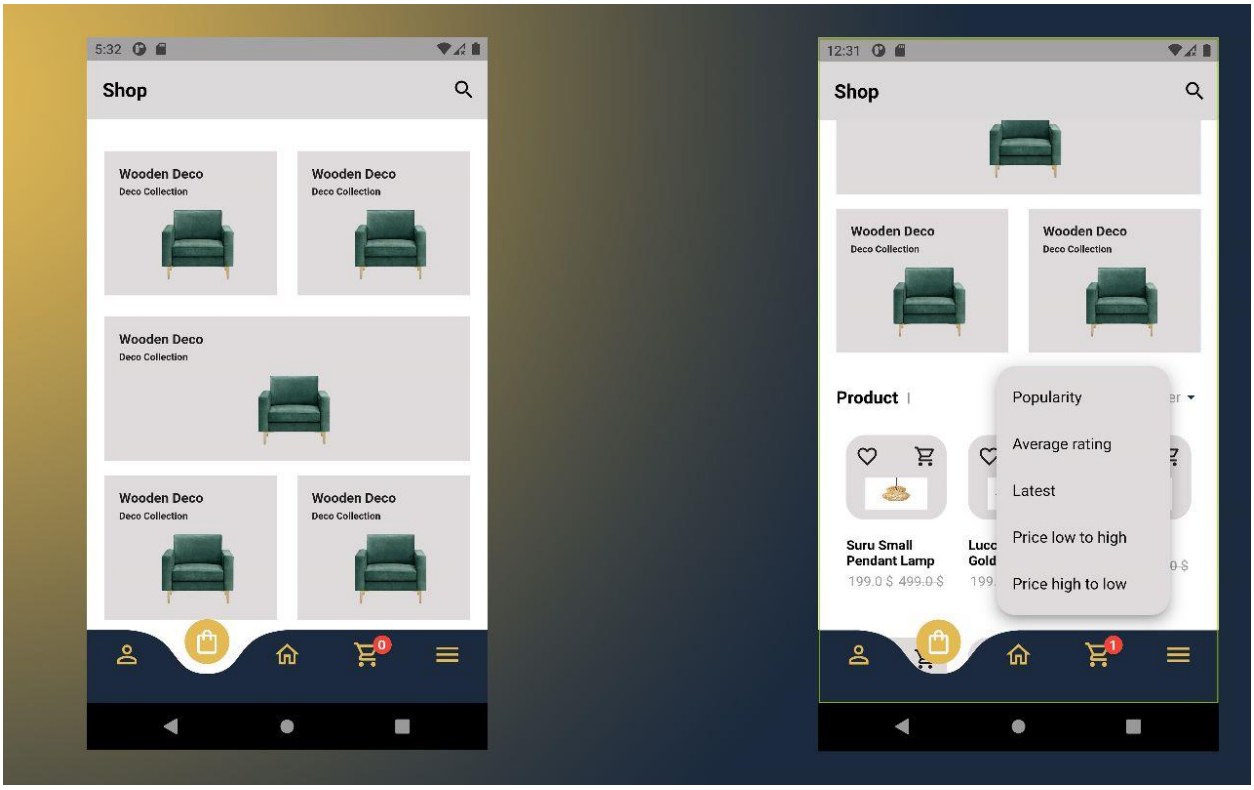
FAQs:



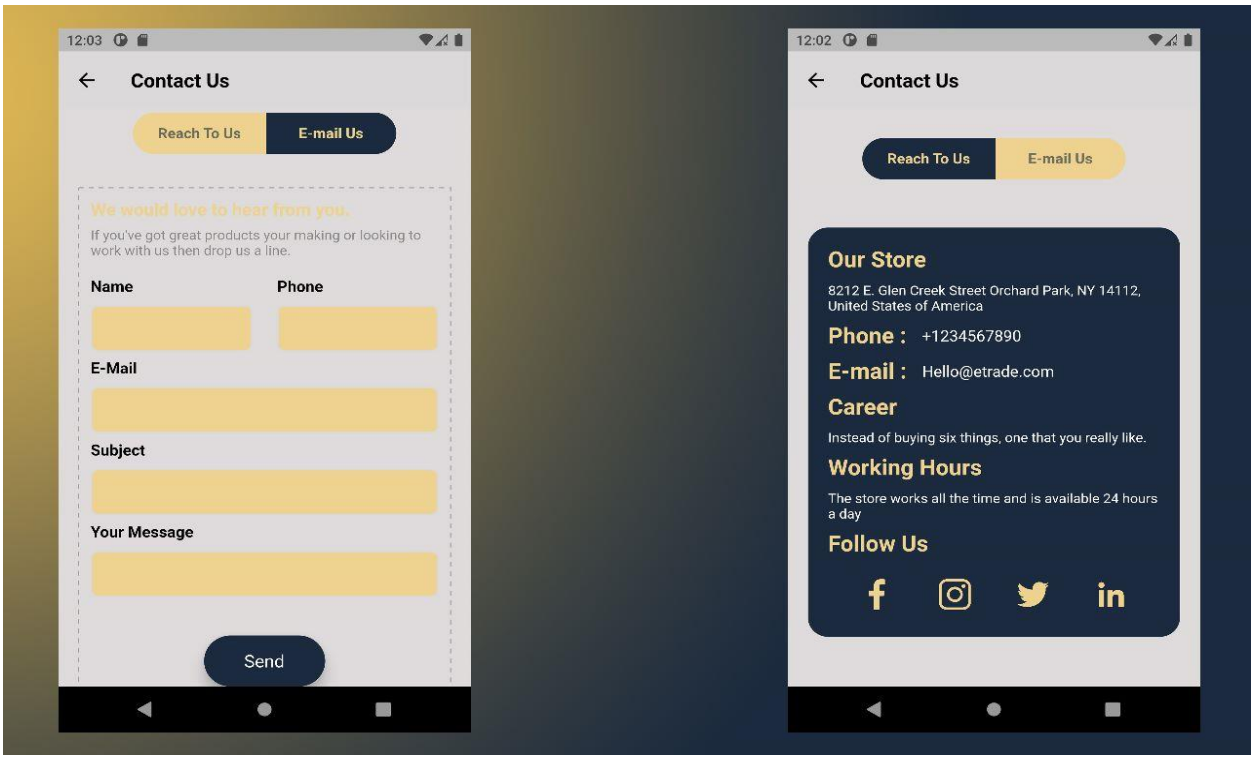
Wishlist / Cart:



Shop:



Contact Us:



5.3. Code of the backend of the website and the mobile application:

```php
<?php

namespace App\Http\Controllers\Api;

use App\Models\User;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Validator;
use App\Notifications\EmailVerificationNotification;

class AuthController extends Controller
{
   /**
     * Create a new AuthController instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth:api', ['except' => ['login', 'register']]);
    }
    /**
     * Get a JWT via given credentials.
     *
     * @return \Illuminate\Http\JsonResponse
     */
```

Login:

```
public function login(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'email'    => 'required|string|email',
            'password' => 'required|string|min:6',
        ]);



        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }
        if (! $token = auth()->attempt($validator->validated())) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }
        return $this->createNewToken($token);
    }

    /**
     * Register a User.
     *
     * @return \Illuminate\Http\JsonResponse
     */
```

Register:

```
public function register(Request $request)
    {
        //dd($request);
        $validator = Validator::make($request->all(), [
            'type'       => 'sometimes|required|integer|in:0,1',   // allow
customer or seller only
            'first_name' => 'required|string|between:2,100',
            'last_name'  => 'nullable|string|between:2,100',
            'email'      => 'required|max:100|unique:users',
            'phone'      => 'required|string|min:6|unique:users',
            'password'   => 'required|string|confirmed|min:6',
        ]);
        //dd($validator->validated());
```

```
        if ($validator->fails()) {
            return response()->json($validator->errors(), 400);
        }
        $user = User::create(array_merge(
            $validator->validated(),
            ['password' => bcrypt($request->password)]
        ));
        $user->refresh();

        $user->notify(new EmailVerificationNotification());

        return response()->json([
            'message' => 'User successfully registered',
            'user' => $user
        ], 201);
    }

    /**
     * Log the user out (Invalidate the token).
     *
     * @return \Illuminate\Http\JsonResponse
     */
```

Logout:

```
public function logout()
    {
        auth()->logout();
        return response()->json(['message' => 'User successfully signed out']);
    }

       protected function createNewToken($token)

    {
        return response()->json([
            'access_token' => $token,
            'token_type' => 'bearer',
            'expires_in' => auth()->factory()->getTTL(),
            'user' => auth()->user()
        ]);}
```

Update User Account:

```
public function update(Request $request){
        $user = Auth::user();

        $validator = Validator::make($request->all(), [
            'first_name' => 'sometimes|string|max:255',
            'last_name' => 'sometimes|string|max:255',
            'email' =>
'sometimes|string|email|max:255|unique:users,email,'.$user->id,
            'email' => 'sometimes|string|max:20|unique:users,phone,'.$user->id,
            'current_password' => 'sometimes|string|min:6',
            'password' => 'nullable|string|min:6|confirmed',
        ]);

        if ($validator->fails()) {
            return response()->json(['error' => $validator->errors()], 422);
        }

        if ($request->filled('first_name')) {
            $user->update(['first_name' => $request->input('first_name')]);
        }
        if ($request->filled('last_name')) {
            $user->update(['last_name' => $request->input('last_name')]);
        }
        if ($request->filled('email')) {
            $user->update(['email' => $request->input('email')]);
        }
        if ($request->filled('phone')) {
            $user->update(['phone' => $request->input('phone')]);
        }

        // Verify user's current password
        if (!Hash::check($request->input('current_password'), $user->password)) {
            return response()->json(['error' => 'Current password is
incorrect.'], 422);
        }

        if ($request->filled('password')) {
            $user->update(['password' => bcrypt($request->input('password'))]);
        }

        return response()->json(['message' => 'User account updated
successfully.'], 200);
    }}
```

Checkout:

```php
<?php
namespace App\Http\Controllers;
use App\Models\Cart;
use App\Models\User;
use App\Models\Order;
use App\Mail\OrderPlaced;
use Illuminate\Http\Request;
use App\Mail\AdminOrderPlaced;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Mail;
use App\Http\Resources\CheckOutResource;
use Illuminate\Support\Facades\Validator;

class CheckOutController extends Controller
{
    public function checkout(Request $request)
    {
        $cart = Cart::where('user_id', $request->user()->id)->get();

        if ($cart->isEmpty()) {
            return response()->json(['message' => 'Cart is empty'], 400);
        }

        $validator = Validator::make($request->all(), [
            'first_name' => 'required|string|between:2,100',
            'last_name'  => 'nullable|string|between:2,100',
            'phone'      => 'required|string|min:6|unique:users',
            'address' => ['required', 'string'],
            'company' => ['nullable', 'string'],
            'total' => ['nullable', 'string'],
            'subtotal' => ['nullable', 'string'],
            'shipingCost' => ['nullable', 'string'],
            'payment' => ['required', 'string', 'in:0,1,2'],
            'notes' => ['nullable', 'string'],

        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        $order = new Order();
```

```
        $order->user_id = Auth::id();
        $order->product_id = $cart->first()->product_id;
        $order->fill($validator->validated());
        $order->save();

    // Send email to the user.
    Mail::to($request->user()->email)->send(new OrderPlaced($order,$cart));

    // Send email to the admin.
    $user = User::find($order->user_id);
        if ($user->type !== 2) {
            $admins = User::where('type', 2)->get();
            foreach ($admins as $admin) {
                Mail::to($admin->email)->send(new
AdminOrderPlaced($order,$cart));
            }
        }

    Cart::where('user_id', $request->user()->id)->delete();

        return [(new CheckOutResource($order))];
    }
}
```

```php
<?php

namespace App\Http\Controllers;

use App\Models\Product;
use App\Traits\MediaTrait;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use App\Http\Resources\ProductResource;
use App\Models\User;
use Illuminate\Support\Facades\Validator;

class ProductController extends Controller
{
    use MediaTrait;
```

Add Product:

```php
public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'title'    => 'required',
            'newPrice' => 'required',
            'oldPrice'    => 'nullable',
            'offer'    => 'sometimes',
            'category'    => 'required',
            'color'    => 'required',
            'size'    => 'required',
            'abstract'    => 'required',
            'featuer'    => 'required',
            'pin_code'    => 'sometimes|max:3|unique:products',
            'description'    => 'required',
            'videos'    => 'required',
            'is_special'=> 'required',
            'is_complete'=> 'required',
            'images'    => 'sometimes|file|image|mimes:jpg,gif,png,webp',


            'image_1'    => 'sometimes|file|image|mimes:jpg,gif,png,webp',
            'image_2'    => 'sometimes|file|image|mimes:jpg,gif,png,webp',
            'image_3'    => 'sometimes|file|image|mimes:jpg,gif,png,webp',
            'image_4'    => 'sometimes|file|image|mimes:jpg,gif,png,webp',
```

```
            'image_5'    => 'sometimes|file|image|mimes:jpg,gif,png,webp',
            'image_6'    => 'sometimes|file|image|mimes:jpg,gif,png,webp',
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        $product = new Product();
        $product->user_id= Auth::id();
        $product->fill($validator->validated());
        $product->save();

        $this->handleRequestMediaFiles($product, $request);

        $product = Product::with('media')
            ->withCount('reviews')
            ->findOrFail($product->id);

        return new ProductResource($product);
    }
```

