VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING

INFORMATION TECHNOLOGIES STUDY PROGRAM 3RD YEAR COURSE WORK

Online flower and bouquet shop

Internetinė gėlių ir puokščių parduotuvė

Done by:

Valeriia Rudenko

Supervisor:
dr. Joana Katina

Vilnius
2024

Contents

Abstract

This project involves developing a comprehensive bouquet shop website that allows users to browse and purchase flowers and custom bouquets. The primary objective is to provide a user-friendly platform where customers can register, authenticate, create personalized bouquets, and complete purchases seamlessly. The system includes features like product search, shopping cart, order history, and payment processing, along with an admin dashboard for managing inventory and user data. The result is a responsive and efficient e-commerce site that enhances the flower shopping experience for users. Key functionalities include custom bouquet creation, secure user authentication, and robust order management.

Keywords: E-commerce, Custom Bouquet, User Authentication, Shopping Cart, Responsive Design, Admin Dashboard, React.js, Node.js, MongoDB, PayPal, JWT.

Santrauka

Darbo pavadinimas kita kalba

Šis projektas apima išsamios puokščių parduotuvės svetainės, kurioje naudotojai gali naršyti ir pirkti gėles bei nestandartines puokštes, kūrimą. Pagrindinis tikslas – sukurti patogią platformą, kurioje klientai galėtų registruotis, autentifikuotis, kurti asmenines puokštes ir sklandžiai atlikti pirkimus. Sistema apima tokias funkcijas kaip produktų paieška, pirkinių krepšelis, užsakymų istorija ir mokėjimų apdorojimas, taip pat administratoriaus valdymo skydelį, skirtą inventoriui ir naudotojų duomenims tvarkyti. Rezultatas – jautri ir veiksminga el. parduotuvės svetainė, kuri pagerina naudotojų apsipirkimo gėlėmis patirtį. Pagrindinės funkcijos – pasirinktinų puokščių kūrimas, saugus naudotojo autentifikavimas ir patikimas užsakymų valdymas.

Raktiniai žodžiai: el. parduotuvė, pasirinktinė puokštė, naudotojo autentifikavimas, pirkinių krepšelis, jautrus dizainas, administratoriaus valdymo skydelis, React.js, Node.js, MongoDB, PayPal, JWT.

1 Introduction

In this century, technology and everything related to it are quickly developing. There are new programs for different systems and new web programs for various browsers. This project is a web program, a website, and a convenient online shop for buying flowers, featuring delivery options and various types of payment. It offers a large assortment and the possibility to create a bouquet online, allowing customers to choose flowers and packaging.

This topic was chosen for several reasons. First, there is a big interest in web development, and creating an online store is very interesting. As a frequent buyer of flowers, I have seen many flower shop websites, but none of them had what I needed, particularly the creation of a custom bouquet. Therefore, this topic and this project will bring a lot of benefits to its developers and to those who use it.

The main goal of the project was to create an online flower shop with a light and convenient interface, an adaptive design, and ease of use on devices with small screens. It includes features such as registration, authentication, a large selection of flowers, multiple payment methods using API connections, the ability to create a custom bouquet, and an admin panel with all the necessary functions.

The tasks that were set and performed included the analysis of similar systems, specifically flower websites, setting goals and objectives, dividing functions into separate iterations for easier development, designing, and implementing planned functions, testing to ensure everything works, deploying the system, and creating documentation.

As a result, an online store for buying flowers and bouquets was developed, that works on all modern browsers, has a responsive web design, all the needed features that a usual shop must have, and a custom feature to create and order bouquets made by user, few types of payment, and admin panel for managing orders, users and items in a shop. This system was successfully deployed on the university's infrastructure. And in the end, this document was created.

The structure of the project is as follows: the first part involves the analysis of similar systems; the second part covers the design system and requirements; the third part focuses on the implementation of the website itself, both frontend and backend; the fourth part is about the deployment system; and finally, the project concludes with recommendations for the future system.

2 Similar System Analysis

In this section, were analyzed 3 different similar systems - online shops for buying flowers and bouquets. It contains the advantages and disadvantages of each system, small overview of each system, screenshots and links to them.

2.1 Flowers and Lovers

(https://flowersandlovers.lt)

