Web Application Development

Web App Development

GrabFood Web Application

Team: PDM

Members:

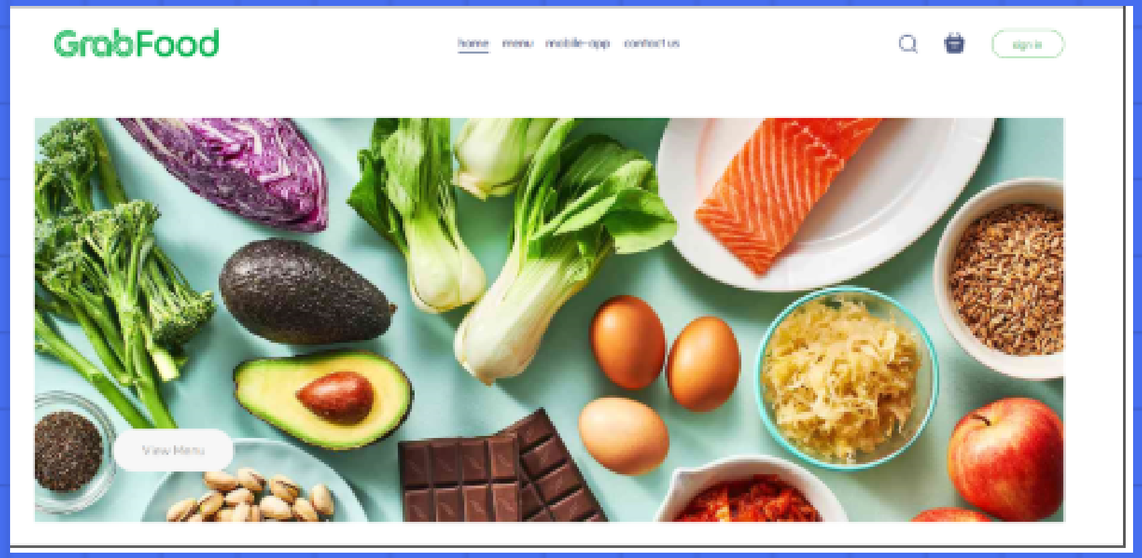
1. Nguyễn Hoàng Minh Khôi - ITITIU21229
2. Nguyễn Trần Gia Huy - ITITIU21054
3. Đặng Nguyễn Trường Huy - ITITIU21010
4.