Get One Product:

```
public function show($id)
    {
        // get the product with media images and reviews count
        $product = Product::with('media', 'reviews', 'reviews.user')
            ->withCount('reviews')
            ->rating()
            ->findOrFail($id);

        return new ProductResource($product);
    }
```

Show All Products:

```
public function index(Request $request)
    {
        $products = Product::with('media')
            ->withCount('reviews')
            ->rating()
```

New Arrival Products:

```
->when($request->query('new_arrival'), function ($query) {
                $query
                    ->where('created_at', '>=', now()->subMonth())
                    ->orderBy('id', 'DESC');
            })
```

Top Rated Products:

```
->when($request->query('top_rated'), function ($query) {
                $query
                    ->having('rating', '>', 0)
                    ->orderBy('rating', 'DESC');
            })
```

Special Products:

```
->when($request->query('is_special'), function ($query) {
                $query
                    ->where('is_special', '=', 1);
            })->get();

        return ProductResource::collection($products);

    }
```

Get Seller Products:

```
public function sellerProducts(Request $request, $id)
    {
        if ($id != auth()->user()->id) {
            return response()->json(['error' => 'Unauthorized'], 401);
        }

        $products = Product::with('media')
            ->withCount('reviews')
            ->rating()
            ->where('user_id', $id)
            ->when($request->query('new_arrival'), function ($query) {
                $query
                    ->where('created_at', '>=', now()->subMonth())
                    ->orderBy('id', 'DESC');
            })
            ->when($request->query('top_rated'), function ($query) {
                $query
                    ->having('rating', '>', 0)
                    ->orderBy('rating', 'DESC');
            })
            ->when($request->query('is_special'), function ($query) {
                $query
                    ->where('is_special', '=', 1);
            })

            ->get();

        return ProductResource::collection($products);
    }
```

Related Products:

```
public function relatedProducts(Request $request, $id)
    {
        $product = Product::findOrFail($id);

        $products = Product::with('media')
```

```
            ->withCount('reviews')
            ->rating()
            ->whereNot('id', $product->id)
            ->where('category', $product->category)
            ->when($request->query('new_arrival'), function ($query) {
                $query
                    ->where('created_at', '>=', now()->subMonth())
                    ->orderBy('id', 'DESC');
            })
            ->when($request->query('top_rated'), function ($query) {
                $query
                    ->having('rating', '>', 0)
                    ->orderBy('rating', 'DESC');
            })
            ->when($request->query('is_special'), function ($query) {
                $query
                    ->where('is_special', '=', 1);
            })

            ->get();

        return ProductResource::collection($products);
    }
```

Delete Product:

```
public function delete($id){
    $product= Product::find($id);
    $product->delete($id);
    return response()->json('deleted');
}
```

Update Product:

```
public function update(Request $request)
{
```

```
        $product = Product::find($request->id);

        if (!$product) {
            return response()->json(['error' => 'Product not found'], 404);
        }
        $product->user_id= Auth::id();
        $product->update($request->all());
        $product->save();

        $this->handleRequestMediaFiles($product, $request);

        return["updated Successfully"];

}}
```

Review:

```
<?php
namespace App\Http\Controllers;
use App\Models\Review;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Validator;

class ReviewController extends Controller
{
```

Add Review:

```
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'product_id'    => ['required', 'integer', 'exists:products,id'],
            'star'    => ['required', 'integer', 'min:1', 'max:5'],
            'review' => ['required', 'string'],
        ]);
```

```
        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        $review = new Review();
        $review->fill($validator->validated());
        $review->user_id = Auth::id();
        $review->save();

        return response($review, 200, ["added successfully"]);
    }
```

Delete Review:

```
    public function delete($id){
        $product= Auth::user()->reviews()->findOrFail($id);
        $product->delete($id);
        return response()->json('Review Deleted');
    }
}
```

Customer Middleware:

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class IsCustomer
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
```

```
     * @param  \Closure(\Illuminate\Http\Request):
(\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)  $next
     * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     */
    public function handle(Request $request, Closure $next)
    {
        if ($request->user() && ( $request->user()->type === 0)) {
            return $next($request);
        }

        return $request->expectsJson()
            ? response([
                'message' => 'Access Denied.',
            ], 403)
            : abort(403, 'Access Denied.');
    }

}
```

Seller Middleware:

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class IsSeller
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure(\Illuminate\Http\Request):
(\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)  $next
     * @return \Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     */
    public function handle(Request $request, Closure $next, ...$guards)
    {
        if ($request->user() && ( $request->user()->type === 1)) {
```

```
            return $next($request);}
        return $request->expectsJson()
            ? response([
                'message' => 'Access Denied.',
            ], 403)
            : abort(403, 'Access Denied.');
    }}
```

Checkout Resource:

```
<?php
namespace App\Http\Resources;
use App\Models\User;
use App\Http\Resources\UserResource;
use Illuminate\Support\Facades\Auth;
use App\Http\Resources\ProductResource;
use App\Models\Product;
use Illuminate\Http\Resources\Json\JsonResource;

class CheckOutResource extends JsonResource
{
    private $paymentNames = [
        0 => 'Paypal',
        1 => 'Cash',
        2 => 'Bank',
    ];
    public function toArray($request)
    {
        $paymentName = $this->paymentNames[$this->payment];
        $user = User::find(Auth::id());
        return [
            'message' => 'Order Place Successfully',
            'id' =>  $this->id,
            'total' => $this->total,
            'address' => $this->address,
            'payment' => [
                'code' => $this->payment,
                'name' => $paymentName,
            ],
            'company' => $this->company,
            'first_name' => $this->first_name,
```

```
            'last_name' => $this->last_name,
            'phone' => $this->phone,
            'subtotal' => $this->subtotal,
            'shipingCost' => $this->shipingCost,
            'user' => [
                'id' =>  $user->id,
                'email' => $user->email,
            ],
        ];
    }
}
```