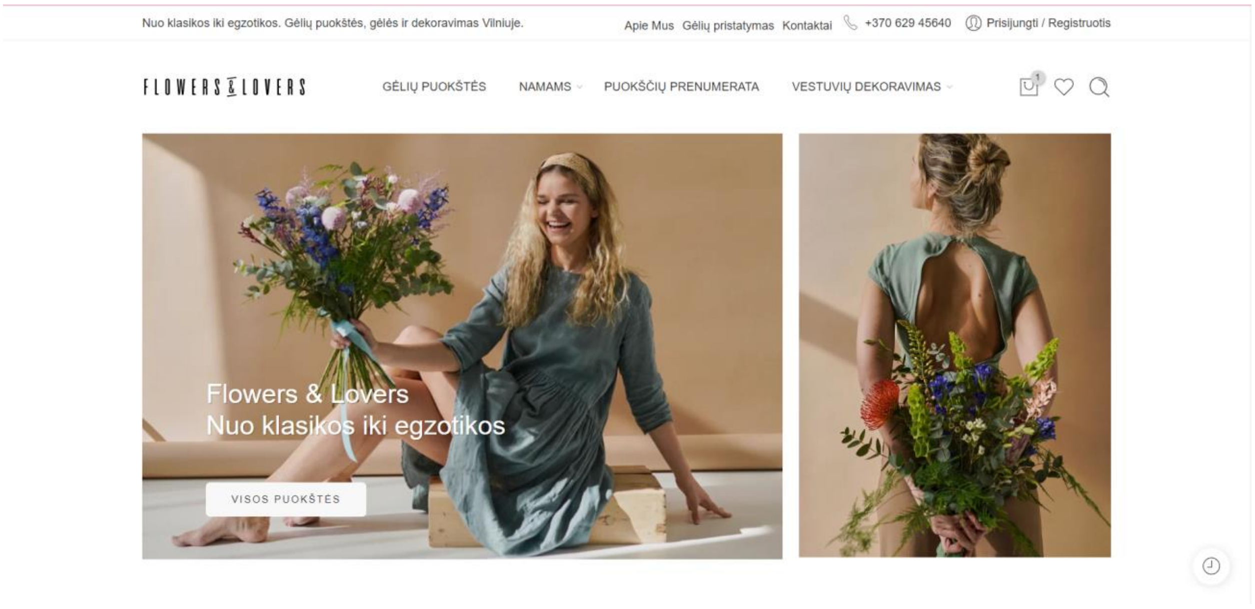

Figure 1. Flowers and Lovers

Overview: The Flowers and Lovers that is on Figure 1 is a well-structured platform for buying flower bouquets. It features a clean design with a focus on high-quality images of products, making it visually appealing and user-friendly. It has easy navigation with categories, and filters like price, color, and type for enhancing the shopping experience. Each item has a description, image, and review, which is a very useful thing.

Functionality: Users can order pre-made bouquets only, as there is no feature to create custom bouquets. Payment can be made via two methods: bank transfer or sending money to a card. There are no options for PayPal or payment upon receipt.

Advantages:

- High-quality visuals and detailed product descriptions.
- Easy-to-navigate interface with useful filtering options.
- Clear call-to-action buttons for purchasing and adding items to the cart.

Disadvantages:

- Limited payment options.
- No functionality for creating custom bouquets.

2.2 Camellia Shop

(https://shop.camellia.ua)

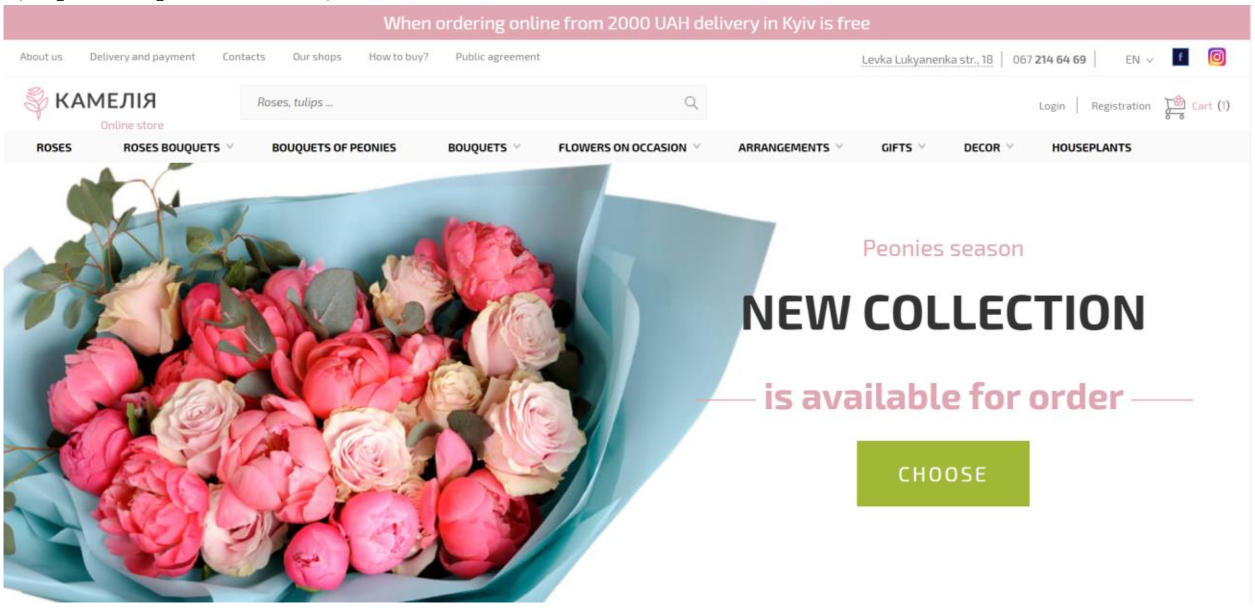

Figure 2. Camellia Shop

Overview: Camellia Shop on Figure 2 is a comprehensive online store that offers a wide range of flowers and related products. It is tailored for both individual and corporate clients. The site has an intuitive interface and search function. As the previous website, it has various categories. Here also included a "blog" with hacks and tips on flower care, which is a very useful functionality that is not hard to implement but brings a lot of benefits.

Functionality: Like Flowers and Lovers, Camellia Shop does not offer an option to create custom bouquets online. Although it is stated that payment can be made by cash or card, the actual checkout process during testing revealed that only card payments are available.

