CSE3026– NoSQL Databases

J Component - Project Report

Review III

Online Food Delivery Webapp using MERN stack

By

20MIA1147	Shashank Pandey
20MIA1081	Gaurav Sharan

M.Tech CSE with Specializaton

Submitted to

Dr.A.Bhuvaneswari,
Assistant Professor Senior,
SCOPE, VIT, Chennai

School of Computer Science and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

September 2023

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computing Science and Engineering

VIT Chennai

Vandalur - Kelambakkam Road, Chennai - 600 127

WINTER SEM 22-23

Worklet details

Programme	M.Tech with Specialization	
Course Name / Code	NOSQL - CSE3086	
Slot	C2	
Faculty Name	Dr.A.Bhuvaneswari	
Digital Assignment	III	
Team Members Name | Reg. No	Shashank Pandey	20MIA1147
	Gaurav Sharan	20MIA1081

Team Members(s) Contributions – Tentatively planned for implementation:

Worklet Tasks	*Contributor's Names*
Dataset Collection	Gaurav Sharan
Preprocessing	Shashank Pandey
Architecture/ Model/ Flow diagram	Shashank Pandey
Model building (suitable algorithm)	Shashank Pandey
Results – Tables, Graphs	Gaurav Sharan
Technical Report writing	Gaurav Sharan
Presentation preparation	Shashank Pandey

ABSTRACT

In the contemporary fast-paced society, there is an escalating demand for efficient and convenient food delivery services. The Food Delivery Application project seeks to fulfil this burgeoning requirement by harnessing the capabilities of contemporary web technologies, particularly the MERN (MongoDB, Express.js, React, and Node.js) stack. This endeavour aims to construct a seamless and user-centric platform that connects hungry patrons with a diverse assortment of dining establishments, enabling them to peruse menus, initiate orders, and have their preferred culinary selections delivered straight to their location. The core facets of the Food Delivery Application encompass a robust backend infrastructure founded on Node.js and Express.js. This backend handles vital functionalities such as user authentication, order processing, and restaurant administration. MongoDB, a NoSQL database system, is employed to store and manage the application's data resourcefully, ensuring speedy access and scalability. On the frontend, the project utilizes React, a prominent JavaScript library, to craft an interactive and dynamic user interface, ensuring a gratifying user journey. This amalgamation of technologies guarantees real-time updates, seamless navigation, and an adaptable design suited for both web and mobile interfaces. Noteworthy attributes of the Food Delivery Application comprise user registration and authentication, effective administration of restaurants and menus, seamless order placement, dynamic real-time order tracking, and a secure payment mechanism. Furthermore, the application incorporates recommendation algorithms to tailor the user experience, enhancing customer contentment. With an unwavering emphasis on user input and continual enhancement, the Food Delivery Application strives to furnish an exceptional dining experience that caters to the unique preferences and choices of its users.

This undertaking not only fulfils the practical necessity for food delivery services but also serves as an instructive opportunity for developers intrigued by the MERN stack. By following the development process of this application, developers can gain valuable insights into the deployment of contemporary web technologies, best practices in application design, and strategies for building a durable and scalable platform. Consequently, the Food Delivery Application project contributes to the ever-evolving domain of online food delivery and web application development.

1. INTRODUCTION

The digital era has ushered in significant transformations across numerous facets of our lives, and the dining experience is no exception. In today's fast-paced society, the demand for efficient and user-friendly food delivery services has surged to unprecedented levels. Consumers now anticipate the luxury of savouring their preferred dishes from the comfort of their homes or workplaces, facilitated by a few taps on their smartphones or clicks on their computers. To meet this escalating demand and adapt to the evolving tastes of consumers, the development of a Food Delivery Application employing the MERN (MongoDB, Express.js, React, and Node.js) stack has evolved from a mere convenience to an essential requirement.

The Food Delivery Application project embodies the fusion of contemporary web technologies with the ever-expanding food industry. As individuals' lives become more hectic and demanding, the necessity for a platform that seamlessly connects customers with a diverse array of restaurants and culinary options has become increasingly vital. This project's primary aim is to construct a solution that effectively bridges the gap between these two facets of the culinary world, delivering a user-friendly and feature-rich experience for both customers and restaurant proprietors.

At its essence, the MERN stack is the driving force behind this initiative. Comprising MongoDB, Express.js, React, and Node.js, the MERN stack represents a formidable amalgamation of technologies that empowers the development of responsive, real-time, and scalable web applications. By harnessing the capabilities of these cutting-edge tools, the Food Delivery Application pledges to provide a dynamic, interactive, and intuitive interface accessible to users across diverse devices and platforms.

This project encompasses several vital elements, with each contributing to a comprehensive food delivery experience. The backend, founded on Node.js and Express.js, oversees critical functions like user authentication, order processing, and restaurant management. MongoDB, a NoSQL database system, adeptly stores and manages the application's data, guaranteeing swift access and scalability to accommodate the needs of a continually expanding user base.

On the frontend, React takes center stage, enabling the creation of an aesthetically pleasing and responsive user interface. Users will have the seamless ability to explore menus, place orders, and monitor their deliveries in real-time, all through an intuitive and captivating application. Additionally, the project incorporates advanced recommendation algorithms to customize the user experience, providing customers with personalized dining choices tailored to their preferences and previous orders.

By responding to the surging demand for food delivery services and offering valuable insights into the development process, the Food Delivery Application employing the MERN stack represents a testament to the ever-evolving realm of online food delivery and web application development. It seeks to simplify the process of ordering and relishing meals while simultaneously empowering developers to harness the potential of contemporary web technologies in crafting innovative and user-centric solutions for the digital age.

2. LITERATURE SURVEY

Sl no	Title	Author / Journal name / Year	Technique	Result
1	OnlineFood Ordering System	Joshi, Umesh et al. / International Journal of Advanced Research in Computer Science / 2022	Web-based system, intuitive interface	Simplifies online ordering forsmall restaurants, bridging the gap in the competitive food industry.
2	Evolutionary food quality and location strategies	He, Zhou et al. / International Journal of Production Economics / 2019	Agent-based model	Customer preferences impactrestaurant food quality decisions; delivery services equalize location decisions.
3	Online Food Order System for Restaurants	Patel, Mayurkumar / 2015	Web platform with Oracle database	Streamline restaurant operations, minimizes labor costs, and offers a user-friendly ordering process.
4	Food Ordering Web Application for Fitnes sFreaks	Tarun Garg, Ms. Meenu Garg, and Dr. Bhoomi Gupta / Year not specified	Web application using unspecified tech	Enhances takeaway orders, userprivacy, and customer engagement through ratings andreviews.
5	Food Ordering Application for Canteenusing ReactNative	Joseph George, Sreenath P, Siddharth K G, Jeswill Paulose INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH	Mobile app with QRcode generation	Reduces canteen order times from 5-10 minutes to approximately 3 minutes,enhancing efficiency and user convenience.

Sl no	Title	Author / Journal name / Year	Technique	Result
6	Design of Web App for Online Food Services	Pathak, Harsh et al. /International Journal forResearch in Applied Science and Engineering Technology / 2022	MERN stack development	Offers insights into MERN stack technologies and their rolein creating dynamic web applications.
7	Using MongoDB to Understand Encryption in NoSQL database	Byali, Ramesh / Year notspecified	Encryption methods in MongoDB	Focuses on enhancing MongoDB's data securitythrough encryption, access control, and privacy access policies.
8	A review on various aspectsof MongoDB databases	Chauhan, Anjali / Int. J. Eng. Res. Sci. Technol / 2019	Review of MongoDB's features andissues	Highlights MongoDB's strengths and challenges in comparison to traditional relational databases.
9	Exploration of the MERN Stack for Full-Stack Development	Yogesh Baiskar1, Priyas Paulzagade2 , Krutik Koradia3 , Pramod Ingole4 ,Dhiraj Shirbhate5	Overview of MERNstack technologies	Provides insights into front-end and back-end development using the MERN stack for creating dynamic web applications.

[1] The Online Food Ordering System in this paper is designed for small restaurants, providing an affordable solution without extensive custom software development. It offers a flexible system that allows restaurant staff to easily manage their website, especially the menu, through an intuitive interface. The website dynamically updates to reflect real-time changes, making it user-friendly. Registered users can quickly navigate the menu, add items to their orders, and specify delivery preferences. Placed orders are promptly displayed for efficient processing. This system simplifies the ordering process, bridging the gap between small-scale and established businesses in the competitive food industry.

[2] In the competitive online-to-offline (O2O) food ordering and delivery market, independent restaurants strive to attract customers amid fierce competition, with food quality and location being crucial factors. To understand restaurant behaviors in this context, we introduce an agent-based model (AOFOM) involving customers, restaurants, and online food ordering platforms. Our research highlights the significant influence of customer preferences on restaurant food quality decisions. Interestingly, the impact of restaurant location decisions on customer waiting times is mitigated by online delivery services. We also investigate the characteristics of top-performing restaurants and the effects of different delivery policies. Efficient operations management is essential in this thriving O2O food ordering and delivery market, which presents both opportunities and challenges for restaurants.

[3] The "FOODTOPIA" mobile app revolutionizes traditional canteen ordering methods by offering a digitalized solution. Users can browse menus, place orders with online payments, and receive a QR code bill for in-canteen redemption, reducing wait times and enhancing convenience. While existing food delivery apps like Swiggy offer similar services, FOODTOPIA stands out with its QR code generation for canteen orders. The app simplifies the ordering process, with user-friendly registration, menu selection, and payment options. It also features an efficient admin system managed through a React and Node-based website hosted on Google Cloud (App Engine). The app and website work together seamlessly, significantly reducing order processing time.

[4] The paper underscores the significance of online food ordering in the digital era, emphasizing convenience and accessibility. It highlights advantages like user-friendly interfaces, secure payments, delivery time predictions, and GPS tracking. Customer feedback through ratings and reviews is vital for service improvement. The paper also discusses benefits for restaurants, such as cost savings and customer retention, while acknowledging potential challenges. It offers insights into web app development for online food services, providing a comprehensive view of the industry's impact on customers and businesses.

[5] This study explores encryption methods for enhancing data security in MongoDB, a NoSQL database system. It introduces a privacy access policy with user credentials encryption to ensure robust security and high-speed performance. The research emphasizes the importance of securing data fields in NoSQL databases without compromising performance. Various encryption algorithms, both asymmetric and symmetric, such as DES, triple DES, RSA, AES, ECC, BLOWFISH, and RC5, are evaluated for secure data transmission. Access control in NoSQL data storage is highlighted, aiming to balance data sharing and privacy protection, especially for sensitive information. MongoDB implementation uses BSON for data storage, and the study discusses data flow, authentication, and secure data handling, protecting data both at rest and in transit

[6] This paper offers a comprehensive review of MongoDB, a popular NoSQL database, highlighting its key features and flexibility. It compares MongoDB to MySQL, showcasing its efficiency in basic operations but acknowledging potential issues like reliability and schema challenges. The paper suggests future research directions, making it a valuable resource for understanding MongoDB's capabilities and areas for improvement.

