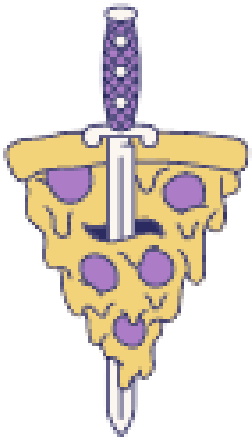
# *Munch: A MERN Stack Food Delivery Website*



**Munch.**

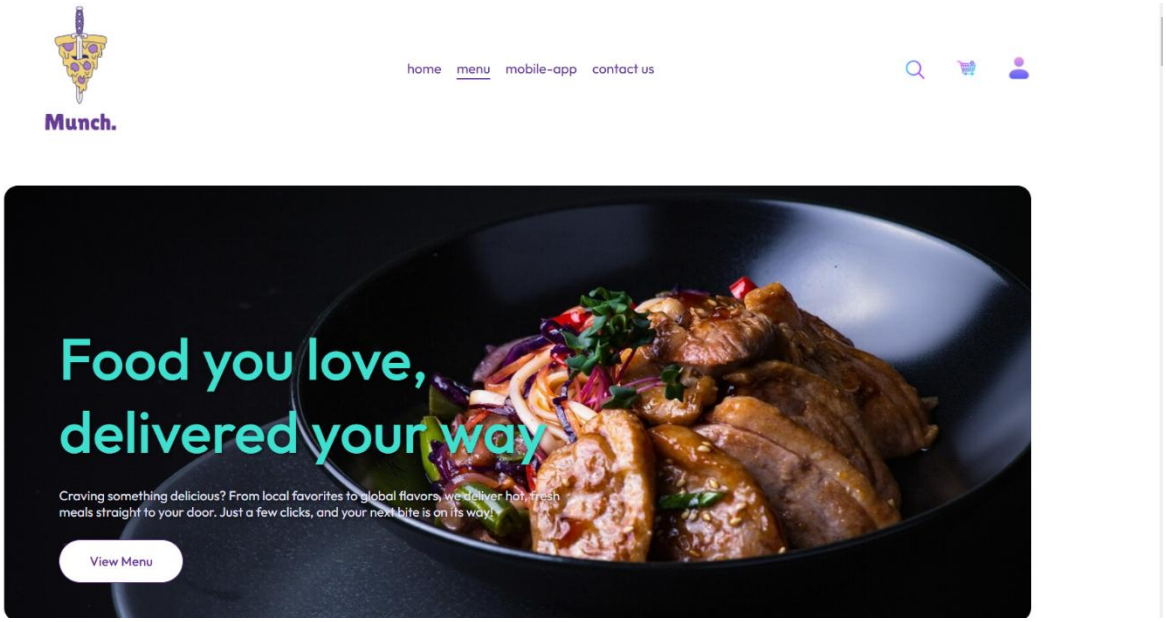
*Kseniia Starodubtseva*

*Matricola: 546484*

# Introduction

Munch is a web-based food delivery platform developed using the MERN (MongoDB, Express.js, React, Node.js) stack. The platform facilitates food ordering and delivery through an intuitive user interface. It is designed with two primary panels: the User Panel, where customers can browse food items, add them to their cart, place orders, and track order status, and the Admin Panel, where administrators can manage food listings, view and update orders, and oversee platform operations.

The development of Munch aimed to create a scalable and efficient food delivery system, integrating essential functionalities such as authentication, cart management, order processing, and order tracking.