Advantages:

- Extensive product range with detailed descriptions and care tips.
- Blog section for customer engagement and value addition.
- Multiple payment options, including credit cards and bank transfers.

Disadvantages:

- The design is somewhat cluttered, which might overwhelm new users.
- Loading times can be slow, affecting the overall user experience.
- No ability to create custom bouquets.

2.3 Flora VIP Luxe

(https://floravipluxe.com)

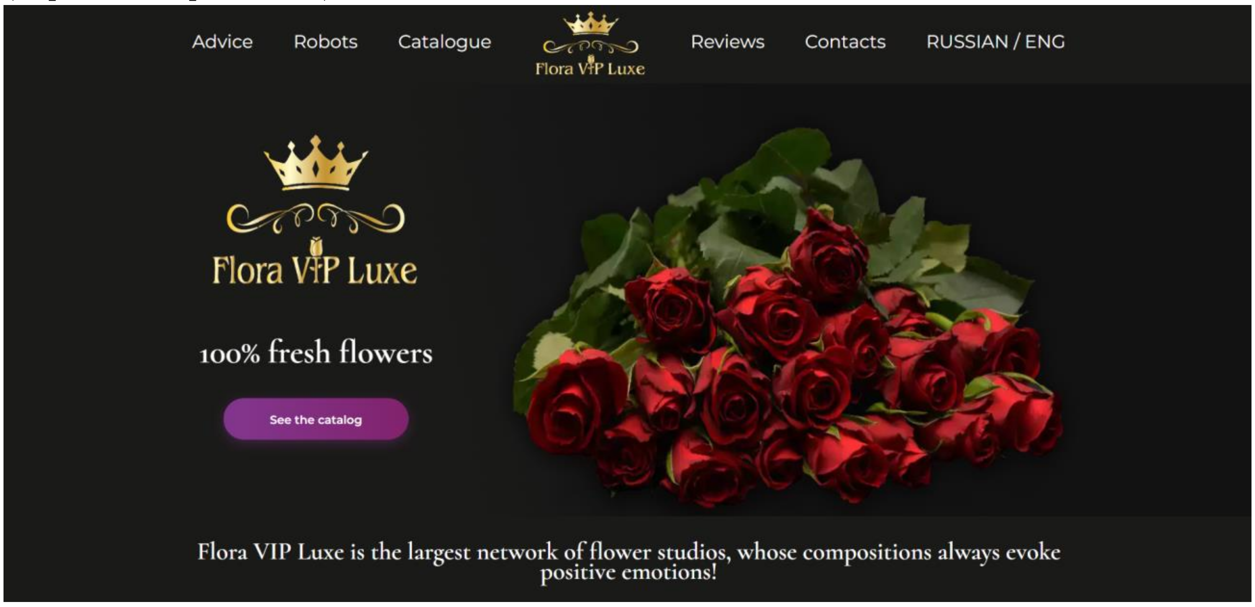

Figure 3. Flora VIP Luxe

Overview: Flora VIP Luxe, Figure 3, targets a high-end market with luxurious flower arrangements and exclusive gifts. The website design reflects its premium positioning. The site offers a sophisticated and elegant design with high-resolution images, but it has only one page and if the user wants to order or just to see the catalog, he has to go to the chat and ask everything there.

Functionality: While Flora VIP Luxe has an elegant online presence, the actual ordering process redirects users to a Telegram bot. This bot handles all communications and customizations for orders, offering a unique but somewhat indirect shopping experience.

Advantages:

- Elegant design with a focus on premium products.
- Effective use of high-resolution images to showcase products.

Disadvantages:

- Higher price point may limit the customer base.
- Limited information on delivery options and policies.
- Users must switch to a Telegram bot for completing orders, which may not be convenient for everyone.

2.4 Summary of the Analysis

Title	User Interface	Payment Options	Custom Bouquets	Loading Speed
Flowers And Lovers	Clean and user-friendly design	Bank transfer, card payment	No	Fast
Camellia Shop	Intuitive but cluttered	Bank transfer, card payment	No	Slow
Flora VIP Luxe	Sophisticated and elegant	Telegram bot	Yes	Slow

Table 1. Comparison table

3 System Design

The creation of the flower bouquet e-commerce system began with extensive research into existing flower shops and their online offerings. The aim was to identify gaps in the market and opportunities for innovation. Key observations included the lack of custom bouquet creation tools and limited payment options, which informed the direction of development.

3.1 Functional Requirements (F)

A set of functional (F) and non-functional (NF) requirements were Functional Requirements (F)

- **F1:** Users can browse a list of available bouquets, individual flowers, and packaging options.
- **F2:** Each product in the list includes a name, image, price, and rating for easy identification.
- **F3:** Users can view detailed information about each product, including the quantity available.
- **F4:** Users can add products to their cart for future ordering.
- **F5:** Users can view the contents of their cart and modify the quantity of items or remove items.
- **F6:** Users can proceed to checkout by providing necessary shipping details.
- **F7:** Users can select a payment method, choosing either PayPal or cash on delivery.
- **F8:** Users can view their order history and check the status of their orders.
- **F9:** Users can leave feedback for products they have purchased.
- **F10:** Users can register or log in to save their data and view order history.
- **F11:** Users can search for products by name.
- **F12:** Users can filter products on the "Create Custom Bouquet" page using buttons to display flowers or packaging.
- **F13:** Users can see the overall rating and the quantity of products available on each product page.
- **F14:** Database data should be populated with generated data using custom functions for quick filling of the database with test or demonstration data.
- **F15:** Users can create and order custom bouquets by selecting flowers and packaging.
- **F16:** Administrators can add, edit, and delete products from the database.
- **F17:** Administrators can view a list of users and their orders.