Công ty MDM

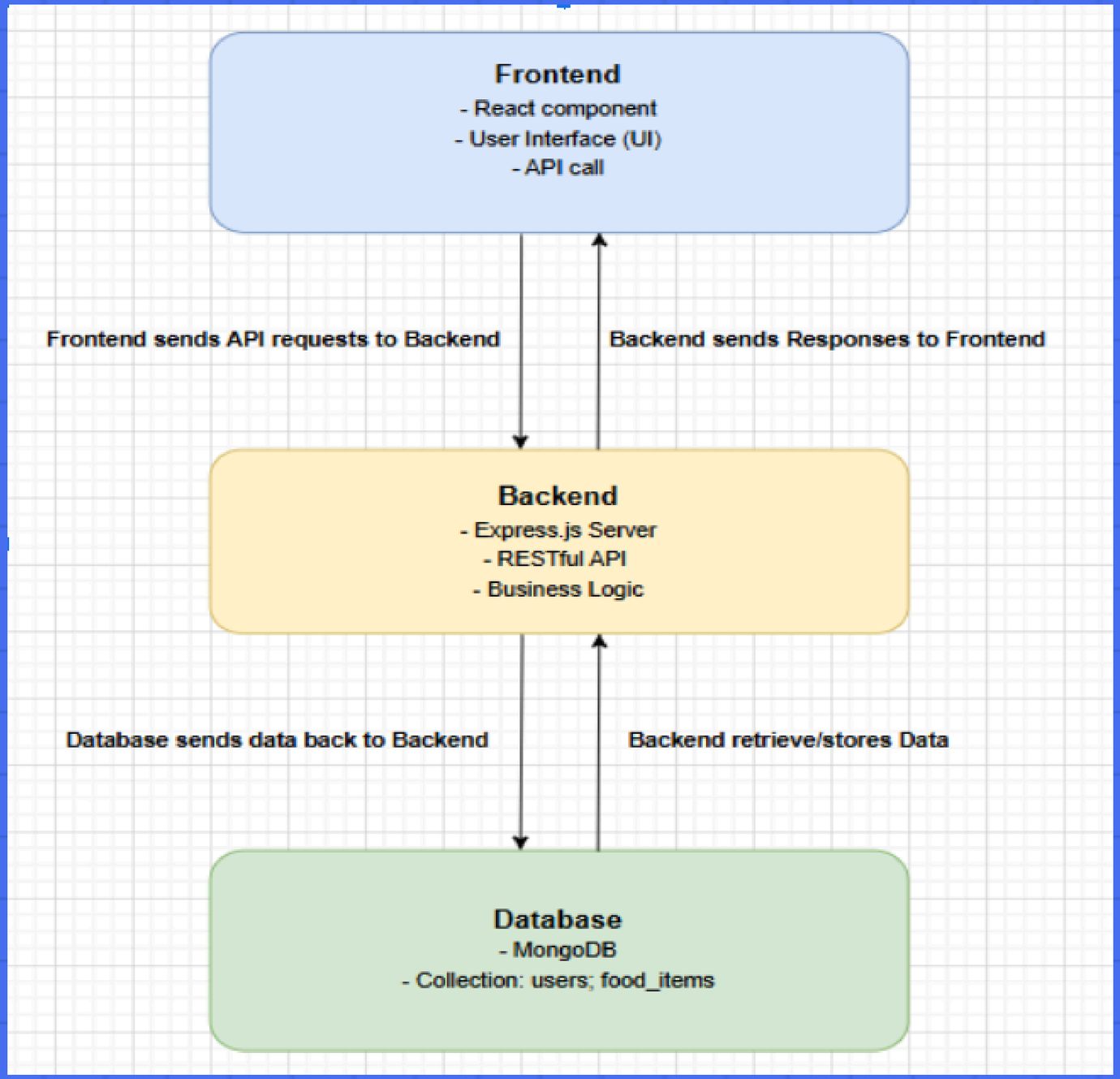
1 tháng 6 năm 2030

Introduction

The GrabFood Web Application simplifies food ordering with a user-friendly platform to browse menus, place orders, and make secure payments from home. Built with React.js, Node.js, JavaScript, and MongoDB, it ensures performance, accessibility, scalability, and data security. Focused on core features like menu browsing and secure payments, the app caters to busy individuals seeking a reliable and efficient food delivery solution, optimized for web use



System Design Overview



1. Frontend: Handles user interaction (React.js)
2. Backend: Processes requests and communicates with the database (Node.js, Express.js).
3. Database: Stores user, food, and order data (MongoDB).

Technology

Frontend Technology

- HTML: Defines the structure of the web pages.
- CSS: Applies styling to make the interface appealing and responsive.
- JavaScript: Enables interactivity and dynamic behavior.
- React.js: Component-based framework for reusable and scalable UI.
- Vite.js: Fast build tool for development and production.

GrabFood

home menu mobile-app contact us

sign in

Explore our menu

Choose from our menu


Salad


Rolls


Deserts


Sandwich

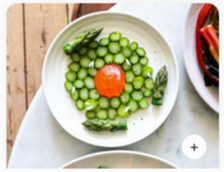

Cake


Pure Veg


Pasta


Noodles

Top dishes near you


Greek salad ★★★★☆

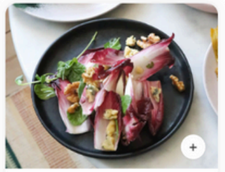
Food provides essential nutrients for overall health and well-being

$12


Veg salad ★★★★☆

Food provides essential nutrients for overall health and well-being

$18


Clover Salad ★★★★☆

Food provides essential nutrients for overall health and well-being

$16


Chicken Salad ★★★★☆

Food provides essential nutrients for overall health and well-being

$24


Lasagna Rolls ★★★★☆

Food provides essential nutrients for overall health and well-being

$14

Items	Title	Price	Quantity	Total	Remove
	Veg salad	$18	1	$18	x
	Clover Salad	$16	3	$48	x

Cart Total

Subtotal	$66
Delivery Fee	$2
Total	**$68**

PROCEED TO CHECKOUT

Enter promocode

promocode Submit

Technology

Backend Technology

- Node.js: Handles asynchronous requests and ensures smooth server-side processing.
- Express.js: Defines routes and middleware for APIs.
- MongoDB: NoSQL database for flexible, document-based storage
- RESTful APIs: /api/user, /api/food, /api/orders.
- Authentication: Secured with JSON Web Tokens (JWT).

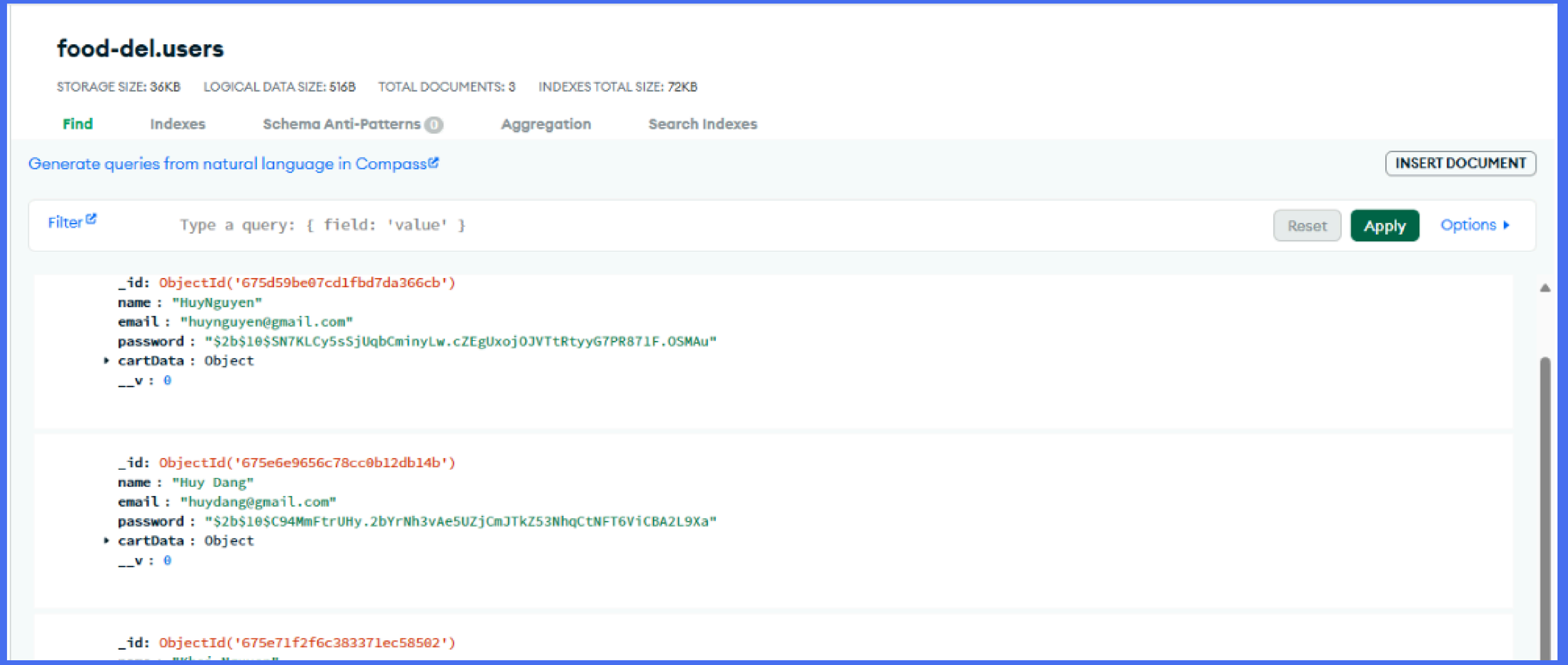
```
import express from "express"
import cors from "cors"  5k (gzipped: 2.1k)
import { connectDB } from "./config/db.js"
import userRouter from "./routes/userRoute.js"
import 'dotenv/config'  7.5k (gzipped: 3.2k)


//app config
const app = express()
const port = 5137

//middlewares
app.use(express.json())
app.use(cors())

//db connection
connectDB();
      HuyNguyen2305, 21 hours ago • update 21:00 241215
//api endpoints
app.use("/api/user", userRouter)

app.get("/",(req,res)=>{
    res.send("API Working")
})
app.listen(port, ()=>{
    console.log(`Server is running on http://localhost:${port}`)
})
```