*Homepage*

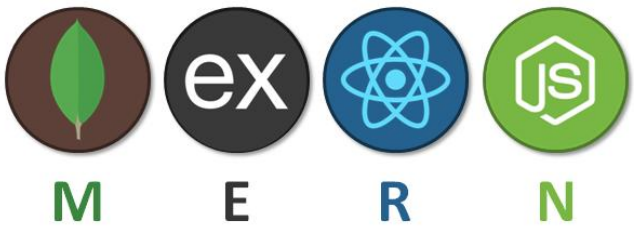
# Motivation for Technology Choices

The MERN stack was chosen for its efficiency, scalability, and ability to build full-stack applications using JavaScript. Here is a breakdown of each technology and why it was used:

1. *MongoDB*: A NoSQL database that allows flexible data storage, ideal for handling user-generated content such as food orders and customer details.
2. *Express.js*: A lightweight Node.js framework for handling backend logic, API routes, and middleware functionalities efficiently.
3. *React.js*: A powerful frontend library that ensures fast, dynamic, and interactive UI updates without refreshing the page.
4. *Node.js*: A JavaScript runtime that enables asynchronous, event-driven programming, making it well-suited for handling multiple user requests concurrently.

Other libraries and tools were also incorporated, including:

1. *Axios:* For handling HTTP requests between the frontend and backend.
2. *JWT (JSON Web Token):* For secure authentication and user authorization.
3. *Mongoose:* For interacting with MongoDB using an object data modeling (ODM) library.
4. *CSS Modules:* To ensure component-based styling with minimal conflicts.



# Project Structure

The project is structured into three main directories: admin, backend, and frontend. Below is a breakdown of their contents:

*1) Admin Panel (admin/src)*

- *Assets*: Contains images, icons, and other static resources.
- *Components*: Reusable UI components like Navbar.jsx, Sidebar.jsx.
- *Pages:*
    - Add.jsx: Allows admins to add new food items.
    - List.jsx: Displays all food items in the system.
    - Orders.jsx: Provides an overview of all customer orders with status updates.

*2) Backend (backend/)*

- *Config:*
    - db.js: Manages MongoDB connection.
- *Controllers:*
    - cartController.js: Handles cart-related operations.
    - foodController.js: Manages food item CRUD operations.
    - orderController.js: Processes orders, status updates, and retrieval.
    - userController.js: Handles user authentication and profile management.
- *Middleware:*
    - auth.js: Middleware to protect routes that require authentication.
- *Models:*

- foodModel.js: Defines the structure of food items.
- orderModel.js: Stores order details and status.
- userModel.js: Manages user authentication and profile information.

- *Routes:*
  - cartRoute.js, foodRoute.js, orderRoute.js, userRoute.js: Define API endpoints for various functionalities.
- *Uploads:* Stores images of food items.
- *.env File*: Contains environment variables such as database credentials and JWT secret keys.

*3) Frontend (frontend/src)*

- *Assets:* Static resources for the UI.
- *Components:* UI components like Navbar, Footer, and FoodDisplay.
- *Context:*
  - StoreContext.jsx: Manages global state using React Context API.
- *Pages:*
  - Cart.jsx: Displays cart contents and allows order placement.
  - Home.jsx: The main landing page with featured food items.
  - MyOrders.jsx: Displays previous orders for logged-in users.
  - PlaceOrder.jsx: Handles checkout and order confirmation.
  - Verify.jsx: Implements email-based verification.

# API Structure

The backend exposes multiple RESTful API endpoints for data exchange. Key routes include:

*User Authentication*

- POST /api/user/register – Registers a new user.
- POST /api/user/login – Authenticates a user and returns a JWT token.

*Food Management*

- GET /api/food – Fetches all available food items.
- POST /api/food – Adds a new food item (admin only).
- DELETE /api/food/:id – Removes a food item (admin only).

*Cart and Orders*

- POST /api/cart/add – Adds an item to the user's cart.
- POST /api/order/place – Places an order for the items in the cart.
- GET /api/order/userorders – Retrieves orders for the logged-in user.
- POST /api/order/update – Allows admins to update order statuses.