3.2 Non-Functional Requirements (NF)

- **NF1:** The website must have a responsive design to ensure correct display and convenient use on various devices.

- **NF2: Usability**: The user interface must be intuitive and easy to navigate, ensuring that users can find information and perform actions with minimal effort.

- **NF3: Compatibility**: The website should be compatible with the latest versions of major web browsers, including Chrome, Firefox, Safari, and Edge.

3.3 Explanation of Decisions and Expected Results

4 **Browsing and Filtering Products:**
4.3 **Decision:** Provide an intuitive interface for users to browse and filter products.
4.4 **Reason:** Enhance user experience and ease of finding desired products.
4.5 **Expected Result:** Users can quickly find and view the products they are interested in.
5 **Detailed Product Information:**
5.3 **Decision:** Include comprehensive details for each product.
5.4 **Reason:** Ensure users have all necessary information to make informed purchasing decisions.
5.5 **Expected Result:** Increased user satisfaction and reduced return rates.
6 **Shopping Cart Management:**
6.3 **Decision:** Allow users to manage their shopping cart with ease.
6.4 **Reason:** Improve the shopping experience by enabling easy modifications to the cart.
6.5 **Expected Result:** Higher cart completion rates and increased sales.
7 **Payment Methods:**
7.3 **Decision:** Integrate PayPal and offer cash on delivery.
7.4 **Reason:** Provide secure and familiar payment options to users.
7.5 **Expected Result:** Enhanced trust and convenience, leading to higher conversion rates.
8 **Order Management:**
8.3 **Decision:** Allow users to view order history and status.
8.4 **Reason:** Increase transparency and trust by keeping users informed about their orders.
8.5 **Expected Result:** Improved customer satisfaction and loyalty.
9 **User Authentication:**
9.3 **Decision:** Implement registration and login functionalities.
9.4 **Reason:** Securely manage user data and provide personalized experiences.
9.5 **Expected Result:** Secure user data storage and enhanced user experience.
10 **Search and Filter Functions:**
10.3**Decision:** Enable search and filter options for products.
10.4**Reason:** Improve usability and help users quickly find products.
10.5**Expected Result:** Increased user engagement and satisfaction.
11 **Rating and Stock Availability:**
11.3**Decision:** Display product ratings and stock levels.
11.4**Reason:** Provide users with additional information to make purchase decisions.
11.5**Expected Result:** Increased trust and informed purchasing decisions.
12 **Database Population:**
12.3**Decision:** Use custom functions to populate the database with test data.
12.4**Reason:** Facilitate testing and demonstration of the system.
12.5**Expected Result:** Efficient database setup for development and testing purposes.
13 **Responsive Design:**
13.3**Decision:** Ensure the website is responsive.
13.4**Reason:** Provide a seamless experience across different devices.
13.5**Expected Result:** Broader user reach and improved user experience on mobile devices.
14 **Custom Bouquet Creation:**
14.3**Decision:** Allow users to create and order custom bouquets.
14.4**Reason:** Offer a unique feature that differentiates the site from competitors.
14.5**Expected Result:** Increased user engagement and satisfaction due to personalization options.
15 **Admin Access:**
15.3**Decision:** Grant administrators CRUD access to products, users, and orders.
15.4**Reason:** Ensure effective management of the e-commerce platform.
15.5**Expected Result:** Efficient operations and maintenance of the product catalog and user base.

3.4 Database Schema Design

The database schema was designed to support the application's requirements:

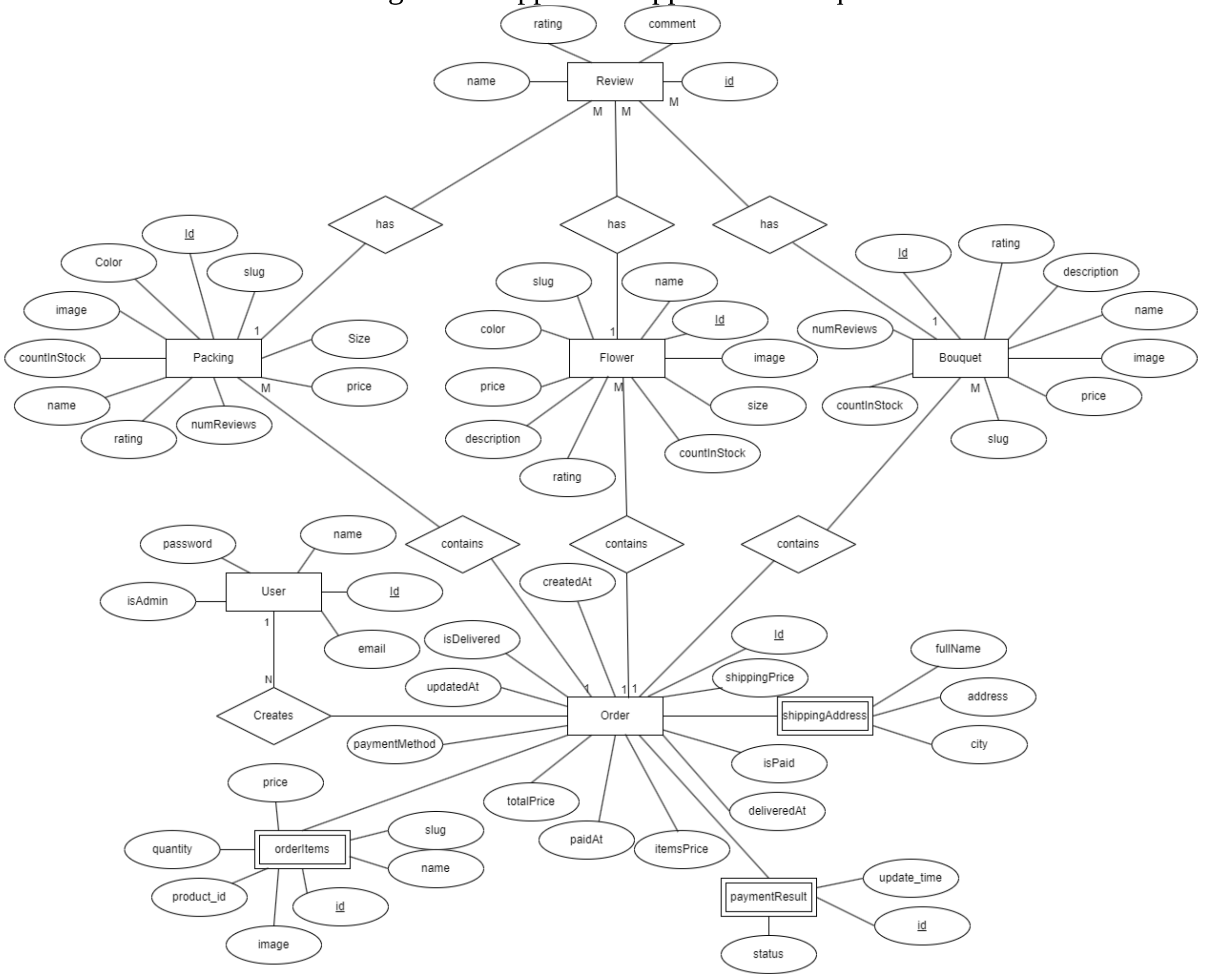

Figure 4. ER diagram

In Figure 4 you can see the ER diagram, and here are explanations of it:

Entities:

1. **User:**

Attributes: id (Primary Key), name, email, isAdmin, password.

2. **Flower:**

Attributes: id (Primary Key), name, slug, color, price, image, size, countInStock, rating, description.

3. **Bouquet:**

Attributes: id (Primary Key), rating, description, name, image, price, slug, countInStock, numReviews.

4. **Packing:**

Attributes: id (Primary Key), price, size, slug, color, image, countInstock, name, rating, numReviews.

5. **Review:**

Attributes: id (Primary Key), comment, rating, name.

6. **Order:**

Attributes: id (Primary Key), createdAt, shippingAdress, isPaid, deliveredAt, itemsPrice, paidAt, totalPrice, updatedAt, isDelivered.

Weak Entities:

1. **paymentResult:**

Attributes: id(Primary key), update_time, status.

2. **orderItems:**

Attributes: id (Primary key), slug, name, image, price, product_id, quantity.

3. **shippingAddress:**

Attributes: city, address, fullName.

Figure 5 illustrates a screenshot from the MongoDB with all created tables.

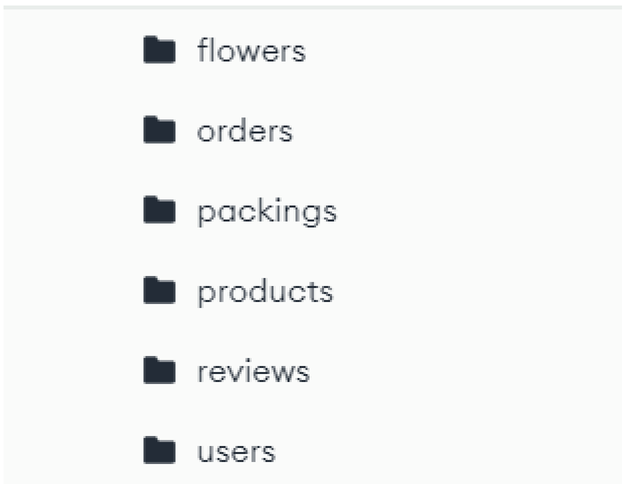
flowers
orders
packings
products
reviews
users

Figure 5. Screenshot of tables from MongoDb

4 System Implementation

The system architecture was designed with modularity in mind, ensuring scalability and maintainability. The key components include Client – Server – Database. The Client was implemented with React, Server with Noed.js and Express, database with MongoDB as shown on Figure 6.