Database Design

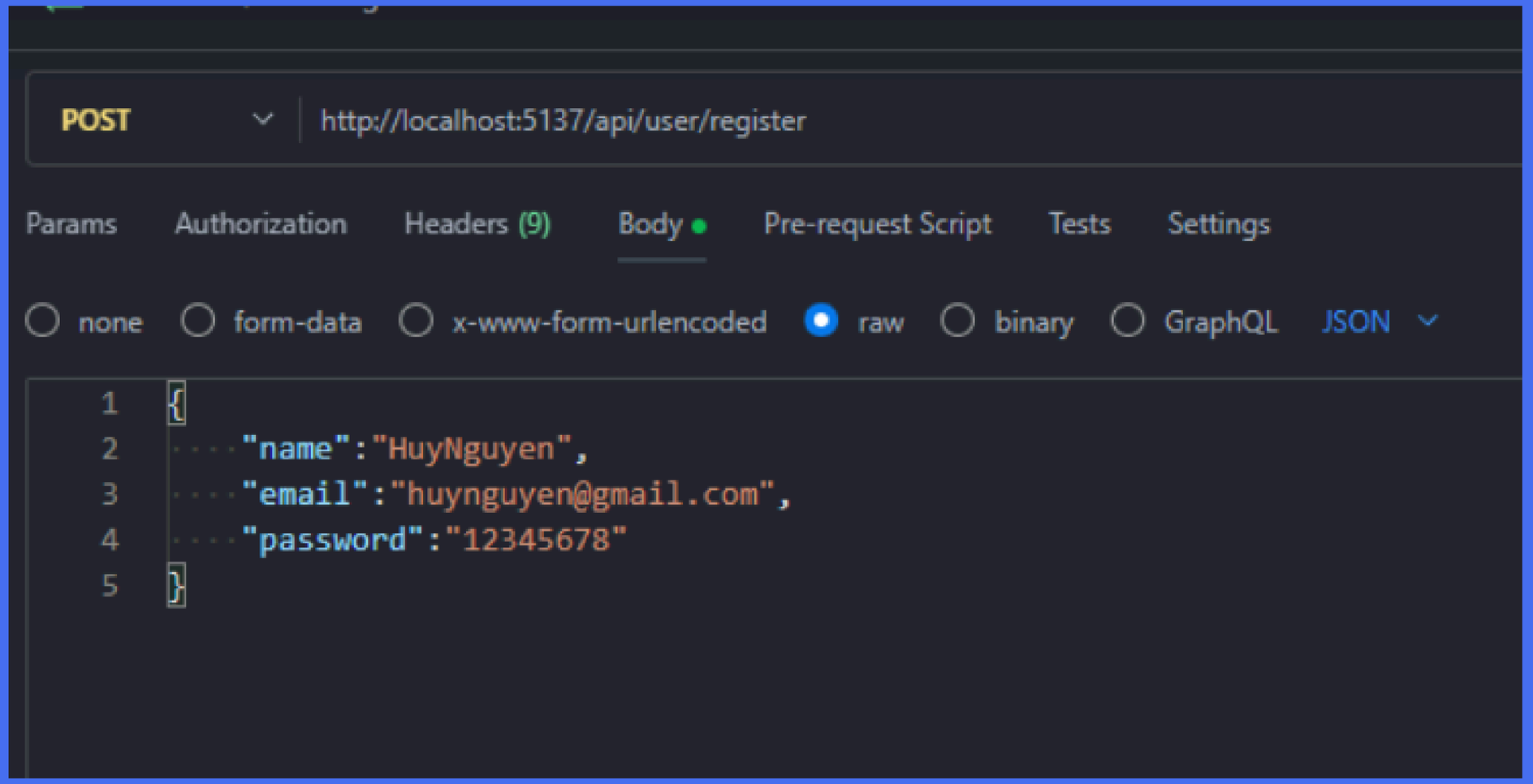
Database Design Schema

MongoDB Schema:

- Users: Stores user credentials (name, email, hashed password).
- Food Items: Stores menu details (name, description, price, category).
- Orders: Links users to their order history (food items, quantity, status).


food-del.users
STORAGE SIZE: 36KB
LOGICAL DATA SIZE: 516B
TOTAL DOCUMENTS: 3
INDEXES TOTAL SIZE: 72KB
Find
Indexes
Schema Anti-Patterns 0
Aggregation
Search Indexes
Generate queries from natural language in Compass
INSERT DOCUMENT
Filter
Type a query: { field: 'value' }
Reset
Apply
Options
_id: ObjectId('675d59be07cd1fbd7da366cb')
name : "HuyNguyen"
email : "huynguyen@gmail.com"
password : "$2b$10$SN7KLCy5sSjUqbCminyLw.cZEgUxojOJVTtRtyyG7PR871F.OSMAu"
cartData : Object
__v : 0
_id: ObjectId('675e6e9656c78cc0b12db14b')
name : "Huy Dang"
email : "huydang@gmail.com"
password : "$2b$10$C94MmFtrUHy.2bYrNh3vAe5UZjCmJTkZ53NhqCtNFT6ViCBA2L9Xa"
cartData : Object
__v : 0
_id: ObjectId('675e71f2f6c383371ec58502')

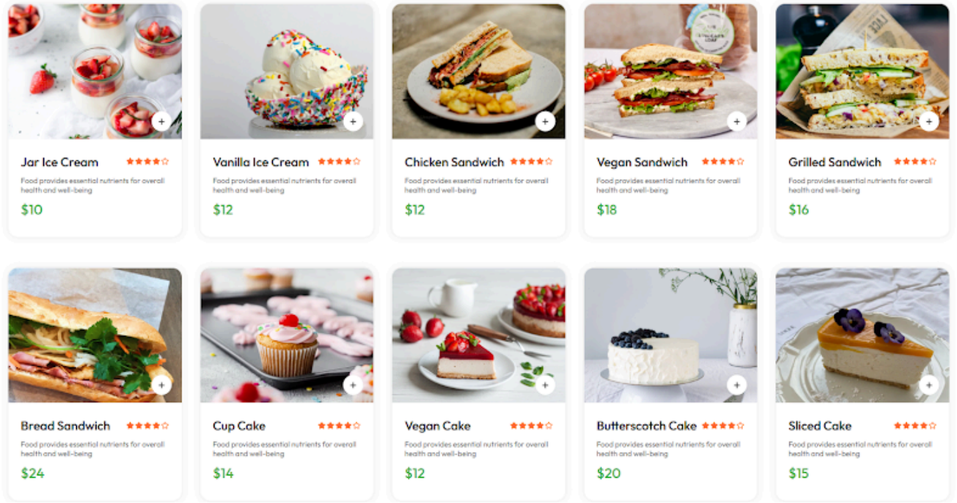
API POST Method



Database Design

Implementation

- **Front end**
- **Back end**



Front end

Frontend development of the application focused on creating a responsive, user-friendly interface using React. Key components and their features include:

React Components: Modular, reusable components were created for features like Navbar, Home, Cart, and Food Display.

General Features

State Management: Managed using React hooks (useState, useContext) for handling authentication, cart data, and dynamic interactions.

Styling and Responsiveness: CSS ensured visual appeal and compatibility across devices and screen sizes.

Key Components and Functionalities

Navigation Bar (Navbar)
Login Popup
Add to Cart
Food Display

Component Structure

```
You, 5 hours ago | 2 authors (HuyNguyen2305 and one other)
import React, { useContext, useState } from 'react'  6.9k (gzipped: 2.7k)
import './Navbar.css'
import { assets } from '../../assets/assets'
import { Link, useNavigate } from 'react-router-dom';  224.7k (gzipped: 71k)
import { StoreContext } from '../../context/StoreContext';

const Navbar = ({setShowLogin}) => {

    const [menu,setMenu] = useState("home");

    const {getTotalCartAmount, token, setToken} = useContext(StoreContext);

    const navigate = useNavigate();

    const logout = () =>{
        localStorage.removeItem("token");
        setToken("");
        navigate("/")
    }
```

```
return (
    <div className='navbar'>
        <Link to='/'><img src={assets.logo} alt="" className="logo" /></Link>
        <ul className='navbar-menu'>
            <Link to='/' onClick={()=>setMenu("home")} className={menu==="home"?"active":""}>home</Link>
            <a href="#explore-menu" onClick={()=>setMenu("menu")} className={menu==="menu"?"active":""}>menu</a>
            <a href='#app-download' onClick={()=>setMenu("mobile-app")} className={menu==="mobile-app"?"active":""}
            <a href='#footer' onClick={()=>setMenu("contact-us")} className={menu==="contact-us"?"active":""}>conta
        </ul>
        <div className='navbar-right'>
            <img src={ assets.search_icon} alt="" />
            <div className='navbar-search-icon'>
                <Link to="/cart"><img src={assets.basket_icon} alt="" /></Link>
                <div className={getTotalCartAmount()===0?"":"dot"}></div>
            </div>
            {!token?<button onClick={()=>setShowLogin(true)}>sign in</button>
            :<div className='navbar-profile'>
                <img src={assets.profile_icon} alt="" />      HuyNguyen2305, 14 hours ago • update 21:00 24121
                <ul className="nav-profile-dropdown">
                    <li><img src={assets.bag_icon} alt="" /><p>Orders</p></li>
                    <hr />
                    <li onClick={logout}><img src={assets.logout_icon} alt="" /><p>Logout</p></li>
                </ul>
            </div>}

        </div>
    </div>
)
}

export default Navbar
```

Nav Bar

- Dynamic menu highlighting using useState.
- Global state integration via useContext for authentication and cart updates.
- Conditional rendering of authentication options (e.g., Sign In vs. user profile).
- Logout functionality clears tokens and redirects users to the home page.
- Responsive design allows seamless navigation across application sections.