[9] This paper is a comprehensive exploration of contemporary web technologies, with a primary emphasis on the MERN stack (MongoDB, Express, React, and Node.js). It aims to demystify the realm of full-stack development, offering valuable insights into the various components and processes involved. The paper starts by introducing front-end development, encompassing HTML, CSS, and JavaScript, along with popular frameworks like React.js. It then delves into back-end development, highlighting the significance of technologies such as Node.js and Express.js. Database management, including relational and non-relational databases like MongoDB, is also discussed. The MERN stack's pivotal role in modern web development is emphasized, with a focus on its constituent technologies and their synergy in creating dynamic and responsive web applications.

3. Dataset and Database tools

Custom Dataset Creation: Our project, the Food Delivery System, involves the creation of a custom dataset tailored to the requirements of our application. This dataset is unique to our platform and consists of various categories of data essential for the food delivery ecosystem.

1. **Data Categories:** The custom dataset encompasses a wide range of data categories, including but not limited to:
2. **User Data:** Information about registered users, including profiles, preferences, and order history.
3. **Restaurant Data:** Details about partner restaurants, such as menus, locations, ratings, and special offerings.
4. **Menu Items:** Comprehensive listings of dishes, complete with descriptions, prices, and availability status.

5. **Order Records:** Records of customer orders, including timestamps, order contents, and delivery details.

Database tool – MongoDB

Steps involved in creating a food delivery system using the MERN stack and MongoDB:

1. **Create a MongoDB Database:**

We can use the MongoDB Compass GUI or the MongoDB command-line tool to create a new database, such as "food_delivery_db."

2. **Create Collections:**

We can create collections for different entities in the food delivery system, such as "restaurants," "menus," "orders," and "users."

3. **Define Collection Schemas:**

For each collection, we define the schema by specifying the fields that will be stored. For example, in the "restaurants" collection, we might have fields like name, location, cuisine, and ratings.

4. **Load Data:**

We can load data into the collections. This can be done manually by inserting documents or by using data loader tools, scripts, or import methods.

5. **Create the MERN Stack Application:**

We can set up a MERN stack application, which consists of:

- A frontend React app for the user interface.
- A backend Node.js app using Express.js for server-side logic.
- The MongoDB database for data storage.

6. **Connect the Frontend to Backend:**

We can establish a connection between the frontend and backend using RESTful API endpoints. The frontend sends HTTP requests to the backend to retrieve or manipulate data.

7. **Connect the Backend to the Database:**

We use the MongoDB driver for Node.js (e.g., mongoose) to connect the backend to the MongoDB database. We provide the database connection string, authenticate, and handle database operations.

8. **Implement Food Delivery Features:**

In our application, we can implement features such as:

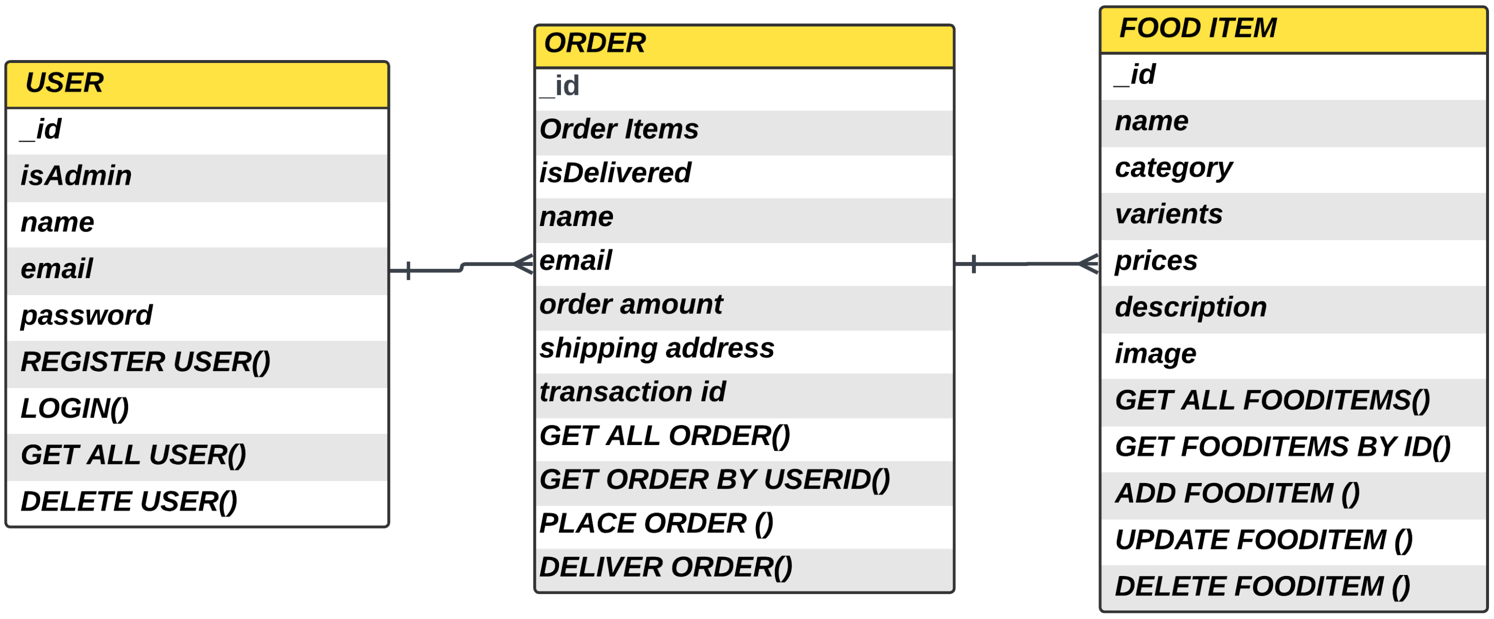
- User registration and login for customers and restaurants.
- Restaurant search and menu browsing.
- User authentication and authorization to ensure secure access.
- Order placement and cart management.
- Payment processing, which may involve integrating payment gateways.
- Order tracking and notification handling.
- Admin panels for restaurant management and order fulfillment.

4. Algorithms / Techniques description

1. **MERN stack:** The MERN stack is a popular JavaScript stack that consists of MongoDB, Express.js, React, and Node.js. MongoDB is a NoSQL database that is used to store data efficiently. Express.js is a web framework that is used to create RESTful APIs. React is a JavaScript library that is used to build user interfaces. Node.js is a runtime environment that is used to run JavaScript code on the server.
2. **User authentication:** User authentication is important to ensure that only authorized users can access the app. There are a variety of ways to implement user authentication, such as using a username and password, social media login, or two-factor authentication.
3. **Database management:** MongoDB is used to store data for the food delivery app. The database schema should be designed to meet the specific needs of the app.
4. **Frontend design:** The frontend of the app should be designed to be user-friendly and responsive. React is a good choice for building a frontend that is both dynamic and interactive.
5. **Order processing:** The order processing workflow should be designed to be efficient and scalable. A queuing system can be used to manage orders and ensure that they are processed in a timely manner.
6. **Responsive design:** The webapp should be designed to be responsive so that it can be used on a variety of devices, including smartphones, tablets, and computers.

7. **Scalability:** The app should be designed to be scalable so that it can handle increased user loads.

5. Block Diagram of the proposed system design



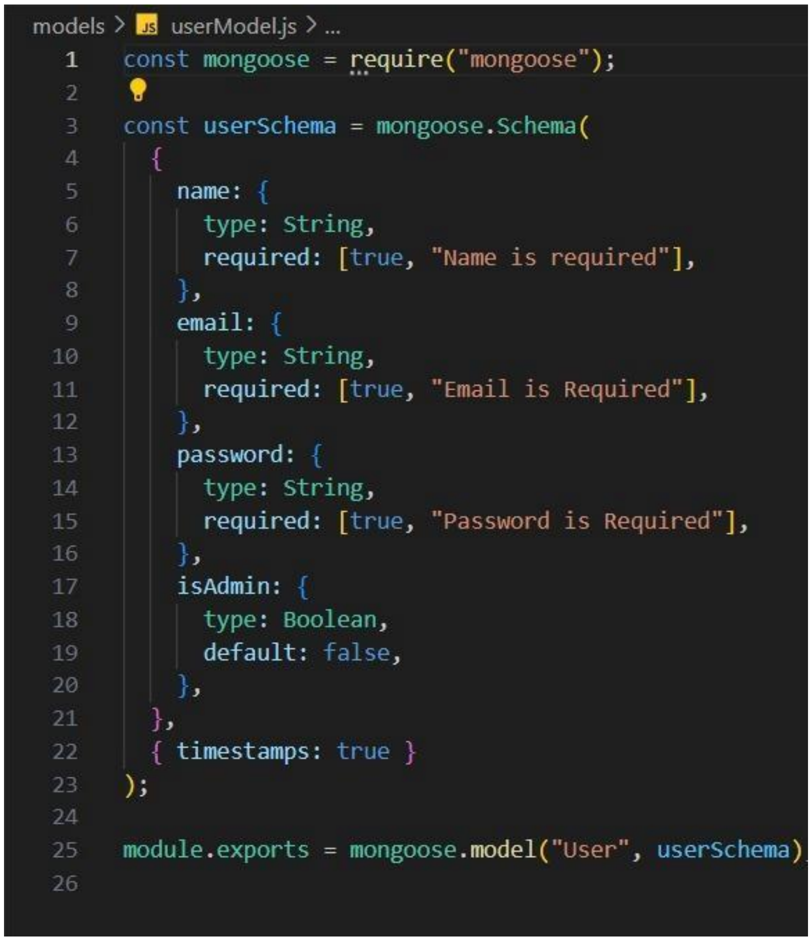
6. Implementation

We have created three schemas

- User
- Food Item
- Order

1. **User schema**

Code:



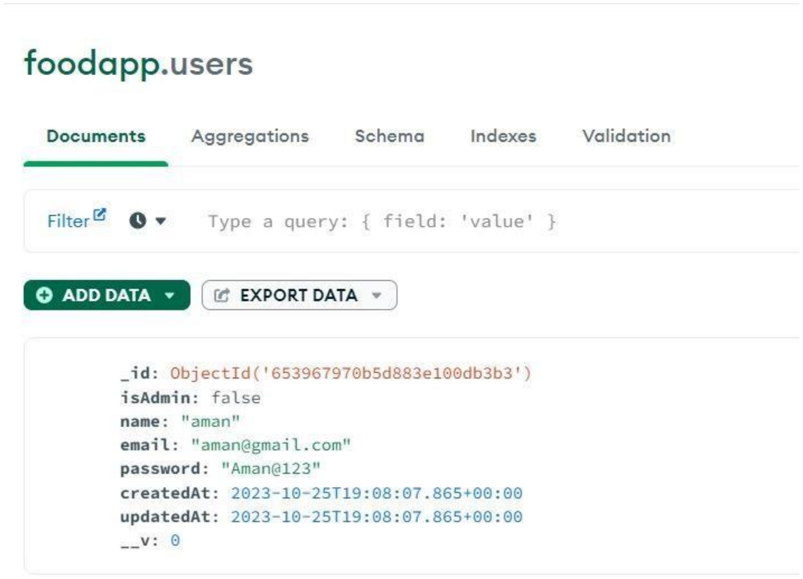
models > userModel.js > ...

```
const mongoose = require("mongoose");

const userSchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "Name is required"],
    },
    email: {
      type: String,
      required: [true, "Email is Required"],
    },
    password: {
      type: String,
      required: [true, "Password is Required"],
    },
    isAdmin: {
      type: Boolean,
      default: false,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("User", userSchema)
```

MongoDB:

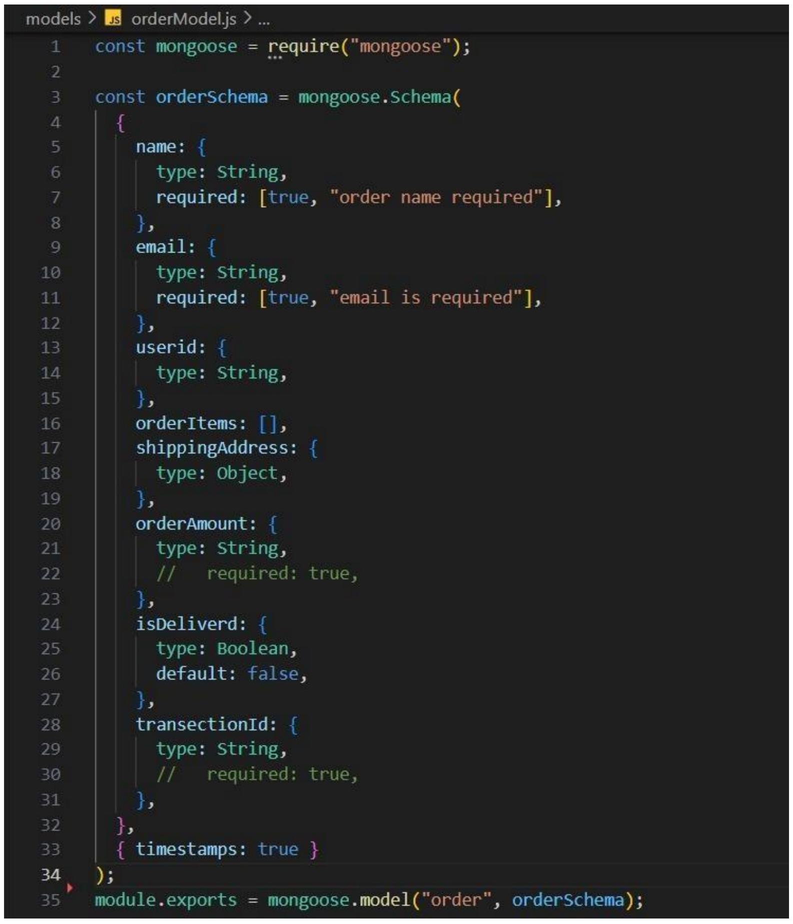


foodapp.users

```
_id: ObjectId('653967970b5d883e100db3b3')
isAdmin: false
name: "aman"
email: "aman@gmail.com"
password: "Aman@123"
createdAt: 2023-10-25T19:08:07.865+00:00
updatedAt: 2023-10-25T19:08:07.865+00:00
__v: 0
```

2. Order schema

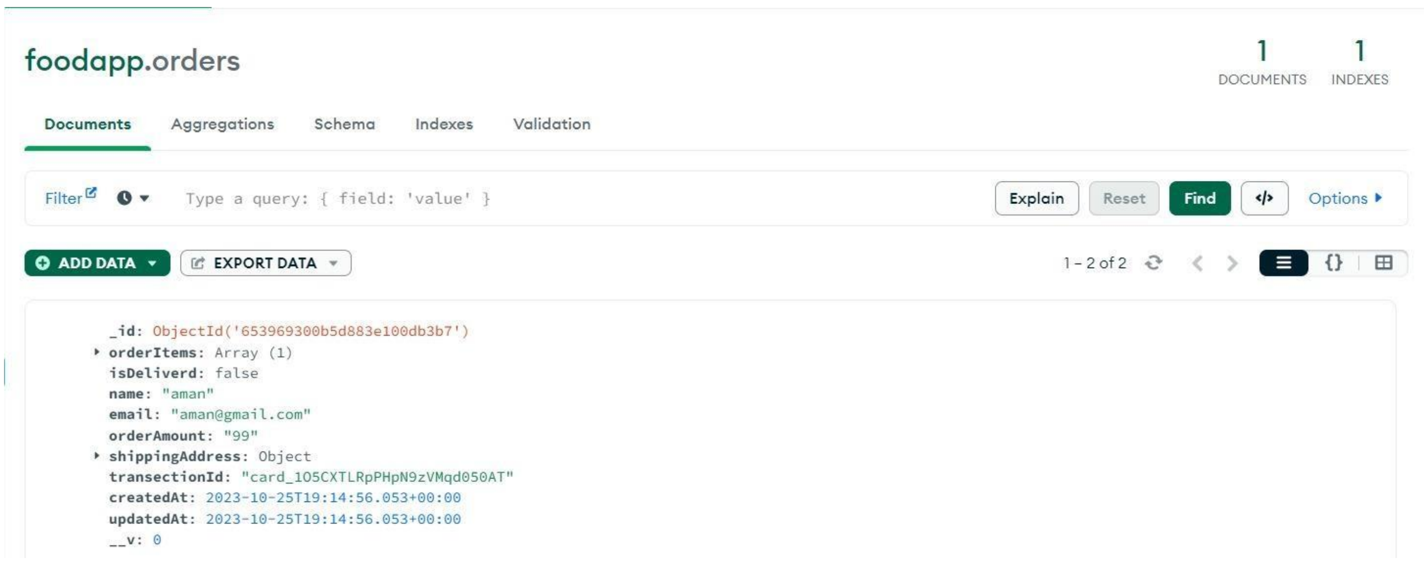
Code:


models > orderModel.js > ...

```
const mongoose = require("mongoose");

const orderSchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "order name required"],
    },
    email: {
      type: String,
      required: [true, "email is required"],
    },
    userid: {
      type: String,
    },
    orderItems: [],
    shippingAddress: {
      type: Object,
    },
    orderAmount: {
      type: String,
      //   required: true,
    },
    isDeliverd: {
      type: Boolean,
      default: false,
    },
    transectionId: {
      type: String,
      //   required: true,
    },
  },
  { timestamps: true }
);
module.exports = mongoose.model("order", orderSchema);
```

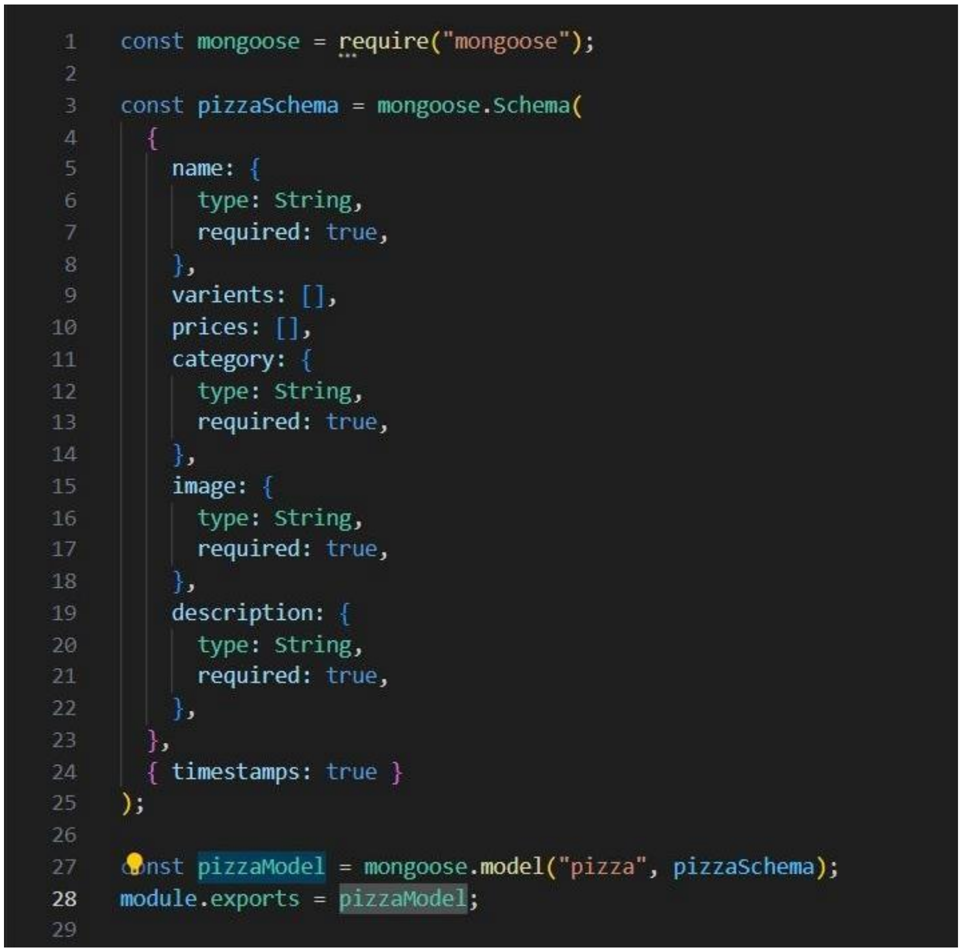
MongoDB:



```
_id: ObjectId('653969300b5d883e100db3b7')
orderItems: Array (1)
isDeliverd: false
name: "aman"
email: "aman@gmail.com"
orderAmount: "99"
shippingAddress: Object
transectionId: "card_105CXTLRpPHpN9zVMqd050AT"
createdAt: 2023-10-25T19:14:56.053+00:00
updatedAt: 2023-10-25T19:14:56.053+00:00
__v: 0
```

3. Food Item schema

Code:

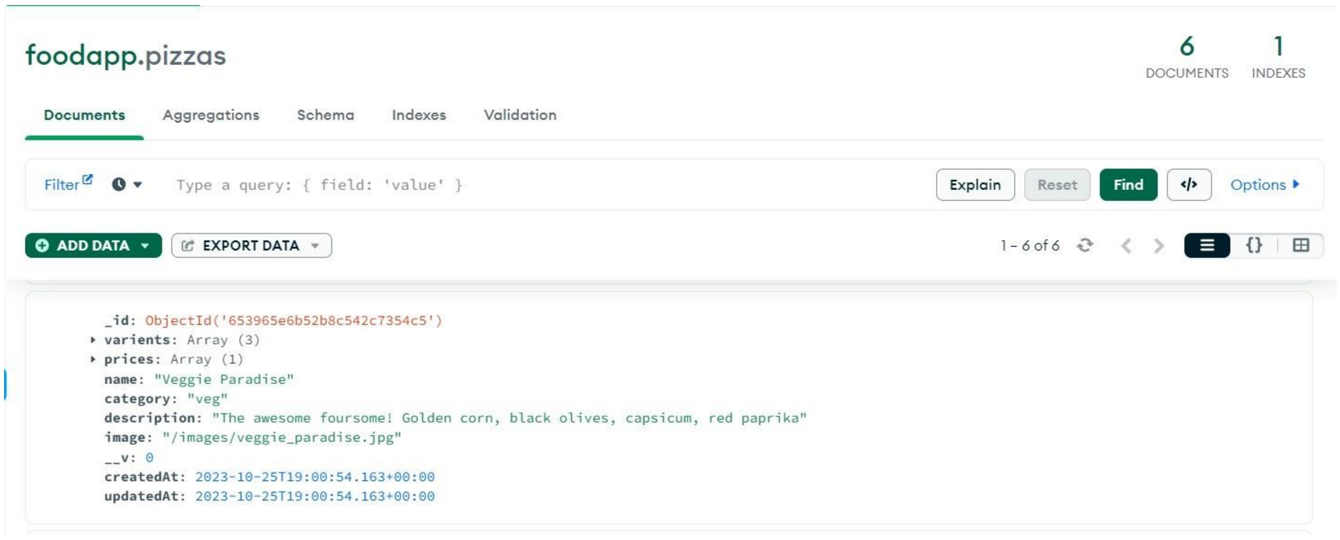


```
const mongoose = require("mongoose");

const pizzaSchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    varients: [],
    prices: [],
    category: {
      type: String,
      required: true,
    },
    image: {
      type: String,
      required: true,
    },
    description: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);

const pizzaModel = mongoose.model("pizza", pizzaSchema);
module.exports = pizzaModel;
```

MongoDB:



foodapp.pizzas

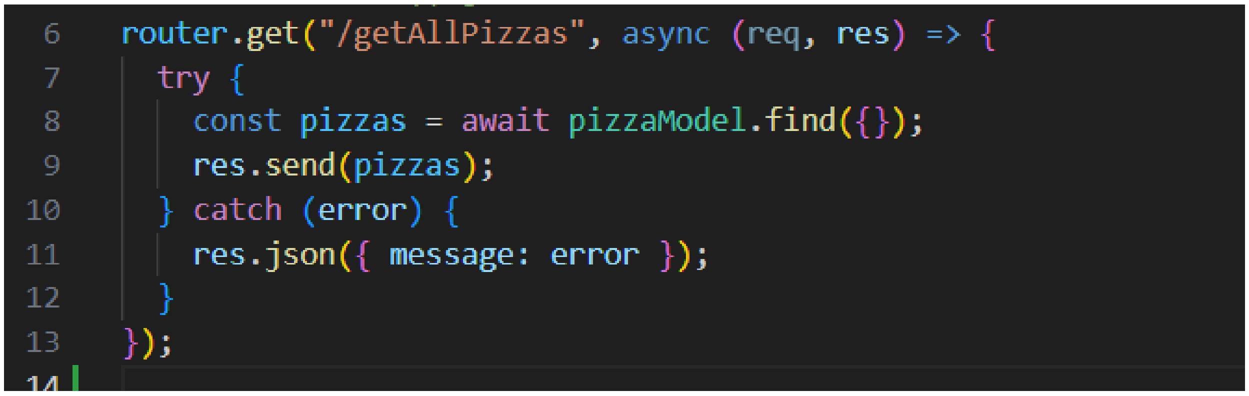
6 DOCUMENTS 1 INDEXES

```
_id: ObjectId('653965e6b52b8c542c7354c5')
▸ varients: Array (3)
▸ prices: Array (1)
name: "Veggie Paradise"
category: "veg"
description: "The awesome foursome! Golden corn, black olives, capsicum, red paprika"
image: "/images/veggie_paradise.jpg"
__v: 0
createdAt: 2023-10-25T19:00:54.163+00:00
updatedAt: 2023-10-25T19:00:54.163+00:00
```

BACKEND FUNCTIONALITIES OF OUR WEBSITE

1. Get Pizza
2. Add Pizza
3. Delete Pizza
4. Get Pizza By ID
5. Update Pizza
6. Get all User
7. Register User
8. Login
9. Delete User
10. Get all order
11. Get Order by User ID
12. Deliver order

- **GET PIZZA**

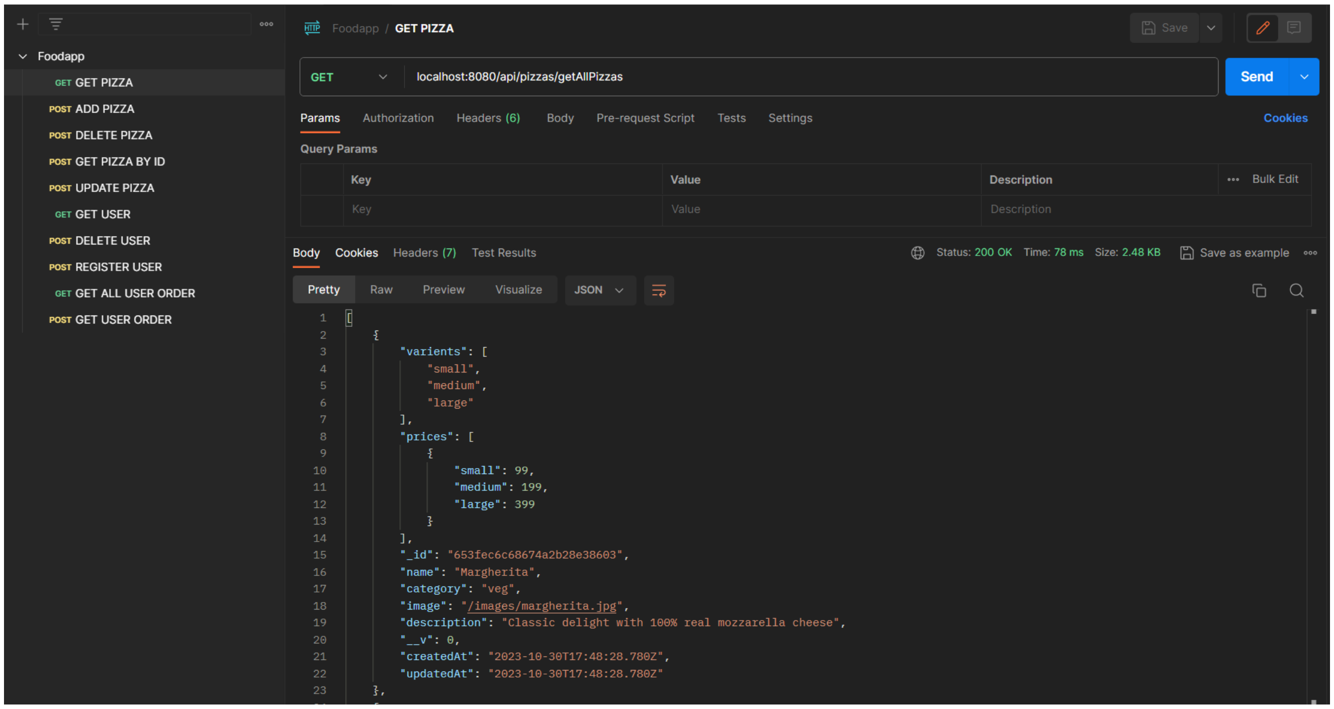


```
router.get("/getAllPizzas", async (req, res) => {
  try {
    const pizzas = await pizzaModel.find({});
    res.send(pizzas);
  } catch (error) {
    res.json({ message: error });
  }
});
```

This Route is responsible for fetching all Pizza records from a database. We have used Postman tool for testing the API routes. Postman is a popular API platform that can be used for a variety of tasks, including:

API Development, API testing, API Documentation.

Testing for GET PIZZA using POSTMAN tool



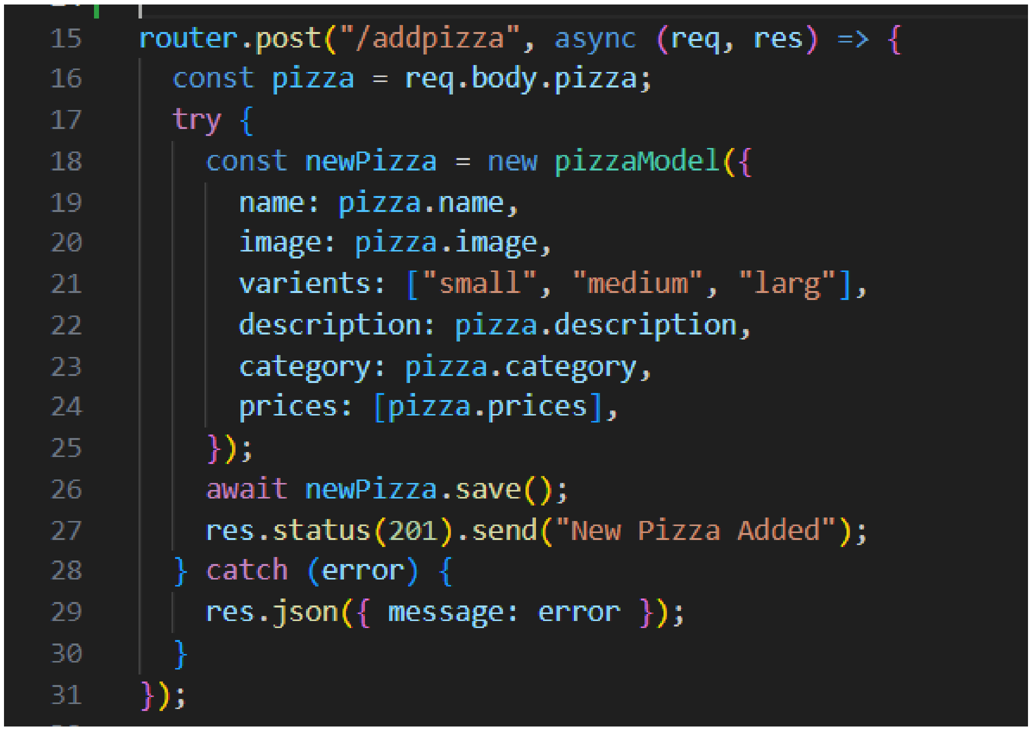
GET localhost:8080/api/pizzas/getAllPizzas