# Database Design

Munch follows a document-based schema using MongoDB. The main collections are:

*User Model (userModel.js)*

```
const UserSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  password: String,
  isAdmin: { type: Boolean, default: false }
});
```

*Food Model (foodModel.js)*

```
const FoodSchema = new mongoose.Schema({
  name: String,
  price: Number,
  category: String,
  image: String
});
```

*Order Model (orderModel.js)*

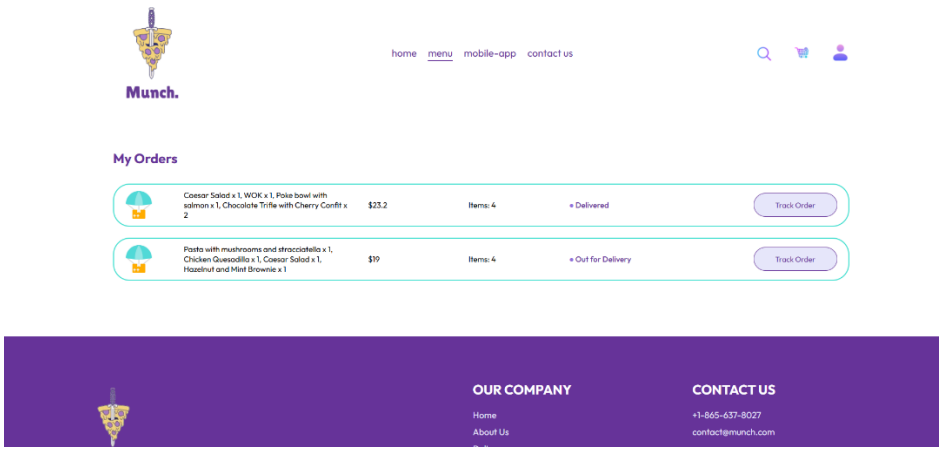
```
const OrderSchema = new mongoose.Schema({
  userId: mongoose.Schema.Types.ObjectId,
  items: [{ foodId: String, quantity: Number }],
  status: { type: String, default: "Pending" }});
```

## Unique Features

1. *Real-time Order Tracking*: Users can monitor their order status in real time.
2. *Secure Authentication*: JWT-based authentication ensures user data protection.
3. *Admin Management Dashboard*: Allows full control over food listings and orders.
4. *Scalability*: The NoSQL database design ensures flexibility for future expansions.
5. *Mobile-Optimized UI*: A responsive frontend ensures usability across devices.
6. *Stripe Payment Integration*: Stripe is integrated for online payments.

## Security Measures

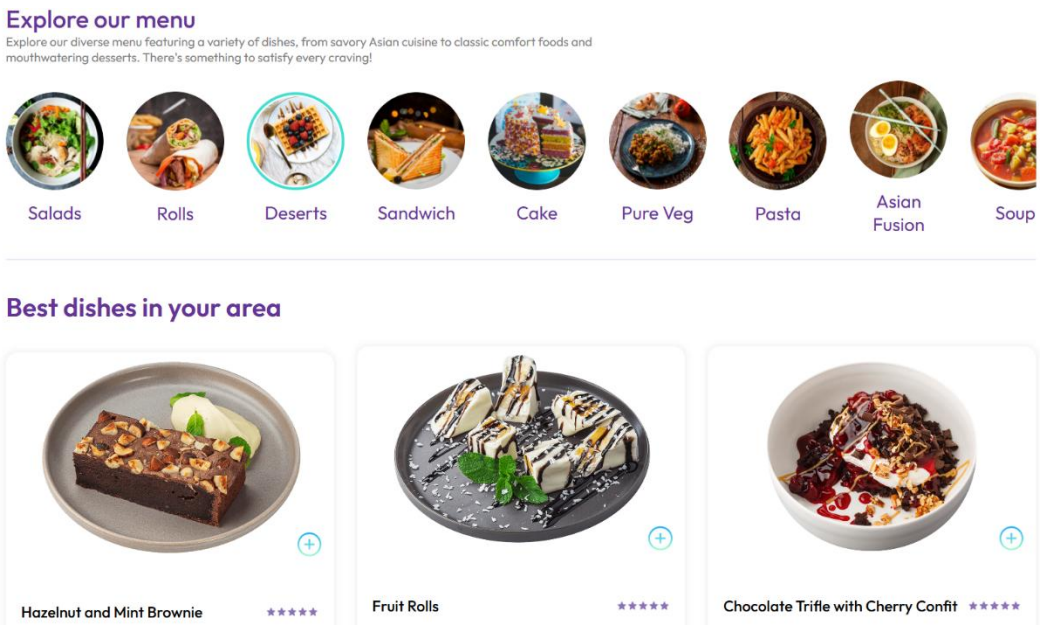
- *JWT Authentication*: Ensures only authorized users can place orders and access personal data.
- *Password Hashing*: User passwords are securely stored using bcrypt hashing.
- *Protected Routes*: Middleware restricts access to certain routes.