Nav Bar implemented on our page

Explore our menu

Choose from our menu


Salad


Rolls


Deserts


Sandwich

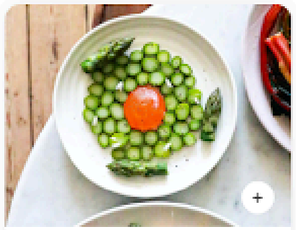

Cake


Pure Veg


Pasta


Noodles

Top dishes near you



Greek salad ★★★★☆

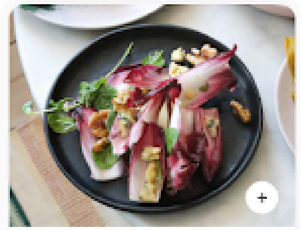
Food provides essential nutrients for overall health and well-being

$12

Veg salad ★★★★☆

Food provides essential nutrients for overall health and well-being

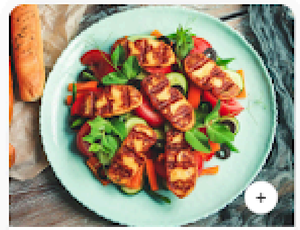
$18



Clover Salad ★★★★☆

Food provides essential nutrients for overall health and well-being

$16



Chicken Salad ★★★★☆

Food provides essential nutrients for overall health and well-being

$24

Lasagna Rolls ★★★★☆

Food provides essential nutrients for overall health and well-being

$14

GrabFood

home menu mobile-app contact us


sign in

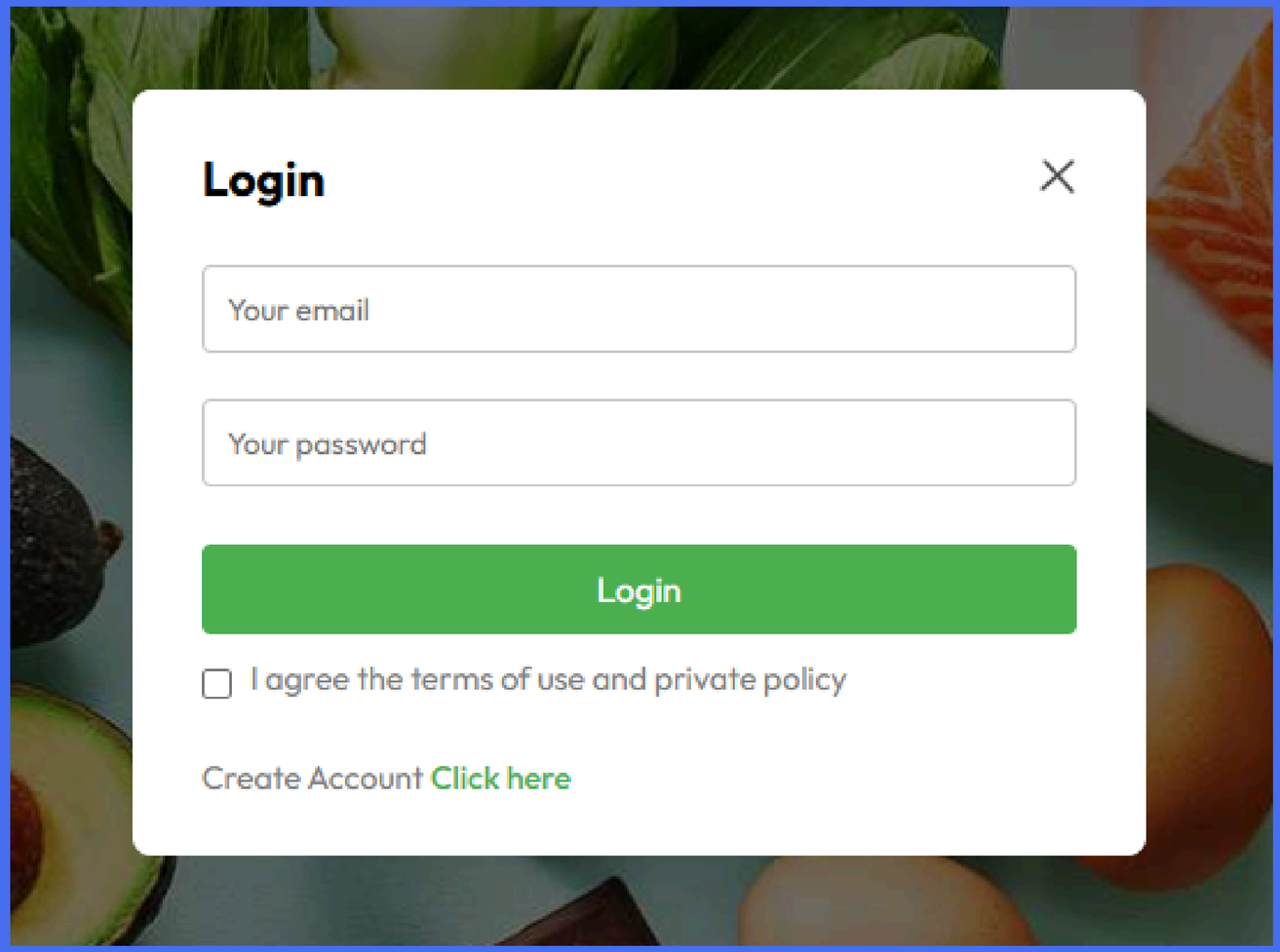
```jsx
const LoginPopup = ({setShowLogin}) => {

    const {url, setToken} = useContext(StoreContext)

    const [currState, setCurrState] =useState("Sign up")
    const [data,setData] = useState({
        name:"",
        email:"",
        password:""
    })

    const onChangeHandler = (event) =>{
        const name = event.target.name;
        const value = event.target.value;
        setData(data=>({...data,[name]:value}))
    }
    const onLogin = async(event) =>{
        event.preventDefault()
        let newUrl = url;
        if(currState==="Login"){
            newUrl += "/api/user/login"
        }
        else{
            newUrl +="/api/user/register"
        }
        const response = await axios.post(newUrl, data);
        if(response.data.success){
            setToken(response.data.token);
            localStorage.setItem("token", response.data.token)
            setShowLogin(false)
        }
        else {
            alert(response.data.message)
        }
    }
```