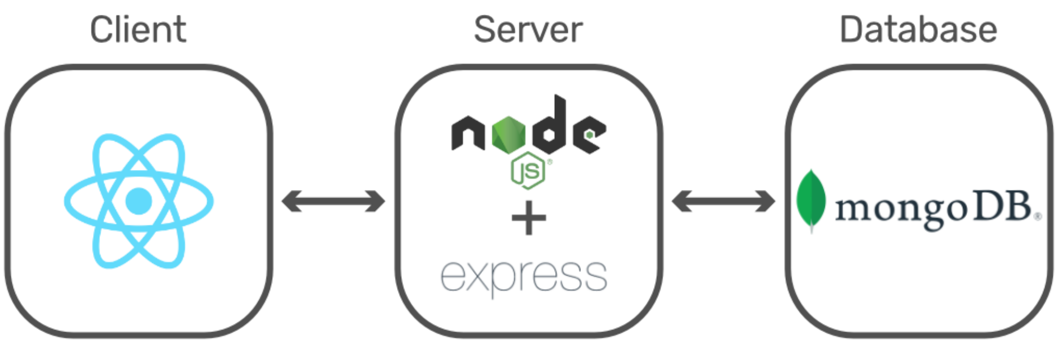

Figure 6. System Architecture

4.1 Key files and methods

The flower shop's application is implemented with a modular architecture, leveraging multiple key files and methods to handle various aspects of its functionality. Here is a detailed description of the key components:

1. **server.js**
   - **Purpose**: This is the main entry point of the application. It sets up the server, connects to the database, and initializes middleware and routes.
   - **Key Methods and Functionality**:
     - mongoose.connect(): Connects to the MongoDB database using the URI from environment variables.
     - app.use(express.static('public')): Serves static files from the public directory.
     - app.use(express.json()) and app.use(express.urlencoded({extended: true})): Parses incoming JSON and URL-encoded payloads.
     - Route handlers for various endpoints (/api/keys/paypal, /api/upload, /api/seed, /api/products, /api/users, /api/orders, /api/flowers, /api/packings).
     - Error handling middleware to catch and respond to errors.
     - Static file serving for the frontend in production mode.

2. **utils.js**
   - **Purpose**: Contains utility functions for authentication and authorization.
   - **Key Methods**:
     - generateToken(user): Generates a JWT token for user authentication.
     - isAuth(req, res, next): Middleware to verify if the user is authenticated by checking the JWT token.
     - isAdmin(req, res, next): Middleware to verify if the authenticated user has admin privileges.

4.2 Frontend

The frontend was developed using React.js to create a dynamic and responsive user interface.

Reason: React.js offers a component-based architecture, which enhances code reusability and maintainability. Its support for state management and the virtual DOM improves performance and user experience.

Libraries:

- **React Bootstrap:** Possibly used to style and structure UI components using pre-built Bootstrap components tailored for React.
- **React Helmet Async:** Employed for managing document head tags like title, meta, link, and script.
- **@paypal/react-paypal-js:** Utilized for integrating PayPal payment functionality into the frontend.

In project there is folder Frontend that has such folders with such files:

- components
  - navbar
    - navbar.jsx
  - reducers
    - reducer.js
    - reducerProducts.js
  - JS files for Rating, Price, admin route, search bar.. etc.
- screens
- App.js
- Index.css
- Index.js
- Store.js

4.3 Backend

The backend was built using Node.js and Express.js, which handle server-side logic and API endpoints.
Reason: Node.js and Express.js are well-suited for creating scalable server-side applications. They provide a robust framework for handling HTTP requests, routing, and middleware.
Libraries:
Axios: Possibly used for making HTTP requests from the server-side code, such as fetching PayPal client IDs.

4.4 Database

MongoDB was selected for its flexibility in managing complex data structures, such as user reviews and product details.
Reason: MongoDB's NoSQL nature allows for dynamic changes in the schema without significant downtime or complex migrations. This flexibility is ideal for an evolving e-commerce platform, as it can easily adapt to new requirements and data models.

4.5 Authentication and Authorization

JSON Web Tokens (JWT) were implemented for secure authentication and authorization.
Reason: JWTs are stateless and can be easily used to verify user identity and permissions across the application. They enhance security by ensuring that sensitive data is not stored on the server and can be verified quickly on each request.

4.6 Payment Integration

The PayPal API was integrated to handle secure and efficient payment processing.
Reason: PayPal is a widely trusted payment gateway that provides robust security features. Its API allows for easy integration with the e-commerce platform, ensuring users can make payments securely and conveniently.
There is a folder Backend which has such folders and files:

- Src
- Models (models for entities)
- Routes (routes for these models)
- Data.js
- Server.js
- Utils.js