Key	Value	Description
Key	Value	Description

Status: 200 OK Time: 78 ms Size: 2.48 KB

```
[
    {
        "varients": [
            "small",
            "medium",
            "large"
        ],
        "prices": [
            {
                "small": 99,
                "medium": 199,
                "large": 399
            }
        ],
        "_id": "653fec6c68674a2b28e38603",
        "name": "Margherita",
        "category": "veg",
        "image": "/images/margherita.jpg",
        "description": "Classic delight with 100% real mozzarella cheese",
        "__v": 0,
        "createdAt": "2023-10-30T17:48:28.780Z",
        "updatedAt": "2023-10-30T17:48:28.780Z"
    },
```

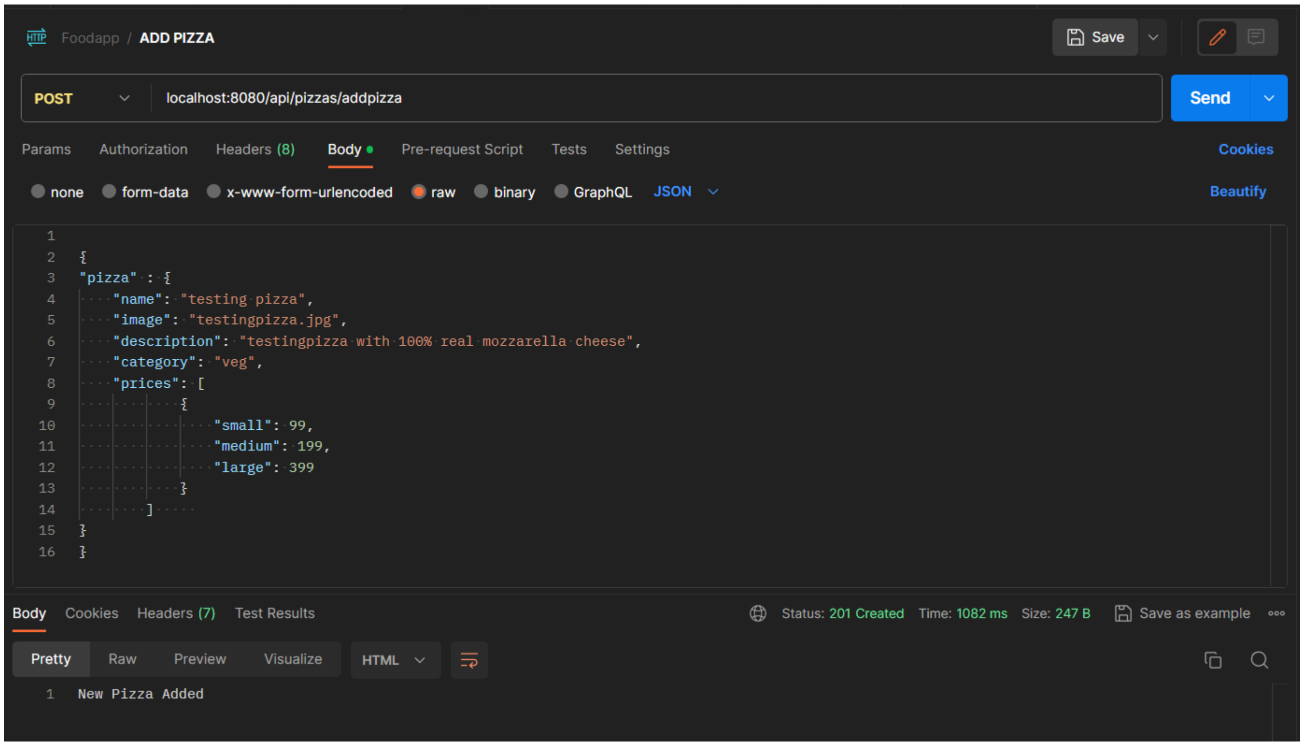
- **ADD PIZZA**



```
router.post("/addpizza", async (req, res) => {
  const pizza = req.body.pizza;
  try {
    const newPizza = new pizzaModel({
      name: pizza.name,
      image: pizza.image,
      varients: ["small", "medium", "larg"],
      description: pizza.description,
      category: pizza.category,
      prices: [pizza.prices],
    });
    await newPizza.save();
    res.status(201).send("New Pizza Added");
  } catch (error) {
    res.json({ message: error });
  }
});
```

This route is responsible for adding a new pizza to the database based on the data provided in the request body.

Postman Testing

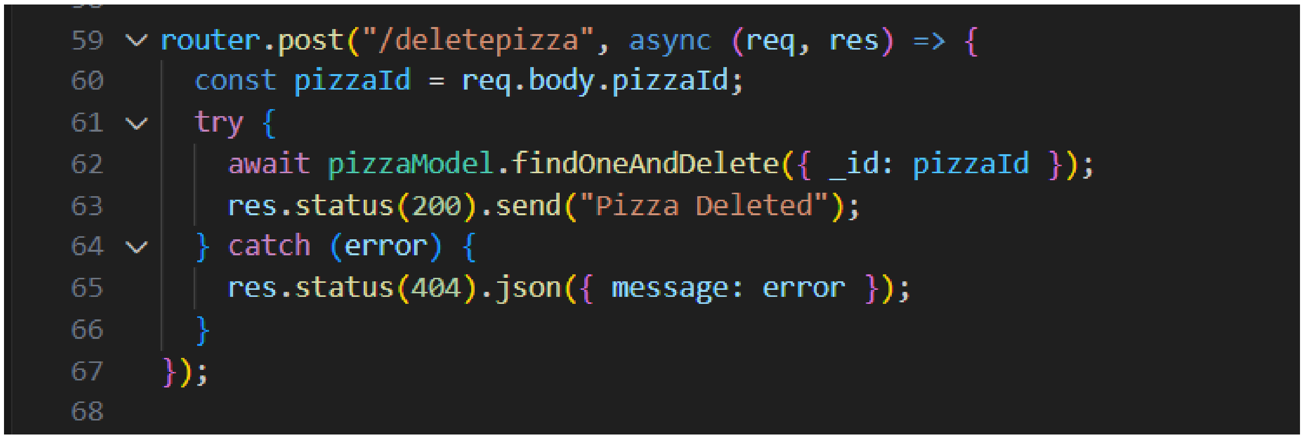


```
{
"pizza" : {
    "name": "testing pizza",
    "image": "testingpizza.jpg",
    "description": "testingpizza with 100% real mozzarella cheese",
    "category": "veg",
    "prices": [
            {
                "small": 99,
                "medium": 199,
                "large": 399
            }
        ]
}
}
```

```
New Pizza Added
```

This json data represents the details of a pizza named "Testing Pizza" with its image, description, category, and pricing information.

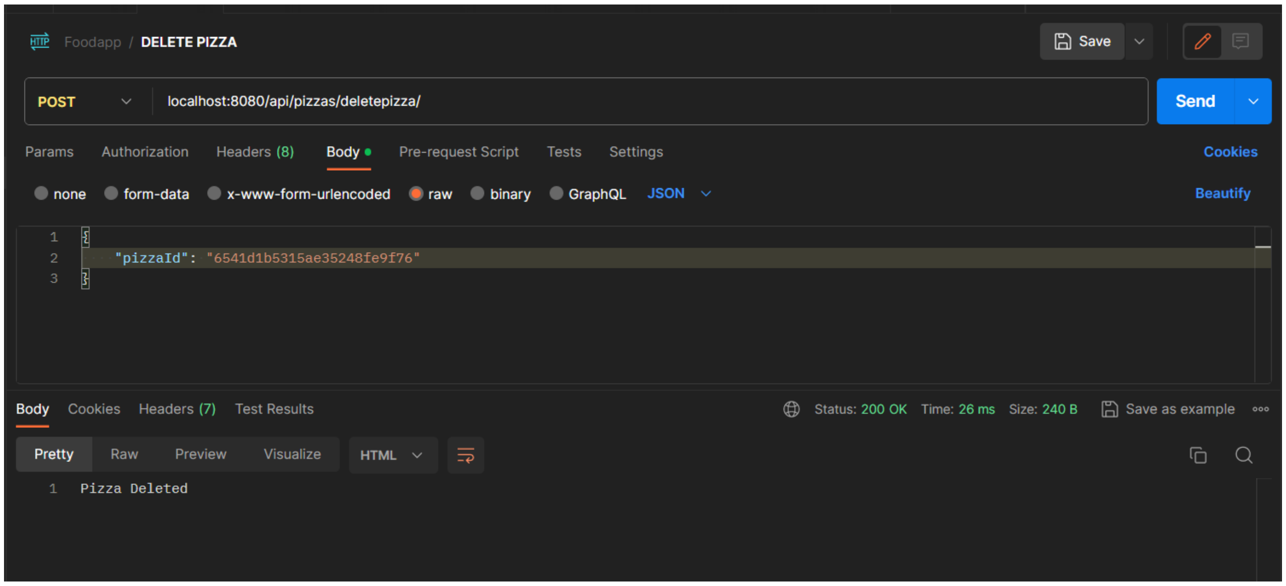
- **DELETE PIZZA**



```
router.post("/deletepizza", async (req, res) => {
  const pizzaId = req.body.pizzaId;
  try {
    await pizzaModel.findOneAndDelete({ _id: pizzaId });
    res.status(200).send("Pizza Deleted");
  } catch (error) {
    res.status(404).json({ message: error });
  }
});
```

This route is responsible for deleting a pizza from the database based on the pizzaId provided in the request body.