Component Structure

Login Popup

- Dynamic form toggling between Login and Signup using useState.
- Integration with StoreContext for API interaction and token storage.
- Form validation and error handling with user feedback via alerts.

LoginPopup implemented on our page



```
const addToCart = (itemId) => {
    if (!cartItems[itemId]){
        setCartitems((prev)=>({...prev,[itemId]:1}))
    }
    else {
        setCartitems((prev)=>({...prev,[itemId]:prev[itemId]+1}))
    }
}

const removeFromCart = (itemId) => {
    setCartitems((prev)=>({...prev,[itemId]:prev[itemId]-1}))
}

const getTotalCartAmount = () => {
    let totalAmount = 0;
    for(const item in cartItems)
    {
        if (cartItems[item]>0){
            let itemInfo = food_list.find((product)=>product._id === item)
            totalAmount += itemInfo.price* cartItems[item];
        }
    }
    return totalAmount;
}
```

Component Structure

Add to cart

- Context API used for cart state and functions (addToCart, removeFromCart).
- Conditional rendering of "Add" button or quantity counter.
- Event handlers update cart state dynamically.

Add to cart and remove from cart implemented on our page

Items	Title	Price	Quantity	Total	Remove
	Chicken Rolls	$20	2	$40	x
	Veg Rolls	$15	2	$30	x
	Ripple Ice Cream	$14	1	$14	x

Cart Total

Subtotal	$84
Delivery Fee	$2
Total	**$86**

PROCEED TO CHECKOUT

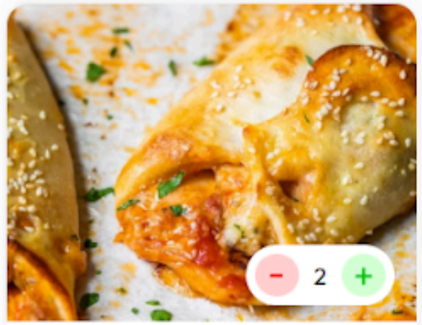
Enter promocode

promocode | Submit

Chicken Rolls ★★★★☆

Food provides essential nutrients for overall health and well-being

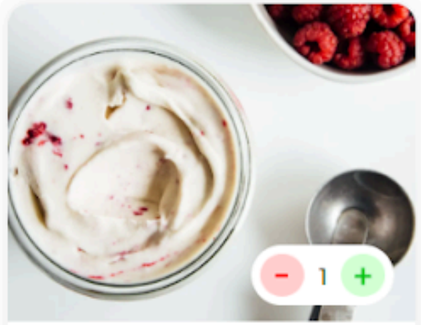
$20



Veg Rolls ★★★★☆

Food provides essential nutrients for overall health and well-being

$15



Ripple Ice Cream ★★★★☆

Food provides essential nutrients for overall health and well-being

$14

```
import React, { useContext } from 'react'      HuyNguyen2305, 5 days ago • Initial commit   6.9k (gzipped: 2.7k)
import './FoodDisplay.css'
import { StoreContext } from '../../context/StoreContext'
import FoodItem from '../FoodItem/FoodItem'

const FoodDisplay = ({category}) => {

    const {food_list} = useContext(StoreContext)

  return (
    <div className='food-display' id='food-display'>
        <h2>Top dishes near you</h2>
        <div className="food-display-list">
          {food_list.map((item,index)=>{
            if(category==="All" || category===item.category){
              return <FoodItem key={index} id={item._id} name={item.name} description={item.description} price={ite
            }

          })}
        </div>
    </div>
  )
}

export default FoodDisplay
```