4.7 Use-cases

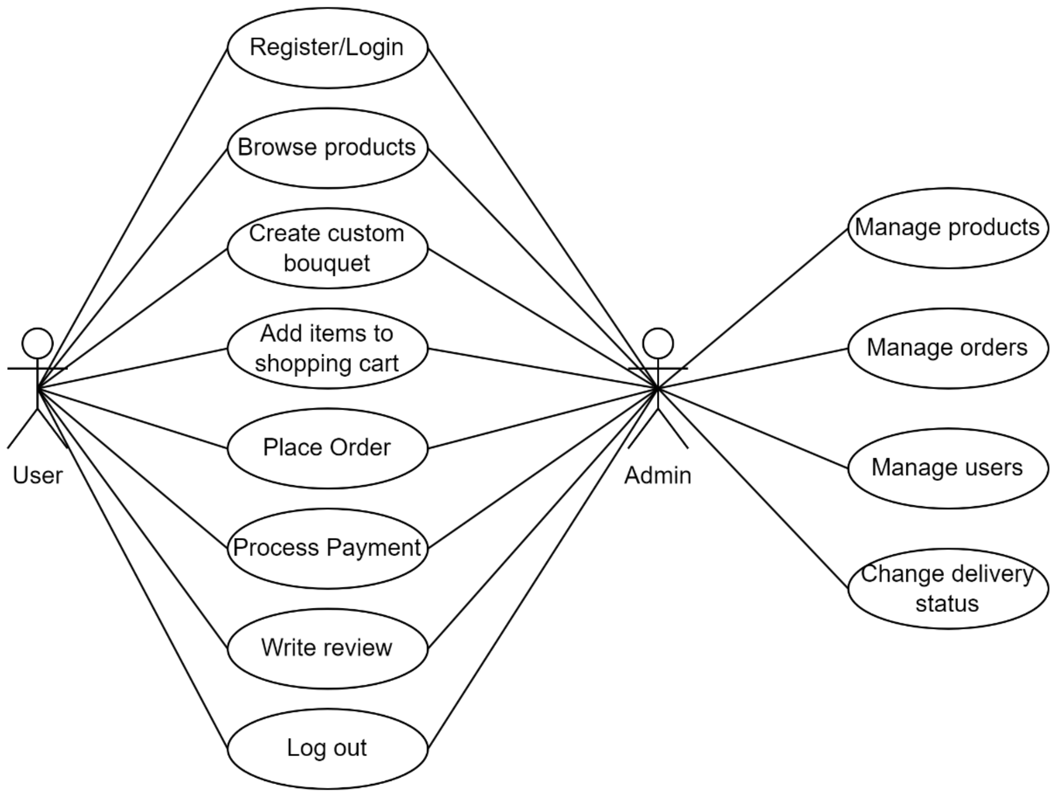

Figure 7. Use case diagram

The diagram in Figure 7 shows how users and admins can interact with the website.

User Use Cases:

1. **Register**: Users can create an account by providing necessary details.
2. **Login**: Users can log in to their account using their credentials.
3. **Browse Products**: Users can view a list of available products.
4. **Create Custom Bouquet**: Users can design their own bouquet by selecting flowers and other items.
5. **Add Items to Shopping Cart**: Users can add products or custom bouquets to their shopping cart.
6. **Place Order**: Users can proceed to checkout and place an order for the items in their shopping cart.
7. **Process Payment**: Users can make payments for their orders using various payment methods.
8. **Write Review**: Users can leave reviews and ratings for products they have purchased.
9. **Log Out**: Users can log out from their account to end their session.

Admin Use Cases:

In addition to all the user functionalities, administrators have additional privileges:

1. **Manage Products**: Admins can add, update, or delete products in the inventory.
2. **Manage Orders**: Admins can view and update the status of orders.
3. **Manage Users**: Admins can manage user accounts, including adding or removing users.
4. **Change Delivery Status**: Admins can update the delivery status of orders to reflect their current state.

4.8 Deployment

Execution Environment:

- **VU MIF OpenNebula Sunstone Cloud**: This is the cloud infrastructure where the entire deployment was hosted. OpenNebula Sunstone provides a web-based user interface for managing virtualized data centers.

Devices:

- **Web Server VM**: This is a virtual machine running on the VU MIF OpenNebula Sunstone Cloud. It hosts the Docker daemon and acts as the primary server for the application.

Operating System:

- **Ubuntu 22.04 Linux**: The operating system running on the Web Server VM. Ubuntu is a popular choice for server environments due to its stability and extensive support community.

Docker Containers:

- **Docker Daemon**: The Docker daemon running on the Web Server VM manages the lifecycle of Docker containers. It is responsible for starting, stopping, and managing containers.
  - **Container 1: Web Server**
    - **Description**: This container runs an instance of the MongoDB database.
    - **Connection Protocol**: Connects using the TCP/IP protocol.
  - **Container 2: Backend**
    - **Description**: This container runs a Node.js and Express application that serves as the backend API for the application.
    - **Connection Protocol**: Connects using the HTTP protocol.
  - **Container 3: Frontend**
    - **Description**: This container runs a React application that serves as the frontend of the web application.
    - **Connection Protocol**: Connects to the backend API using the HTTP protocol.
- **Proxy Server: Nginx**
  - **Description**: Nginx acts as a reverse proxy server. It routes requests from users' web browsers to the appropriate backend service (either the frontend React application or the backend Node.js API).
  - **Connection Protocol**: Connects with port 80 to user devices using any modern web browser.
  - **Artifact**: Serves HTML, CSS, and JavaScript files to the user's browser