Postman Testing



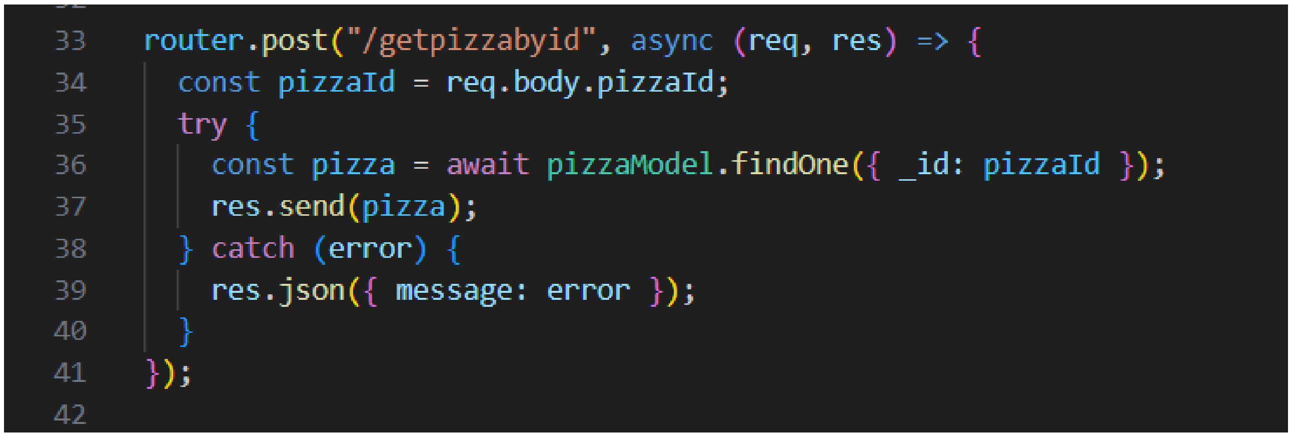
POST localhost:8080/api/pizzas/deletepizza/

```
{
    "pizzaId": "6541d1b5315ae35248fe9f76"
}
```

Status: 200 OK Time: 26 ms Size: 240 B

```
Pizza Deleted
```

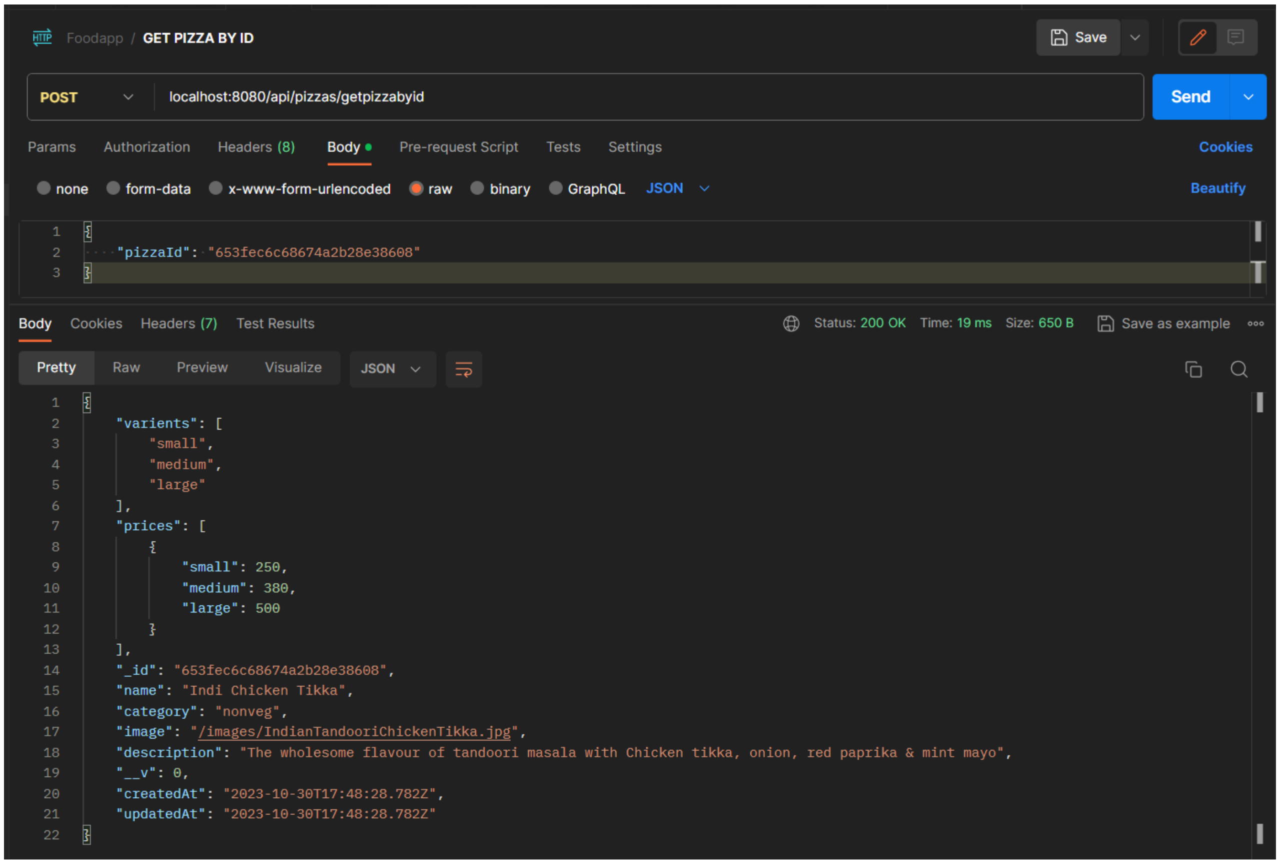
- **GET PIZZA BY ID**



```
router.post("/getpizzabyid", async (req, res) => {
  const pizzaId = req.body.pizzaId;
  try {
    const pizza = await pizzaModel.findOne({ _id: pizzaId });
    res.send(pizza);
  } catch (error) {
    res.json({ message: error });
  }
});
```

This route is responsible for fetching a specific pizza from the database based on the pizzaId provided in the request body.

Postman testing



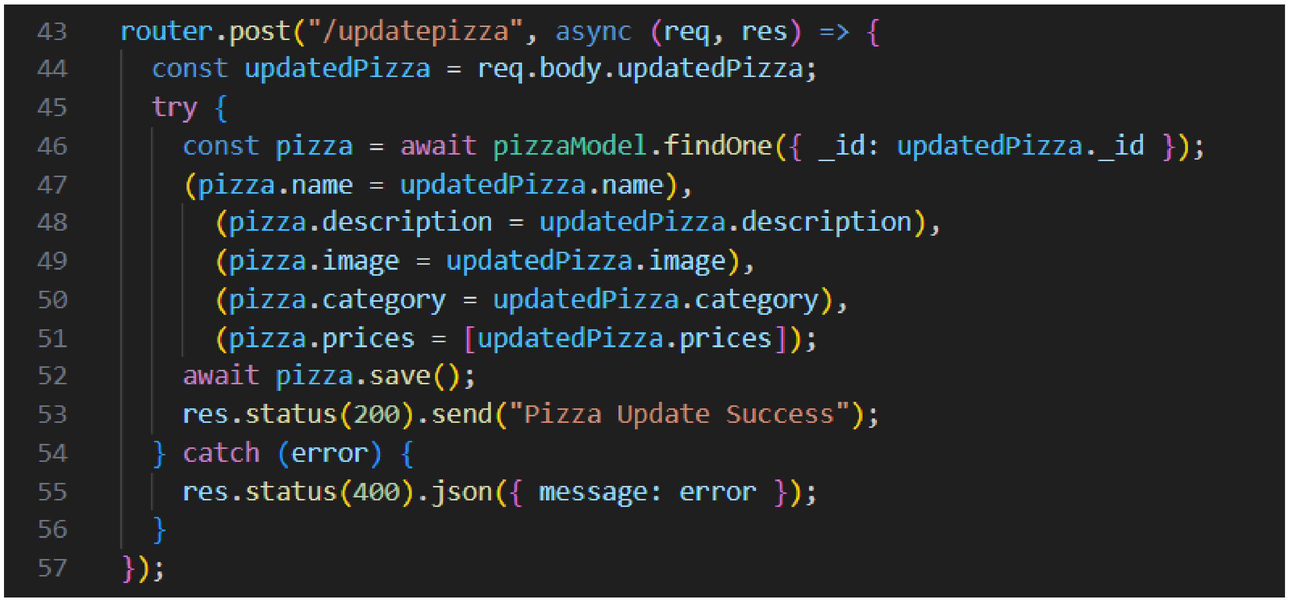
POST localhost:8080/api/pizzas/getpizzabyid

```
{
    "pizzaId": "653fec6c68674a2b28e38608"
}
```

Status: 200 OK Time: 19 ms Size: 650 B

```
{
    "varients": [
        "small",
        "medium",
        "large"
    ],
    "prices": [
        {
            "small": 250,
            "medium": 380,
            "large": 500
        }
    ],
    "_id": "653fec6c68674a2b28e38608",
    "name": "Indi Chicken Tikka",
    "category": "nonveg",
    "image": "/images/IndianTandooriChickenTikka.jpg",
    "description": "The wholesome flavour of tandoori masala with Chicken tikka, onion, red paprika & mint mayo",
    "__v": 0,
    "createdAt": "2023-10-30T17:48:28.782Z",
    "updatedAt": "2023-10-30T17:48:28.782Z"
}
```

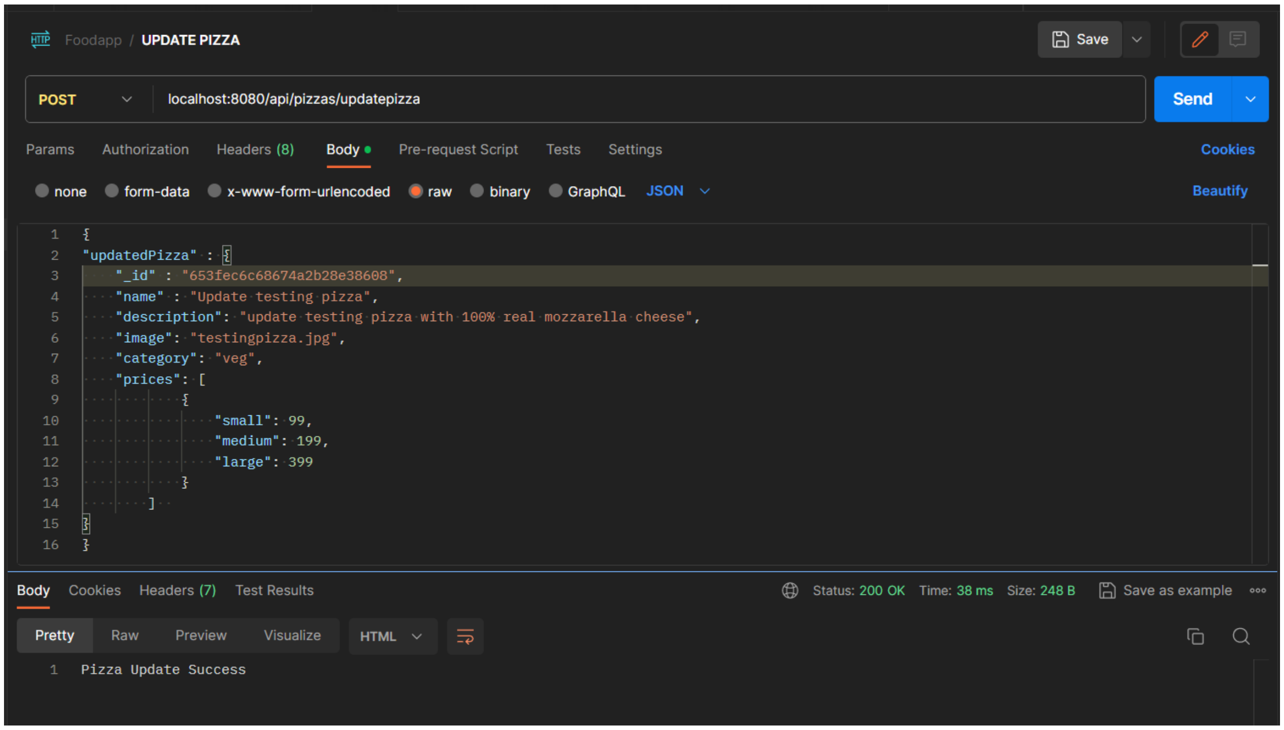
- **UPDATE PIZZA**



```
router.post("/updatepizza", async (req, res) => {
  const updatedPizza = req.body.updatedPizza;
  try {
    const pizza = await pizzaModel.findOne({ _id: updatedPizza._id });
    (pizza.name = updatedPizza.name),
      (pizza.description = updatedPizza.description),
      (pizza.image = updatedPizza.image),
      (pizza.category = updatedPizza.category),
      (pizza.prices = [updatedPizza.prices]);
    await pizza.save();
    res.status(200).send("Pizza Update Success");
  } catch (error) {
    res.status(400).json({ message: error });
  }
});
```

This route is responsible for updating an existing pizza record in the database based on the data provided in the request body.