Product Resource:

```
<?php
namespace App\Http\Resources;
use App\Http\Resources\ReviewResource;
use Illuminate\Http\Resources\Json\JsonResource;

class ProductResource extends JsonResource
{
    /**
     * The "data" wrapper that should be applied.
     *
     * @var string|null
     */
    public static $wrap = null;

    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'id' =>  $this->id,
            'title' => $this->title,
            'newPrice' => $this->newPrice,
            'oldPrice' => $this->oldPrice,
```

```
            'offer' => $this->offer,
            'reviews_count' => $this->reviews_count,
            'rating' => $this->rating,
            'is_complete' => $this->is_complete,
            'is_special' => $this->is_special,


            'details' => [
                'abstract' => $this->abstract,
                'pin_code' => $this->pin_code,
                'description' => $this->description,
                'videos' => [$this->videos],
                'video_html' => $this->when($this->videos, function () {
                    $query_str = parse_url($this->videos, PHP_URL_QUERY);

                    if (! $query_str) {
                        return null;
                    }

                    parse_str($query_str, $params);

                    if (! isset($params['v'])) {
                        return null;
                    }

                    return '<iframe width="560" height="315"
src="https://www.youtube.com/embed/' . $params['v'] . '" title="YouTube video
player" frameborder="0" allow="accelerometer; autoplay; clipboard-write;
encrypted-media; gyroscope; picture-in-picture; web-share"
allowfullscreen></iframe>';
                }, null),

                'colors' => $this->when($this->color, function () {
                    return collect(explode(',', $this->color))
                        ->filter(fn ($item) => trim($item))
                        ->map(fn ($item) => ucfirst(trim($item)))
                        ->values();
                }, null),

                'featuers' => $this->when($this->featuer, function () {
                    return collect(explode(',', $this->featuer))
                        ->filter(fn ($item) => trim($item))
                        ->map(fn ($item) => ucfirst(trim($item)))
                        ->values();
                }, null),
```

```
                'category' => $this->when($this->category, function () {
                    return collect(explode(',', $this->category))
                        ->filter(fn ($item) => trim($item))
                        ->map(fn ($item) => ucfirst(trim($item)))
                        ->values();
                }, null),
                'sizes' => $this->when($this->size, function () {
                    return collect(explode(',', $this->size))
                        ->filter(fn ($item) => trim($item))
                        ->map(fn ($item) => strtoupper(trim($item)))
                        ->values();
                }, null),
                'images' => $this->whenLoaded('media', function () {
                    $images = [];

                    if ($this->hasMedia('image')) {
                        $images[] =[

                            $this?->getFirstMediaUrl('image')
                        ];
                        for ($i = 2; $i <= 6; $i++) {
                            $media = $this?->getFirstMedia('images', ['form_key'
=> 'image_' . $i]);

                            if (! $media) {
                                continue;
                            }
                            $images[] = $media?->getUrl();
                        }
                    }

                    return $images;
                }),
                'reviews' => ReviewResource::collection($this-
>whenLoaded('reviews')),
            ],

        ];
    }
}
```

Review Resource:

```php
<?php
namespace App\Http\Resources;
use App\Http\Resources\UserResource;
use Illuminate\Http\Resources\Json\JsonResource;

class ReviewResource extends JsonResource
{
    /**
     * The "data" wrapper that should be applied.
     *
     * @var string|null
     */
    public static $wrap = null;

    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'id' =>  $this->id,
            'star' => (int) $this->star,
            'review' => $this->review,
            'user' => new UserResource($this->whenLoaded('user')),
            'product' => $this->whenLoaded('product'),
        ];
    }
}
```

Order Model:

```php
<?php
namespace App\Models;
use App\Models\User;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use App\Models\Cart;

class Order extends Model
{

    protected $fillable = [
        'total',
        'address',
        'company',
        'payment',
        'user_id',
        'first_name',
        'last_name',
        'phone',
        'subtotal',
        'shipingCost',
        'paymentNames',
    ];

    public function user()
    {
        return $this->belongsTo(User::class);
    }
    public function product()
    {
        return $this->belongsTo(Product::class);
    }
    public function cart()
{
    return $this->belongsTo(Cart::class);
}
}
```

Product Model:

```php
<?php
namespace App\Models;
use App\Models\Review;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Spatie\Image\Manipulations;
use Spatie\MediaLibrary\HasMedia;
use Spatie\MediaLibrary\InteractsWithMedia;
use Spatie\MediaLibrary\MediaCollections\File;
use Spatie\MediaLibrary\MediaCollections\Models\Media;

class Product extends Model implements HasMedia
{
    use InteractsWithMedia;

    protected $fillable = [
        'title','newPrice','oldPrice','offer','category','color','size'  ,
'abstract','featuer','pin_code','description','videos','is_complete','is_special'
,'user_id'
    ];

    public function reviews()
    {
        return $this->hasMany(Review::class);
    }
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function scopeRating($query)
    {
        $query->addSelect(\DB::raw('(
            SELECT sum(reviews.star) / count(*) FROM reviews where
reviews.product_id = products.id LIMIT 1
        ) AS rating'));
    }

    /**
     * Returns a list of the allowed files to be uploaded.
     *
     * @return array
     */
```

```
    public static function formRequestFileKeys(): array
    {
        return [
            'image' => [
                'image_1',
            ],
            'images' => [
                'image_2',
                'image_3',
                'image_4',
                'image_5',
                'image_6',
            ],
        ];
    }

    public function registerMediaCollections(): void
    {
        $this
            ->addMediaCollection('images')
            ->acceptsFile(function (File $file) {
                return in_array($file->mimeType, config('media-
library.allowed_image_types'));
            })
            ->useFallbackUrl(asset('assets/img/no-image.jpg'));

        $this
            ->addMediaCollection('image')
            ->singleFile()
            ->acceptsFile(function (File $file) {
                return in_array($file->mimeType, config('media-
library.allowed_image_types'));
            })
            ->useFallbackUrl(asset('assets/img/no-image.jpg'));
    }
    public function registerMediaConversions(Media $media = null): void
    {
        foreach (['50', '200'] as $size) {
            $this->addMediaConversion($size)
                ->performOnCollections('images', 'image')
                ->format(Manipulations::FORMAT_JPG)
                ->quality(90)
                ->fit(Manipulations::FIT_CROP, $size, $size)
                ->optimize()
                ->{$size == 200 ? 'nonQueued' : 'queued'}();
```

```
        }

        $this->addMediaConversion('large')
            ->performOnCollections('images', 'image')
            ->format(Manipulations::FORMAT_JPG)
            ->quality(90)
            // ->width(1200)
            ->fit(Manipulations::FIT_CROP, 1200, ceil(1200 / 16 * 9))
            ->optimize();
        $this->addMediaConversion('medium')
            ->performOnCollections('images', 'image')
            ->format(Manipulations::FORMAT_JPG)
            ->quality(90)
            ->width(640)
            // ->fit(Manipulations::FIT_CROP, 640, ceil(640/16*9))
            ->optimize();
        $this->addMediaConversion('small')
            ->performOnCollections('images', 'image')
            ->format(Manipulations::FORMAT_JPG)
            ->quality(90)
            ->fit(Manipulations::FIT_CROP, 360, ceil(360 / 4 * 3))
            // ->width(360)
            ->optimize();
    }
    public function cart()
    {
        return $this->hasOne(Cart::class);
    }

    public function wishlist()
    {
        return $this->hasOne(WishList::class);
    }

    public function order()
    {
        return $this->hasOne(Order::class);
    }

    public function orderItem()
    {
        return $this->hasOne(OrderItem::class);
    }
}
```