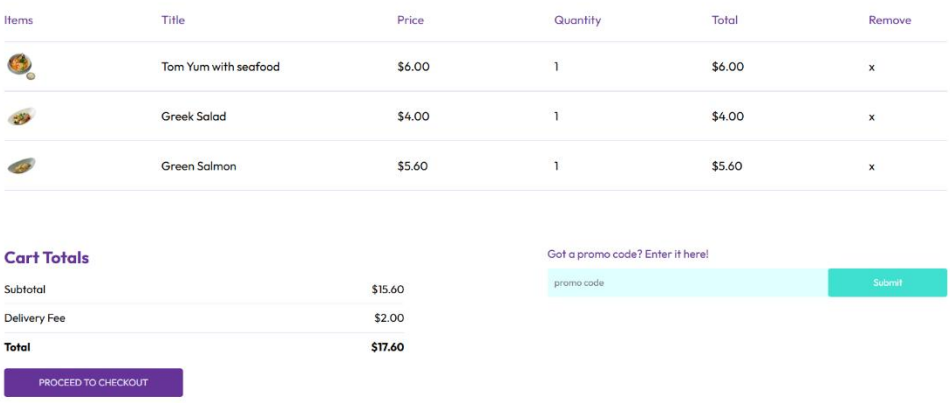

*My orders user page*

# Conclusion

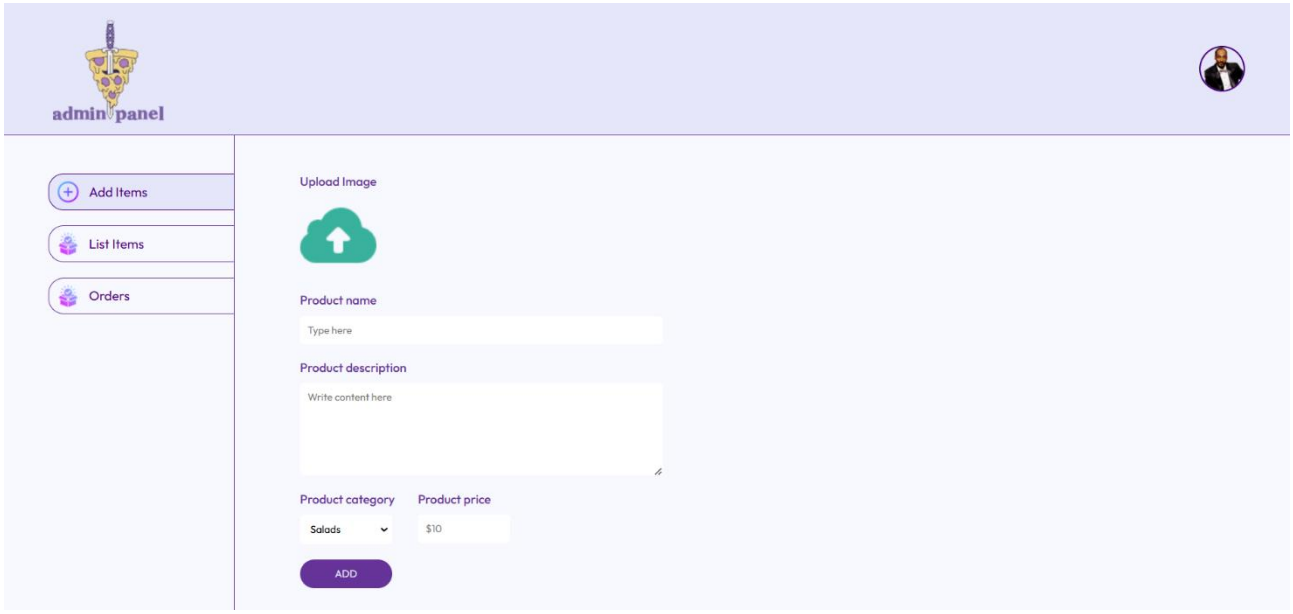
Munch demonstrates the power of the MERN stack in building a robust, scalable food delivery system. By combining an intuitive frontend with a secure backend, it provides a seamless experience for users and administrators. Future improvements will focus on enhancing scalability, automation, and AI-driven features for a more dynamic experience.