Postman Testing


Foodapp / UPDATE PIZZA
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (7) Test Results
Save as example
Pretty Raw Preview Visualize HTML

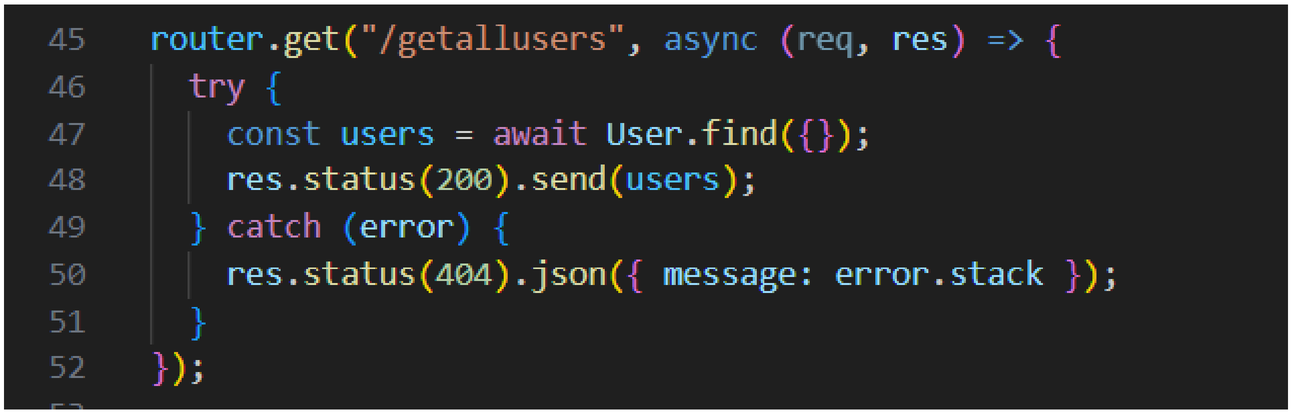
```
POST localhost:8080/api/pizzas/updatepizza
```

```
{
"updatedPizza" : {
    "_id" : "653fec6c68674a2b28e38608",
    "name" : "Update testing pizza",
    "description": "update testing pizza with 100% real mozzarella cheese",
    "image": "testingpizza.jpg",
    "category": "veg",
    "prices": [
            {
                "small": 99,
                "medium": 199,
                "large": 399
            }
        ]
}
}
```

Status: 200 OK Time: 38 ms Size: 248 B

```
Pizza Update Success
```

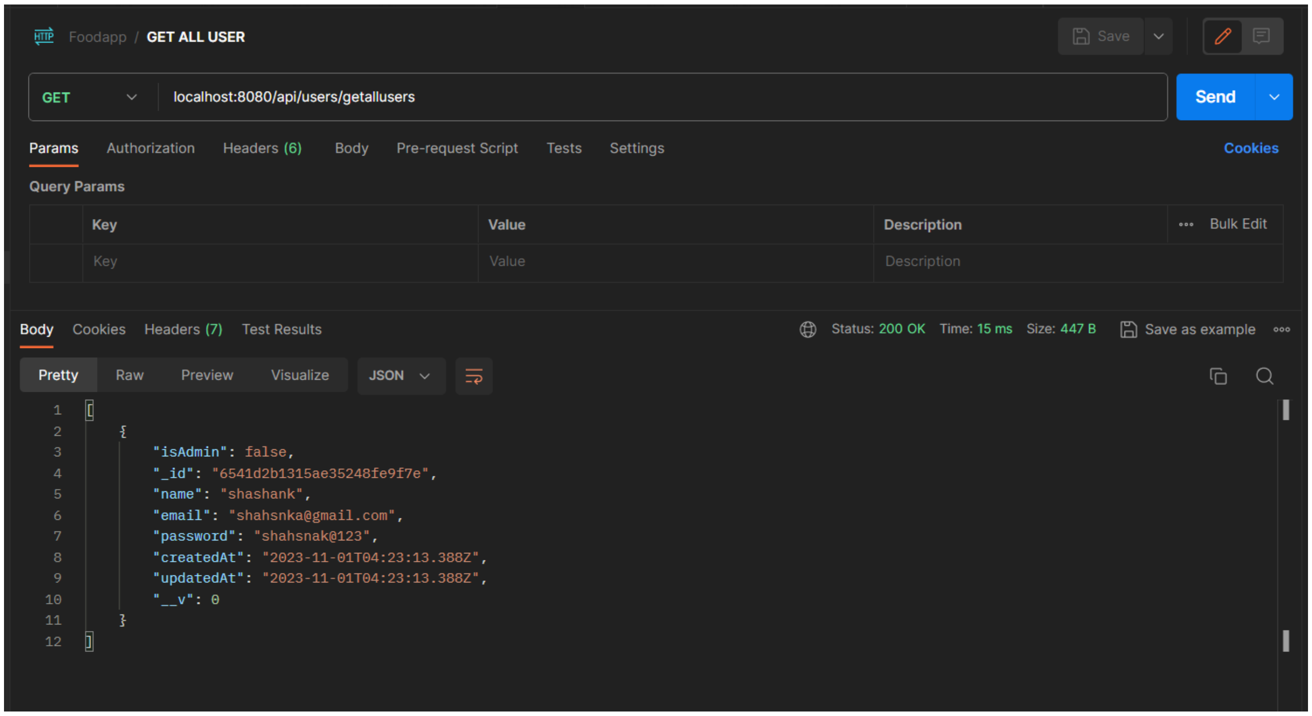
- **GET ALL USER**



```
router.get("/getallusers", async (req, res) => {
  try {
    const users = await User.find({});
    res.status(200).send(users);
  } catch (error) {
    res.status(404).json({ message: error.stack });
  }
});
```

This route is responsible for fetching and returning all user records from the database

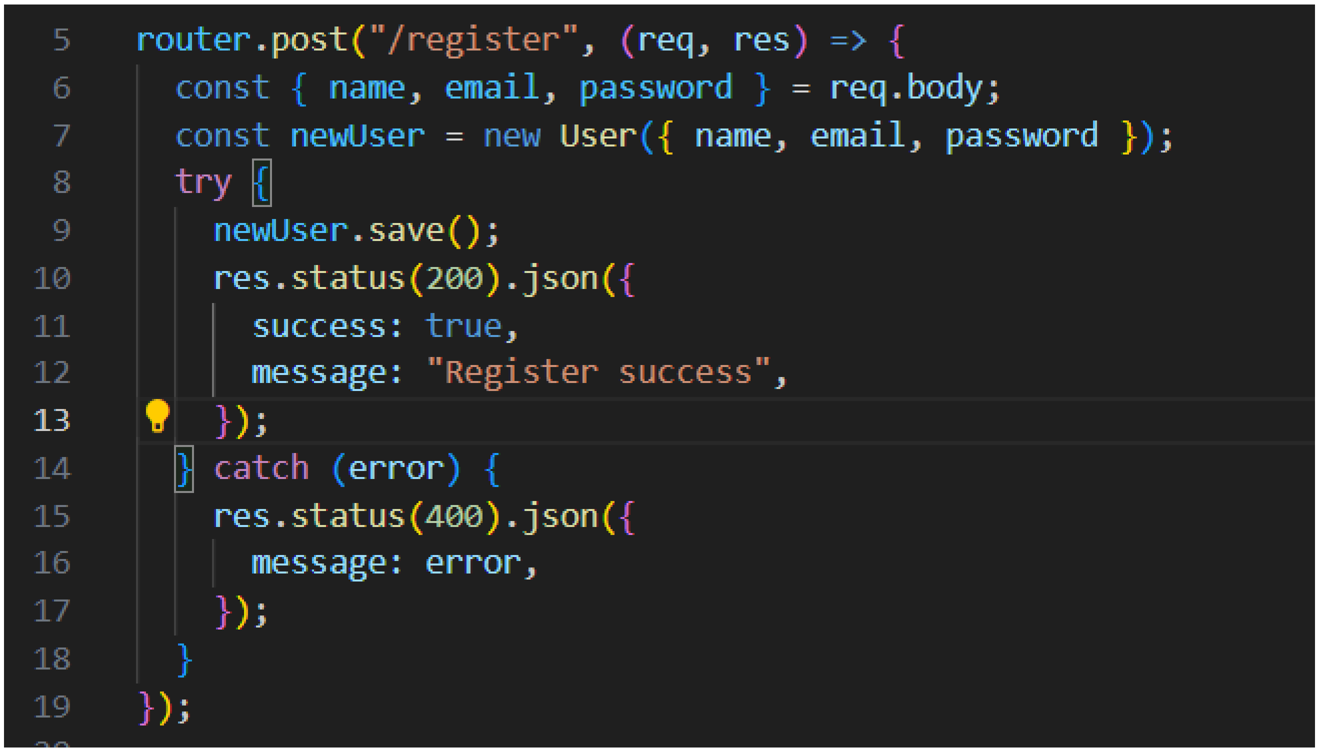
Postman testing



localhost:8080/api/users/getallusers

```
[
    {
        "isAdmin": false,
        "_id": "6541d2b1315ae35248fe9f7e",
        "name": "shashank",
        "email": "shahsnka@gmail.com",
        "password": "shahsnak@123",
        "createdAt": "2023-11-01T04:23:13.388Z",
        "updatedAt": "2023-11-01T04:23:13.388Z",
        "__v": 0
    }
]
```