Review Model:

```php
<?php
namespace App\Models;
use App\Models\Product;
use App\Models\User;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Review extends Model
{
    use HasFactory;

    protected $fillable = [
        'review',
        'star',
        'product_id',
        'user_id',
    ];

    public function product()
    {
        return $this->belongsTo(Product::class);
    }

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

User Model:

```php
<?php
namespace App\Models;
use App\Models\Order;
use App\Models\Review;
use Ichtrojan\Otp\Models\Otp;
use Laravel\Sanctum\HasApiTokens;
use Tymon\JWTAuth\Contracts\JWTSubject;
use Illuminate\Notifications\Notifiable;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable implements JWTSubject
{
    use HasApiTokens;
    use HasFactory;
    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'type',
        'first_name',
        'last_name',
        'email',
        'phone',
        'password',
        'email_verified_at',
        'varifed',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
        'oauth_provider',
        'oauth_id',
```

```
];

/**
 * The attributes that should be cast.
 *
 * @var array<string, string>
 */
protected $casts = [
    'email_verified_at' => 'datetime',
    'type' => 'integer',
];

protected $appends = [
    'type_text'
];

public function getJWTIdentifier()
{
    return $this->getKey();
}

public function getJWTCustomClaims()
{
    return [];
}

public function getTypeTextAttribute(): string
{
    return match ($this->attributes['type']) {
        1       => 'Seller',
        2       => 'Admin',
        default => 'Customer',
    };
}
public function reviews()
{
    return $this->hasMany(Review::class);
}
public function products()
{
    return $this->hasMany(Product::class);
}
public function cart()
{
    return $this->hasMany(Cart::class);
```

```
    }

    public function wishlist()
    {
        return $this->hasMany(WishList::class);
    }

    public function order()
    {
        return $this->hasMany(Order::class);
    }
}
```

Mail sends to Admin after Checkout:

```
<?php
namespace App\Mail;
use App\Models\Order;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;

class AdminOrderPlaced extends Mailable
{
    use Queueable, SerializesModels;

    public $order;
    public $cart;

    /**
     * Create a new message instance.
     *
     * @param Order $order
     */
    public function __construct(Order $order, $cart)
```

```
    {
        $this->order = $order;
        $this->cart = $cart;

    }

    /**
     * Get the message envelope.
     *
     * @return \Illuminate\Mail\Mailables\Envelope
     */
    public function build()
    {
        return $this->subject('Admin Order Placed')
                    ->view('Mails.admin_order_placed')
                    ->with([
                        'order' => $this->order,
                        'cart' => $this->cart,

                    ]);
    }

    /**
     * Get the attachments for the message.
     *
     * @return array
     */
    public function attachments()
    {
        return [];
    }
}
```

Email format:

```
Dear Admin,<br>
A new order has been placed by {{ $order->first_name }} {{ $order->last_name }}
with {{ $order->user->email }} with the following details:<br>
@foreach ($cart as $item)
    Product Name: {{ $item->product->title }}<br>
    Price: {{ $item->product->newPrice }}$<br>
    Quantity: {{ $item->quantity }}<br>
    Color: {{ $item->product->color }}<br>
    Size: {{ $item->product->size }}<br>
    Category: {{ $item->product->category }}<br>
@endforeach
Total price: {{ $order->total }}$ EGP <br>
Payment Method: {{ $order->payment == 0 ? 'Paypal' : ($order->payment == 1 ?
'Cash' : 'Bank') }}<br>Address: {{ $order->address }}<br>Thank you for your
attention.
Best regards
```

Mail send to Customer after Checkout:

```
<?php
namespace App\Mail;
use App\Models\Order;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;

class OrderPlaced extends Mailable
{
    use Queueable, SerializesModels;

    public $order;
    public $total;
    public $cart;
    /**
     * Create a new message instance.
     * @param Order $order
```

```php
     */
    public function __construct(Order $order, $cart)
    {
        $this->order = $order;
        $this->total = $order->total;
        $this->cart = $cart;
    }

    /**
     * Get the message envelope.
     *
     * @return \Illuminate\Mail\Mailables\Envelope
     */
    public function envelope()
    {
        return new Envelope(
            subject: 'Order Placed', );}
    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this->view('Mails.OrderPlaced');
    }}
```

Email format:

```
Dear {{ $order->first_name }},<br>
@foreach ($cart as $item)
    Product Name: {{ $item->product->title }}<br>
    Price: {{ $item->product->newPrice }}$<br>
@endforeach
Total price: {{ $order->total }}$ EGP <br>
Payment Method: {{ $order->payment == 0 ? 'Paypal' : ($order->payment == 1 ?
'Cash' : 'Bank') }}<br>Address: {{ $order->address }}<br>Address: {{ $order-
>address }}<br>
Phone: {{ $order->phone }}<br>
Shiping Cost: {{ $order->shipingCost }}$<br>

Thanks again for your order!
```

Routes:

```php
<?php
use App\Http\Controllers\Api\AuthController;
use App\Http\Controllers\Auth\EmailVerificationController;
use App\Http\Controllers\CartController;
use App\Http\Controllers\CheckOutController;
use App\Http\Controllers\ContactController;
use App\Http\Controllers\WishListController;
use App\Http\Controllers\ProductController;
use App\Http\Controllers\OrderToFinishController;
use Spatie\MediaLibrary\MediaCollections\Models\Media;
use App\Http\Controllers\ReviewController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});
Route::group([
    'middleware' => 'api',
    'prefix' => 'auth'
], function ($router) {
    Route::post('/login', [AuthController::class, 'login']);
    Route::post('/register', [AuthController::class, 'register']);
    Route::post('/logout', [AuthController::class, 'logout']);
    Route::post('/updateAccount', [AuthController::class, 'update']);