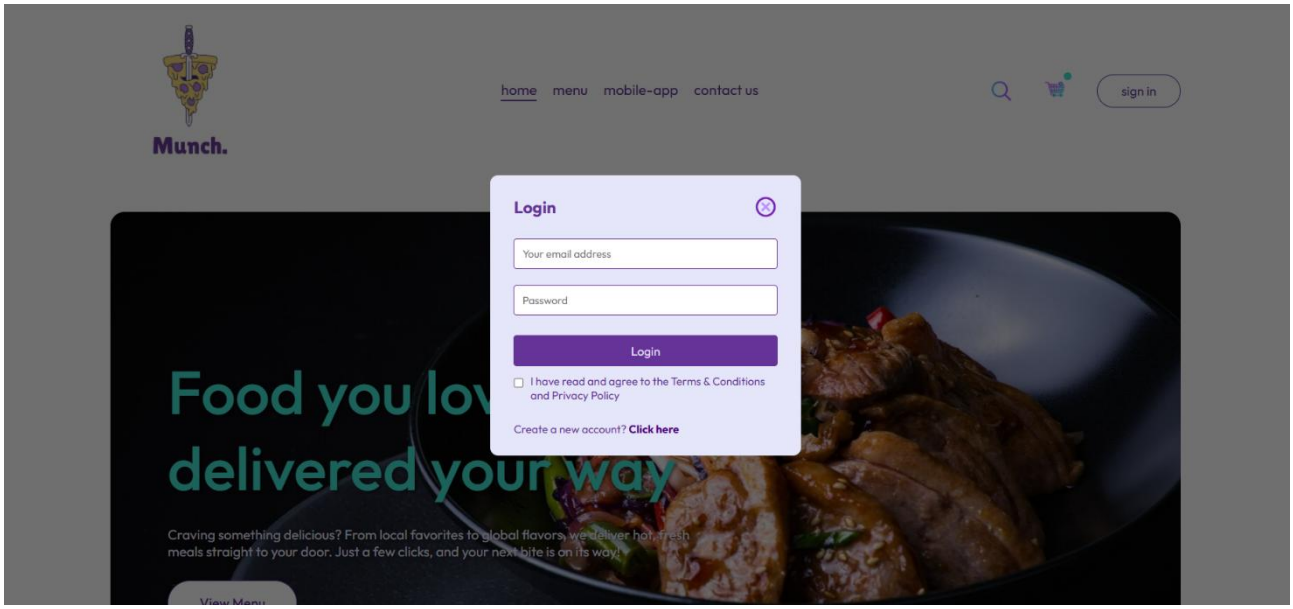

*Filtering food items by category*




*Cart page*




*Admin panel functionality*




*Login/Sign Up*