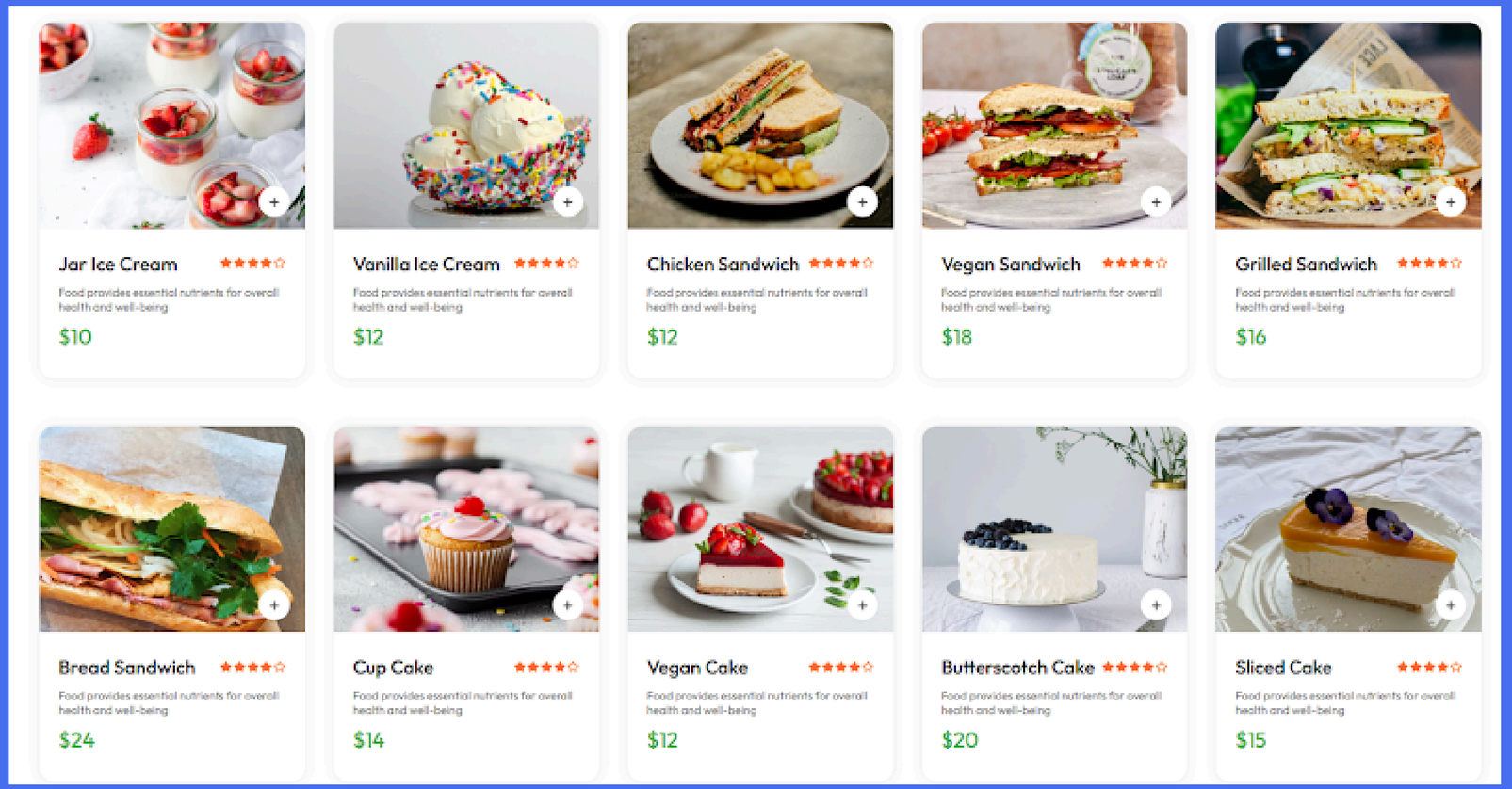
```
export const food_list = [
    {
        _id: "1",
        name: "Greek salad",
        image: food_1,
        price: 12,
        description: "Food provides essential nutrients for overall health and well-being",
        category: "Salad"
    },
    {
        _id: "2",
        name: "Veg salad",
        image: food_2,
        price: 18,
        description: "Food provides essential nutrients for overall health and well-being",
        category: "Salad"
    }, {
```

Component Structure

Food display

- Displays food items from food_list via dynamic mapping.
- Category-based filtering for browsing specific types of food.
- Efficient rendering using unique keys for each item.

Food display and implementation on our page



Back end

Server Setup

Configuration:
Centralized in a server.jsfile, where:
Required packages are imported.
Middleware (e.g., JSON parsing, CORS) is set up.
API routes are defined for modularity.

- **Frameworks Used:**
- **Node.js: Enables JavaScript on the server side.**
- **Express.js: Simplifies HTTP request handling and server development.**

API Implementation

User Authentication:
Router: Authentication APIs are defined in userRoute.js, providing endpoints for:
User Registration: Handles new user creation.
Login: Validates credentials and issues tokens.

Security: Ensures safe handling of user data with proper validation.

Code

db.js

backend > config > db.js > ...

```
import mongoose from "mongoose";

export const connectDB = async() => {
    await mongoose.connect('mongodb+srv://huyworkandcontact:huynguyen2305@cluster0.bblpo.mongodb.net/food-del').then(()=>console.log("DB Connected"));
}
```

backend > models > userModel.js > ...

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
    name: {type:String,required:true},
    email:{type:String,required:true,unique:true},
    password:{type:String,required:true},
    cartData:{type:Object,default:{}}
},{minimize:false})

const userModel = mongoose.models.user || mongoose.model('user', userSchema);
export default userModel;
```

```
//login user
const loginUser = async (req,res) =>{
    const {email,password} = req.body;
    try {
        const user = await userModel.findOne({email})

        if (!user){
            return res.json({success:false,message:"User doesn't exist"})
        }

        const isMatch = await bcrypt.compare(password, user.password);

        if (!isMatch){
            return res.json({success:false,message:"Invalid password"})
        }

        const token = createToken(user._id);
        res.json({success:true,message:"Logged in successfully",token})

    } catch (error) {
        console.log(error);
        res.json({success:false,message:"Error logging in"})
    }
}

const createToken = (id) => {
    return jwt.sign({id},process.env.JWT_SECRET)
}
```

Code