    //EmailVerification
    Route::post('/email_verification', [EmailVerificationController::class,
'email_verification']);
    Route::get('/sendEmailVerification', [EmailVerificationController::class,
'sendEmailVerification']);

});

##############################################################################

Route::middleware('auth')->post('/reviews', [ReviewController::class, 'store']);
Route::middleware('auth')->delete('/delete/{id}', [ReviewController::class,
'delete']);

##############################################################################
Route::group([
```

```
    'prefix' => 'seller',
         'middleware' => ['api', 'auth', 'is.seller'],
     ], function () {
        Route::post('/products', [ProductController::class, 'store']);
        Route::post('/products/{id}', [ProductController::class, 'update']);
        Route::delete('/products/{id}', [ProductController::class, 'delete']);
        Route::get('/{id}/products', [ProductController::class,
'sellerProducts']);
        Route::get('/{id}/products', [ProductController::class,
'sellerProducts']);
        Route::delete('/media/{id}', function (Request $request, int $id) {
        $media = Media::findOrFail($id);
        $media->delete();

    return [
        'message' => __('The file has been deleted successfully.')
    ];
})->name('api.media.delete');
    });
##############################################################################
Route::group([
    'prefix' => 'customer',
         'middleware' => ['api', 'auth', 'is.customer'],
     ], function () {

        Route::post('/cart', [CartController::class, 'store']);
        Route::get('/cart', [CartController::class, 'index']);
        Route::delete('/cart/{id}', [CartController::class, 'delete']);
##############################################################################
        Route::post('/wishlist', [WishListController::class, 'store']);
        Route::get('/wishlist', [WishListController::class, 'index']);
        Route::delete('/wishlist/{id}', [WishListController::class, 'delete']);
        Route::post('/orders', [CheckOutController::class, 'checkout']);
    });
##############################################################################
Route::get('/products', [ProductController::class, 'index']);
Route::get('/products/{id}', [ProductController::class, 'show']);
Route::get('/products/{id}/relatedProducts', [ProductController::class,
'relatedProducts']);

//Mail Route
Route::post('/contact', [ContactController::class, 'submit']);
Route::post('/ordertofinish', [OrderToFinishController::class, 'send']);
```

Cart Controller:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Resources\CartResource;
use App\Models\Cart;
use App\Models\Product;
use App\Models\Shop;
use App\Traits\MediaTrait;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Validator;

class CartController extends Controller
{
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'product_id'    => 'required|exists:products,id',
            'quantity'    => 'required|integer|min:1',
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        Auth::user()->cart()->updateOrCreate(
            [
                'product_id' => $request->product_id,
            ],
            $validator->validated()
        );

        $cartItems = Auth::user()->cart()
            ->with('product', 'product.media')
            ->get();

        return CartResource::collection($cartItems);
    }

    public function index()
    {
        $cartItems = Auth::user()->cart()
```

```
            ->with('product', 'product.media')
            ->get();

        return CartResource::collection($cartItems);
    }

    public function delete($id)
    {
        $cart = Auth::user()->cart()->findOrFail($id);
        $cart->delete($id);
        return response()->json('deleted');
    }
}
```

Cart Resource:

```
<?php

namespace App\Http\Resources;

use App\Http\Resources\ProductResource;
use App\Http\Resources\UserResource;
use Illuminate\Http\Resources\Json\JsonResource;

class CartResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */

    public function toArray($request)
    {
        return [
            'id' =>  $this->id,
```

```
            'quantity' => $this->quantity,
            'product' => new ProductResource($this->whenLoaded('product')),
            'user' => new UserResource($this->whenLoaded('user')),
        ];
    }
}
```

Cart Model

```
<?php

namespace App\Models;

use App\Models\Product;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Cart extends Model
{
    protected $fillable = [
        'title','newPrice','oldPrice','quantity','product_id',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'quantity' => 'integer',
    ];

    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function product()
    {
        return $this->belongsTo(Product::class);
    }
}
```

Wishlist Controller:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Resources\WishListResource;
use App\Models\WishList;
use App\Models\Product;
use App\Models\Shop;
use App\Traits\MediaTrait;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Validator;

class WishListController extends Controller
{
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'product_id'    => 'required|exists:products,id',
            //'quantity'    => 'required|integer|min:1',
        ]);

        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        Auth::user()->wishlist()->updateOrCreate(
            [
                'product_id' => $request->product_id,
            ],
            $validator->validated()
        );

        $wishlistItems = Auth::user()->wishlist()
            ->with('product', 'product.media')
            ->get();

        return WishListResource::collection($wishlistItems);
    }

    public function index()
    {
        $wishlistItems = Auth::user()->wishlist()
```

```
            ->with('product', 'product.media')
            ->get();
        return WishListResource::collection($wishlistItems);
    }

    public function delete($id)
    {
        $wishlist = Auth::user()->wishlist()->findOrFail($id);
        $wishlist->delete($id);
        return response()->json('deleted');
    }
}
```

Wishlist Resource:

```
<?php

namespace App\Http\Resources;

use App\Http\Resources\ProductResource;
use App\Http\Resources\UserResource;
use Illuminate\Http\Resources\Json\JsonResource;

class WishListResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */

    public function toArray($request)
    {
        return [
            'id' =>  $this->id,
            //'quantity' => $this->quantity,
            'product' => new ProductResource($this->whenLoaded('product')),
            'user' => new UserResource($this->whenLoaded('user')),
        ];}}
```

Wishlist Model:

```php
<?php

namespace App\Models;

use App\Models\Product;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class WishList extends Model
{
    protected $fillable = [
        'title','newPrice','oldPrice','product_id',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */

    /*
    protected $casts = [
        'quantity' => 'integer',
    ];
    */
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function product()
    {
        return $this->belongsTo(Product::class);
    }
}
```

Contact us Controller:

```
<?php

namespace App\Http\Controllers;

use App\Mail\contact as MailContact;
use App\Models\Contact;
use Illuminate\Http\Request;
use Mail;

class ContactController extends Controller
{
    public function submit(Request $request)
     {
        //store data in database
        Contact::create($request->all());


        // Send mail to Application Admin
        Mail::to("homebazzar.technology@gmail.com")->send(new MailContact(
$request->name, $request->phone, $request->email, $request->subject, $request-
>message));

        return response()->json(['success' => 'we have recieved your message and
would like to thank you for writing to us.']);