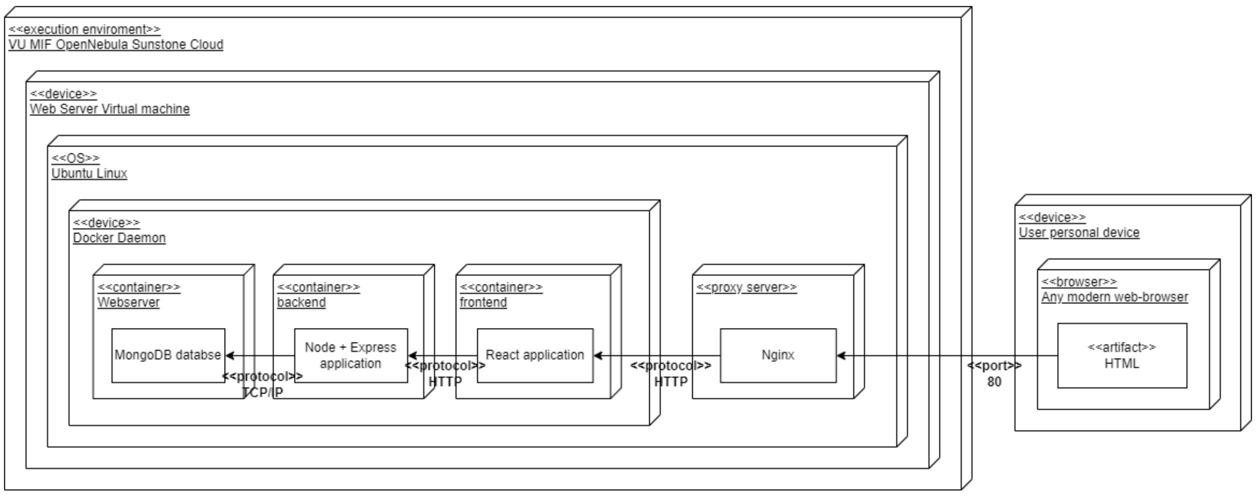

Figure 8. Deployment diagram

Explanation:

1. **User Personal Device**: Users access the application using a web browser. They make HTTP requests to the Nginx proxy server.
2. **Nginx Proxy Server**: Acts as a reverse proxy, forwarding requests to the appropriate service. It routes requests for static assets (HTML, CSS, JS) to the frontend container and API requests to the backend container.
3. **Frontend Container**: Runs the React application, serving the user interface of the application.
4. **Backend Container**: Runs the Node.js and Express application, providing API endpoints and business logic.
5. **MongoDB Database Container**: Hosts the MongoDB database, storing application data. The backend container communicates with this database using the TCP/IP protocol.
6. **Web Server VM**: The host machine running the Docker daemon and all the containers.
7. **VU MIF OpenNebula Sunstone Cloud**: The cloud infrastructure providing the virtual machine and underlying resources for the deployment.

5 Challenges during implementing project

Deploying the application was a challenge for me because I had never done it before. So it was a bit difficult to understand how to work with containers and how to create proxies. In addition, implementing the payment functionality using the PayPal API was a challenge for me as it was also my first time doing it. My developer account was blocked and I had to solve this problem. In the middle of the process, my laptop broke down and I was without it for a few weeks and I had a pause in the development of the application, so that was a real problem, I lost several dozen hours. Working with new libraries and frameworks was also challenging due to my lack of experience, but various manuals, research articles, documentation, and tutorials helped me solve these problems.

Also, designing the project itself was not an easy task because I think it is the most important and responsible part of the whole project. How you plan a project is how it will be. Therefore, I had to take it responsibly and think over every detail and write it down.

But summing up all the challenges I faced, they all gave me invaluable knowledge and experience that I will use for a very long time.

6 Conclusions and Recommendations

It was an interesting and challenging development of this application. And I made some insights for future development. First, preparing a good plan for development is half of all the work. Secondly, to work using methodologies, for example, Agile will make the process faster and with higher quality. Thirdly, splitting tasks into iterations will make the work process better. Cause, the project progresses in manageable parts, allowing for regular assessments. And making research, studying new technologies, and communicating with a mentor or supervisor will make a big impact on the result. This approach fosters innovation and ensures adherence to best practices.

Recommendations would be to make notes, snippets of code for example how PayPal API was integrated into the app, so next time it will be easier to integrate it into another project. Also as for not the process of development but for the project, to add more functionality, to make creating custom bouquet features better visually displayed. Add information about the shop and contacts. And integrate more variants of payment.

References

[1] Create React App. Adding Bootstrap, https://create-react-app.dev/docs/adding-bootstrap/

[2] IJMTST Editor. Application using MERN Stack. https://www.researchgate.net/profile/Editor-Ijmtst/publication/361465446_Application_using_MERN_Stack/links/62b316081010dc02cc538a11/Application-using-MERN-Stack.pdf

[3] Faker.js. https://fakerjs.dev/

[4] LogRocket. How to Make HTTP Requests Like a Pro with Axios, https://blog.logrocket.com/how-to-make-http-requests-like-a-pro-with-axios/

[5] PayPal Developer. REST API. https://developer.paypal.com/api/rest/

[6] Permify. JWT Authentication in React. https://permify.co/post/jwt-authentication-in-react/

[7] Bryntum. React Grid. https://bryntum.com/products/react-grid/

[8] Tran, T. T. H. (2022). Developing a Web Application Using MERN Stack. https://www.theseus.fi/bitstream/handle/10024/752089/Tran_Thi%20Thu%20Hien.pdf?sequence=2

[9] Mohanish Bawane , Ishali Gawande , Vaishnavi Joshi , Rujuta Nikam , Prof. Sudesh A. Bachwani (2022). International Journal for research. A review on Technologies used in MERN stack. https://doi.org/10.22214/ijraset.2022.39868

[10] V. Sharma, H. K. Saxena and A. K. Singh, "Docker for Multi-containers Web Application," 2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore, India, 2020, pp. 589-592, https://doi.org/10.1109/ICIMIA48430.2020.9074925