```js
import express from "express"
import { loginUser, registerUser } from "../controllers/UserController.js"

const userRouter = express.Router()

userRouter.post("/register", registerUser)
userRouter.post("/login", loginUser)

export default userRouter;
```

```js
//register user
const registerUser = async (req,res) => {
    const {name,password,email} = req.body;
    try {
        //checking is user exists
        const exis = await userModel.findOne({email})
        if(exis){
            return res.json({success:false, message:"User already exists"})
        }

        //validating email format and password
        if(!validator.isEmail(email)){
            return res.json({success:false, message:"Please enter valid email"
        }

        if(password.length<5){
            return res.json({success:false,message:"Please enter a stronger pa
        }

        //hashing user password
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(password, salt);

        const newUser = new userModel({
            name:name,
            email:email,
            password:hashedPassword
        })

        const user = await newUser.save()
        const token = createToken(user._id)
        res.json({success:true, token});

    } catch (error) {
        console.log(error);
        res.json({success:false, message:"Error"})
    }
```

Key Features of the Web Application

01
User Authentication:
Secure user registration and login using JSON Web Tokens (JWT).
Protected routes accessible only to authenticated users.

02
Food Item Management:
Categorized food items with detailed information (name, description, price, image).
Users can browse and add items to their cart.

03
Shopping Cart Functionality:
Allows users to add, view, and manage cart items, including quantity adjustments.
Cart state is managed globally for a smooth experience.

04
Responsive Design:
Optimized for both desktop and mobile devices using CSS media queries and flexible layouts.

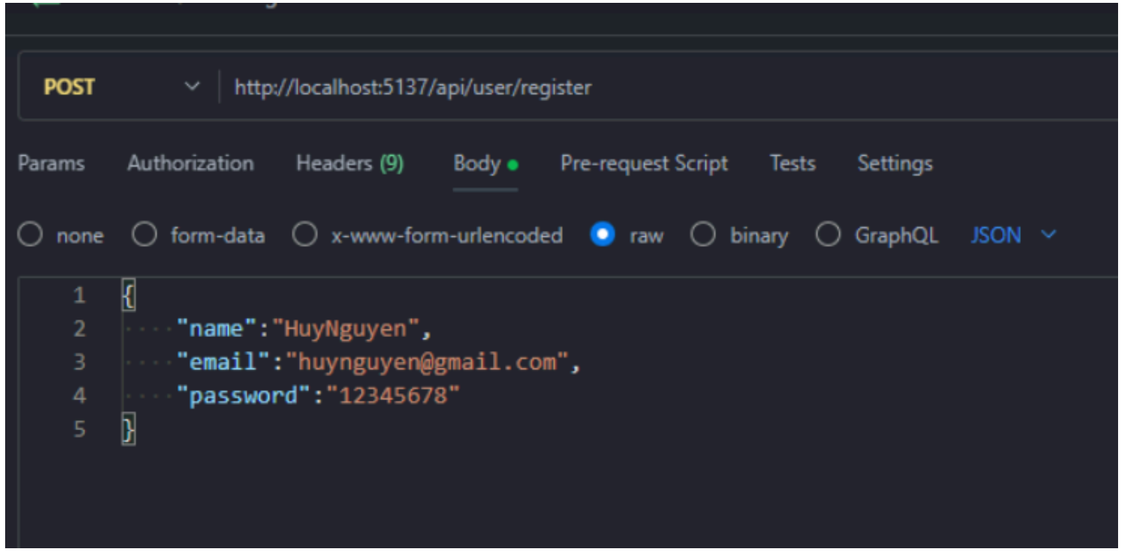
05
Data Visualization:
Clear display of food items with images and descriptions for easy navigation and selection.

06
Error Handling and User Feedback:
Robust error handling during authentication and data submission.
User-friendly feedback for actions like login success or registration errors.

Testing and Debugging

Overview:

- Postman is a tool for testing APIs by sending requests and inspecting responses.
- Supports HTTP methods: GET, POST, PUT, DELETE.
- Steps for Testing:
- Enter API URL (e.g., http://localhost:5137/api/user/register).
- Select the HTTP method (e.g., POST).
- Set Headers (e.g., Content-Type: application/json).
- Add Request Body (in JSON format).
- Click "Send" and inspect the response (status code, body, headers).



Debugging and Challenges

- Using Postman Collections:
  - Organize requests by feature (e.g., User APIs, Food APIs).
  - Use "Runner" to test workflows or sequences.
- Response Validation:
  - Check status codes: 200 for success, 400 for bad requests.
  - Validate response body structure and values.
- Common Challenges:
  - CORS Errors: Configure backend to allow requests from Postman.
  - Authentication Issues: Ensure proper token inclusion and validation.
- Debugging Tips:
  - Use Postman Console to inspect headers, body, and endpoints.
  - Resolve issues by analyzing error messages in response bodies.

Result - Outcome

What did we achieve ?

Fully Functional Food Delivery Platform processing:

- Developed a responsive and dynamic web application for online food ordering and delivery.

Enhanced User Experience:

- The application provides a clean and intuitive user interface built with React.js, responsive design that adapts to both desktop and mobile devices.

Scalable and Secure Backend:

- The backend system, powered by Node.js and MongoDB, ensures efficient handling of API requests with RESTful services

System Reliability:

- Robust error handling implemented to manage invalid inputs and server-side issues

Future Work

Future Implementation

- Real-time order tracking.
- Admin dashboard for managing food items.
- Promotions and discounts
- Develop a dedicated mobile application for iOS and Android
- Suggest food items based on user preferences and order history

THANK YOU