      }

}
```

Contact us Request:

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class ContactRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, mixed>
     */
    public function rules()
    {
        return [
            'name' => ['required'],
            'email' => ['required', 'email'],
            'message' => ['required'],
        ];
    }
}
```

Contact us mail:

```php
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;

class contact extends Mailable
{
    use Queueable, SerializesModels;

    public $name;
    public $phone;
    public $email;
    public $message;
    public $subject;

    /**
     * Create a new message instance.
     *
     * @return void
     */


    public function __construct($name, $phone, $email, $message, $subject)
    {
       $this->name = $name;
       $this->phone = $phone;
       $this->email = $email;
       $this->message = $message;
       $this->subject = $subject;
    }

    /**
     * Get the message envelope.
     *
     * @return \Illuminate\Mail\Mailables\Envelope
     */
    public function envelope()
```

```
    {
        return new Envelope(
            subject: 'contactus',
        );
    }

    public function build()
    {
        return $this->markdown('Mails.contactus');
    }

    /**
     * Get the message content definition.
     *
     * @return \Illuminate\Mail\Mailables\Content
     */

    // public function content()
    // {
    //     return new Content(
    //         view: 'view.name',
    //     );
    // }


    /**
     * Get the attachments for the message.
     *
     * @return array
     */
    public function attachments()
    {
        return [];
    }
}
```

Contact us Mail Format:

```
<h1> Hello </h1> <br><br>


You have got an email from: {{$name}}  <br><br>

User Details: <br><br>

Name:    {{$name}}        <br>
Email:   {{$email}}       <br>
Phone:   {{$phone}}       <br>
subject: {{$subject}}     <br>
message: {{$message}}     <br>

Thanks
```

Contact Model:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Contact extends Model
{
    use HasFactory;
    //protected $primaryKey = '$id';
    public $table = 'contacts';
    public $fillable = ['name', 'phone','email', 'subject', 'message'];
    //protected $timestamps = true;


    // public function user()
    // {
    //     return $this->belongsTo(User::class);
    // }
}
```

Order to finish Controller:

```
<?php

namespace App\Http\Controllers;

use App\Mail\CustomerMail;
use App\Mail\ordertofinish as Mailordertofinish;
use App\Models\OrderToFinish;
use App\Models\Product;
use Illuminate\Http\Request;
use Mail;

class OrderToFinishController extends Controller
{
    public function send(Request $request)
     {
        $product =Product::find($request->product_id);
       // $productPrice = Product::find($product->newprice);
       // dd($product);

       $offer = 10;
       if($product){
        OrderToFinish::create($request->all());

        $product->newprice = $product->oldPrice - ($offer / 100) * $product-
>oldPrice;

        $product->save();

        //  Send mail to Application Admin
        Mail::to("homebazzar.technology@gmail.com")->send(new Mailordertofinish(
$request->name, $request->product_id, $request->email, $request->address,
$request->phone_number, $request->description));

        //  send mail from admin to customer
        Mail::to($request->email)->send(new CustomerMail( $request->name,
$request->email, $request->address, $request->phone_number, $request-
>description, $product->newPrice));
        return response()->json(['success' => 'we have recieved your message and
would like to thank you for writing to us.']);
    }
    else{
        return response()->json(['Error' => 'Please send your order Again']);
    }}}
```

Order to finish Mail:

```php
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;

class ordertofinish extends Mailable
{
    use Queueable, SerializesModels;

    public $name;
    public $product_id;
    public $email;
    public $address;
    public $phone_number;
    public $description;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct($name, $product_id, $email, $address,
$phone_number, $description)
    {
        $this->name = $name;
        $this->product_id = $product_id;
        $this->email = $email;
        $this->address = $address;
        $this->phone_number = $phone_number;
        $this->description = $description;
    }



    /**
     * Get the message envelope.
     *
```

```
     * @return \Illuminate\Mail\Mailables\Envelope
     */
    public function envelope()
    {
        return new Envelope(
            subject: 'ordertofinish',
        );
    }

    public function build()
    {
        return $this->markdown('Mails.ordertofinish');
    }

    /**
     * Get the message content definition.
     *
     * @return \Illuminate\Mail\Mailables\Content
     */
    // public function content()
    // {
    //     return new Content(
    //         view: 'view.name',
    //     );
    // }

    /**
     * Get the attachments for the message.
     *
     * @return array
     */
    public function attachments()
    {
        return [];
    }
}
```

Order to finish mail format:

```
<h1> Order Details </h1> <br><br>

You have got an email from: {{$name}}  <br><br>

User Details: <br><br>

Name:           {{$name}}        <br>
Product id:     {{$product_id}}        <br>
Email:          {{$email}}       <br>
Address:        {{$address}}       <br>
Phone:          {{$phone_number}}     <br>
Description:    {{$description}}     <br>

Thanks.
```

Order to finish Request:

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class OrderToFinishRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
```

```
     * Get the validation rules that apply to the request.
     *
     * @return array<string, mixed>
     */
    public function rules()
    {
        return [
            'name' => ['required'],
            'product_id' => ['required'],
            'email' => ['required', 'email'],
            'address' => ['required'],
            'phone_number' => ['required'],
            'description' => ['required'],

        ];
    }
}
```

Order to finish model:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class OrderToFinish extends Model
{
    use HasFactory;
    public $table = 'ordertofinish';
    public $fillable = ['name', 'product_id', 'email','address', 'phone_number',
'description'];

    public function product()
    {
        return $this->belongsTo(Product::class);
    }}
```

Customer Mail:

```php
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;
use App\Models\Product;

class CustomerMail extends Mailable
{
    use Queueable, SerializesModels;

    public $name;
    public $email;
    public $address;
    public $phone_number;
    public $description;
    public $productprice;
    public $deliverytime = '10-15 business days';


    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct($name, $email, $address, $phone_number,
$description,  $productprice)
    {
        $this->name = $name;
        $this->email = $email;
        $this->address = $address;
        $this->phone_number = $phone_number;
        $this->description = $description;
        $this->productprice = $productprice;
        //$this->deliveryTime = $deliveryTime;
    }

    /**
```

```php
     * Get the message envelope.
     *
     * @return \Illuminate\Mail\Mailables\Envelope
     */
    public function envelope()
    {
        return new Envelope(
            subject: 'CustomerMail',
        );
    }

    /**
     * Get the message content definition.
     *
     * @return \Illuminate\Mail\Mailables\Content
     */
    public function build()
    {
        return $this->markdown('Mails.CustomerMail');
    }


    /**
     * Get the attachments for the message.
     *
     * @return array
     */
    public function attachments()
    {
        return [];
    }
}
```

Customer Mail format:

```
<h1> Order Details </h1> <br><br>
Dear {{$name}},

Thank you for your recent purchase! We wanted to confirm the details of your
order:<br><br>

Email: {{$email}}    <br>
Address: {{$address}} <br>
Phone: {{$phone_number}} <br>
Description: {{$description}} <br>
Product Price: {{$productprice}} <br>
Delivery Time: {{$deliverytime}} <br><br>
If you have any questions or concerns, please do not hesitate to contact us.<br>

Thank you for your business!<br>

Best regards.<br>

Company Name: Home Bazzar.
```

Email Verification Controller:

```
<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use App\Http\Requests\Auth\EmailVerificationRequest;
use App\Models\User;
use Illuminate\Http\Request;
use App\Notifications\EmailVerificationNotification;
use Otp;

class EmailVerificationController extends Controller
{
```

```
    private $otp;
    public function __construct()
    {
           $this->otp = new Otp;
    }



    public function sendEmailVerification (Request $request)
    {
       $request->user()->notify(new EmailVerificationNotification());
       $success['The new code has been sent to your Gmail, please go and check
it'] = true;
       return response()->json($success, 200);
    }


    public function email_verification(EmailVerificationRequest $request)
    {

        $otp2 = $this->otp->validate($request->email, $request->otp);
       // dd($otp2);
        if(!$otp2->status){
            return response()->json(['error' => $otp2], 401);
        }

        $user = User::where('email', $request->email)->first();
         $user->update(['email_verified_at' =>now(), 'varifed'=> 1]);
        $success['email has been verified'] = true;
        return response()->json($success, 200);

    }



}
```

Email Verification Request:

```php
<?php

namespace App\Http\Requests\Auth;

use Illuminate\Foundation\Http\FormRequest;

class EmailVerificationRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, mixed>
     */
    public function rules()
    {
        return [
            'email' => ['required', 'email', 'exists:users'],
            'otp' => ['required', 'max:6'],

        ];
    }
}
```

Email Verification Notification:

```php
<?php

namespace App\Notifications;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Notifications\Messages\MailMessage;
use Illuminate\Notifications\Notification;
use Otp;

class EmailVerificationNotification extends Notification
{
    use Queueable;

    public $message;
    public $subject;
    public $fromEmail;
    public $mailer;
    private $otp;


    /**
     * Create a new notification instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->message = 'Use the below code for verification process';
        $this->subject = 'Verification Needed';
        $this->fromEmail = 'rowida001@gmail.com';
        $this->mailer = 'smtp';
        $this->otp = new Otp;

    }

    /**
     * Get the notification's delivery channels.
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function via($notifiable)
```

```
    {
        return ['mail'];
    }

    /**
     * Get the mail representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return \Illuminate\Notifications\Messages\MailMessage
     */
    public function toMail($notifiable)
    {
        $otp = $this->otp->generate($notifiable->email,6, 60);
        return (new MailMessage)
        ->mailer('smtp')
        ->subject($this->subject)
        ->greeting('Hello'.$notifiable->first_name)
        ->line($this->message)
        ->line('code: '. $otp->token);

    }

    /**
     * Get the array representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function toArray($notifiable)
    {
        return [
            //
        ];
    }
}
```

CHAPTER 6

Conclusion & Future Work

Conclusion:

Our Furniture Online Store system is a comprehensive e-commerce solution that includes both a website and a mobile application. This system is structured around three primary categories: sellers, management, and clients. The website and application offer a user-friendly, intuitive interface that allows clients to browse and purchase furniture products with ease. The sellers' platform enables them to upload product information, manage inventory, and track orders. Meanwhile, the management platform provides administrators with tools for overseeing inventory, analyzing sales data, and managing the website and application. By combining these elements into a single, integrated system, the Furniture Online Store project aims to provide a seamless, end-to-end experience for all users.

Future Work:

- **Implement a recommendation system:** A recommendation system can help improve the customer experience and increase sales by suggesting products that are relevant to the customer's interests and preferences. Consider implementing a collaborative filtering algorithm or a content-based filtering algorithm to make personalized recommendations based on the customer's browsing and purchase history.
- **Add 3D images:** 3D images can provide customers with a more immersive and realistic experience of the product, which can help to increase customer confidence and reduce the likelihood of returns. Consider implementing 3D images for some or all of the products, using tools like SketchUp or Blender.
- **Expand the product range:** Consider expanding the product range to include more categories and product types, such as lighting, rugs, or home decor. This can help to attract new customers and increase the average order value.
- **Implement augmented reality (AR):** Augmented reality can allow customers to visualize how a product will look in their home before making a purchase. Consider implementing an AR feature using tools like ARKit or ARCore to provide customers with a more immersive and interactive experience.
- **Improve the website's search functionality:** Improving the website's search functionality can help customers find the products they are looking for more easily and quickly. Consider implementing a search algorithm that takes into account spelling mistakes, synonyms, and related keywords.
- **Enhance the mobile shopping experience:** With more and more customers shopping on their mobile devices, it's essential to optimize the website for mobile use. Consider implementing a responsive design, improving load times, or simplifying the navigation menu for smaller screens.
- **Implement data analytics:** Data analytics can provide valuable insights into customer behavior and preferences, which can be used to optimize the website and marketing strategies. Consider implementing tools like Google Analytics or conducting surveys to gather feedback from customers.

References:

- https://youtube.com/playlist?list=PLDoPjvoNmBAy41u35AqJUrI-H83DObUDq
- https://youtube.com/playlist?list=PLftLUHfDSiZ4GfPZxaFDsA7ejUzD7SpWa
- https://youtube.com/playlist?list=PLftLUHfDSiZ6MfN8UhhcXDhh64eejvIKK
- https://youtube.com/playlist?list=PLftLUHfDSiZ5LAQuaKUUpN8F_dvKTPEtc
- https://youtube.com/playlist?list=PLCm7ZeRfGSP5e07XG-waxCb2kLq7M4J5m
- https://youtube.com/playlist?list=PLvNu8E-aj20kDsDL5-8LCYlPc8w934sU-
- https://youtu.be/Tm2pDr1gQaI
- https://youtu.be/TvFMI8rhQpA
- https://youtube.com/playlist?list=PLRheCL1cXHruG6bV4tAIF4AhkUMaABf3F
- draw.io (diagrams.net)
- https://www.udemy.com/course/complete-flutter-arabic/learn/lecture/25709104
- https://www.udemy.com/course/fluttercourse/learn/lecture/28730254
- https://www.youtube.com/channel/UCJL1VRChDJm-JpjOzUfIlEw