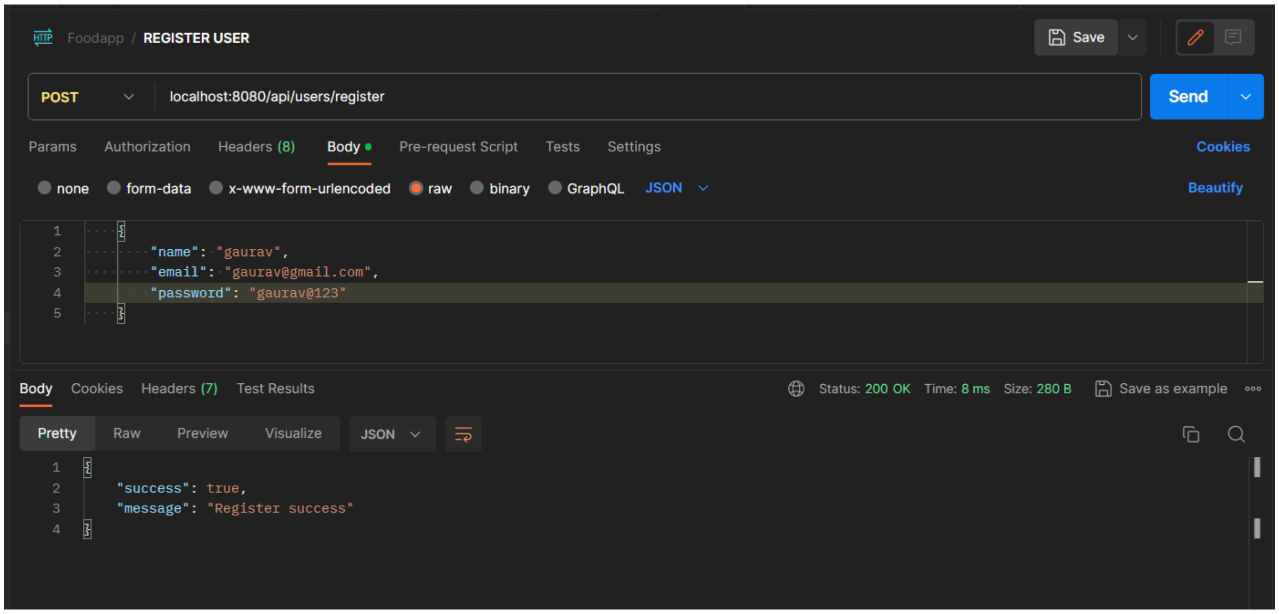
- **REGISTER USER**



```
router.post("/register", (req, res) => {
  const { name, email, password } = req.body;
  const newUser = new User({ name, email, password });
  try {
    newUser.save();
    res.status(200).json({
      success: true,
      message: "Register success",
    });
  } catch (error) {
    res.status(400).json({
      message: error,
    });
  }
});
```

This route is responsible for registering a new user based on the data provided in the request body.

Postman testing:



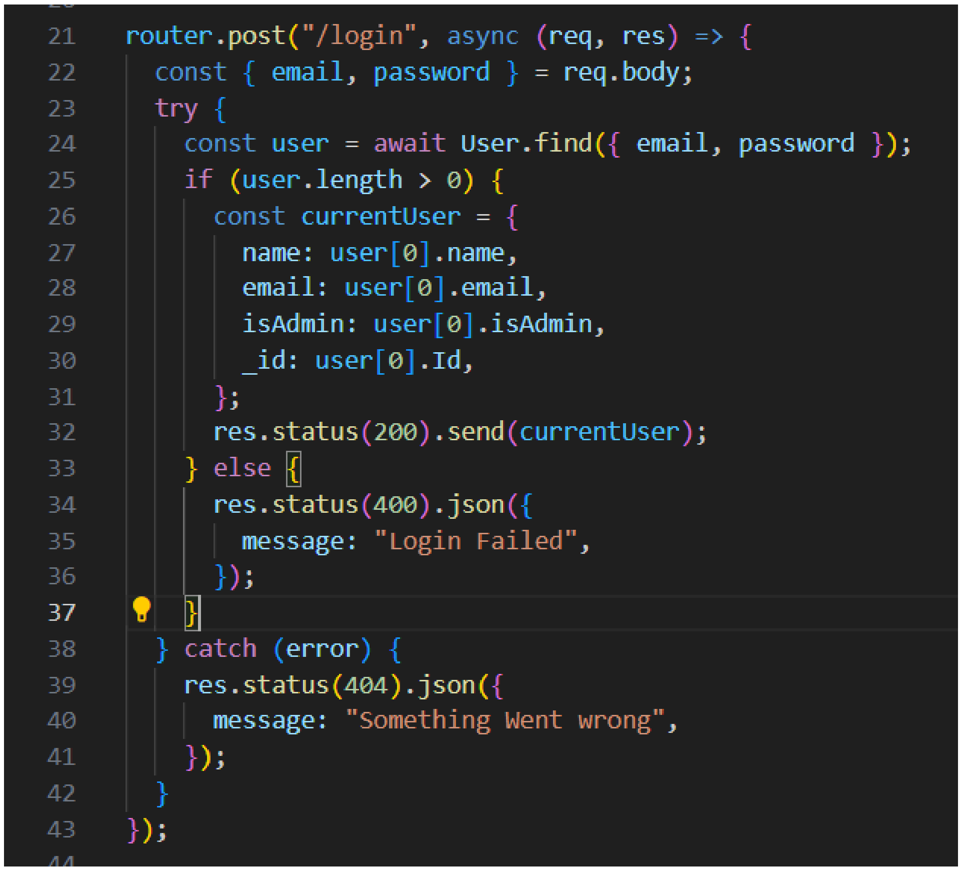
localhost:8080/api/users/register

```
{
    "name": "gaurav",
    "email": "gaurav@gmail.com",
    "password": "gaurav@123"
}
```

Status: 200 OK Time: 8 ms Size: 280 B

```
{
    "success": true,
    "message": "Register success"
}
```

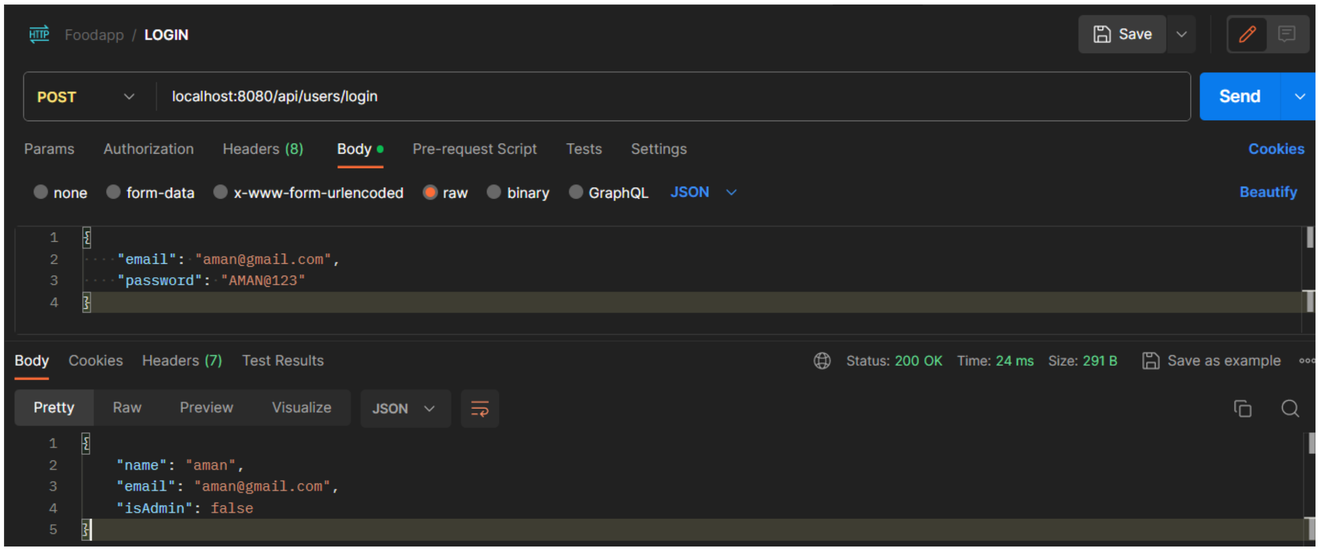
- **LOGIN USER**



```
router.post("/login", async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await User.find({ email, password });
    if (user.length > 0) {
      const currentUser = {
        name: user[0].name,
        email: user[0].email,
        isAdmin: user[0].isAdmin,
        _id: user[0].Id,
      };
      res.status(200).send(currentUser);
    } else {
      res.status(400).json({
        message: "Login Failed",
      });
    }
  } catch (error) {
    res.status(404).json({
      message: "Something Went wrong",
    });
  }
});
```

This route is responsible for handling user login based on the provided email and password.

Postman testing



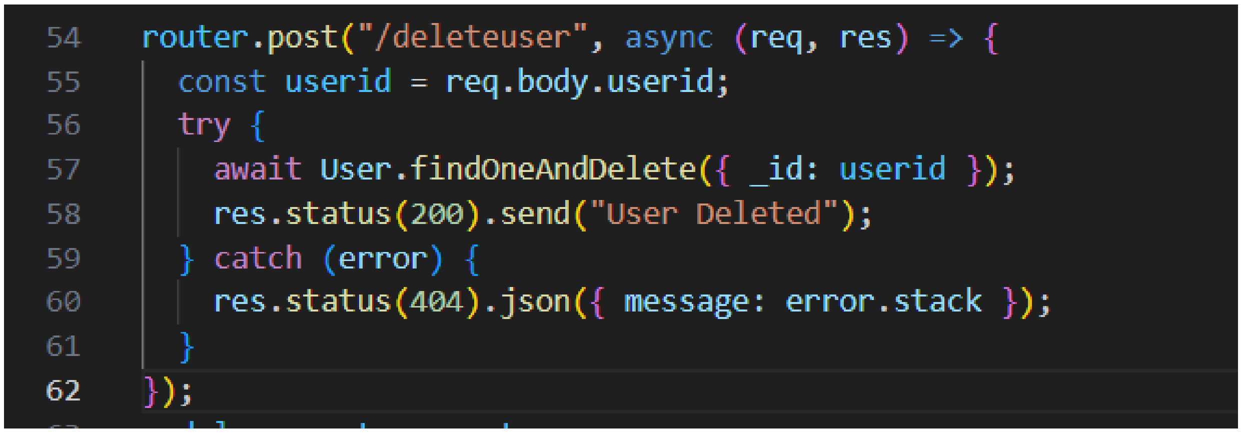
POST localhost:8080/api/users/login

```
{
    "email": "aman@gmail.com",
    "password": "AMAN@123"
}
```

Body — Status: 200 OK Time: 24 ms Size: 291 B

```
{
    "name": "aman",
    "email": "aman@gmail.com",
    "isAdmin": false
}
```

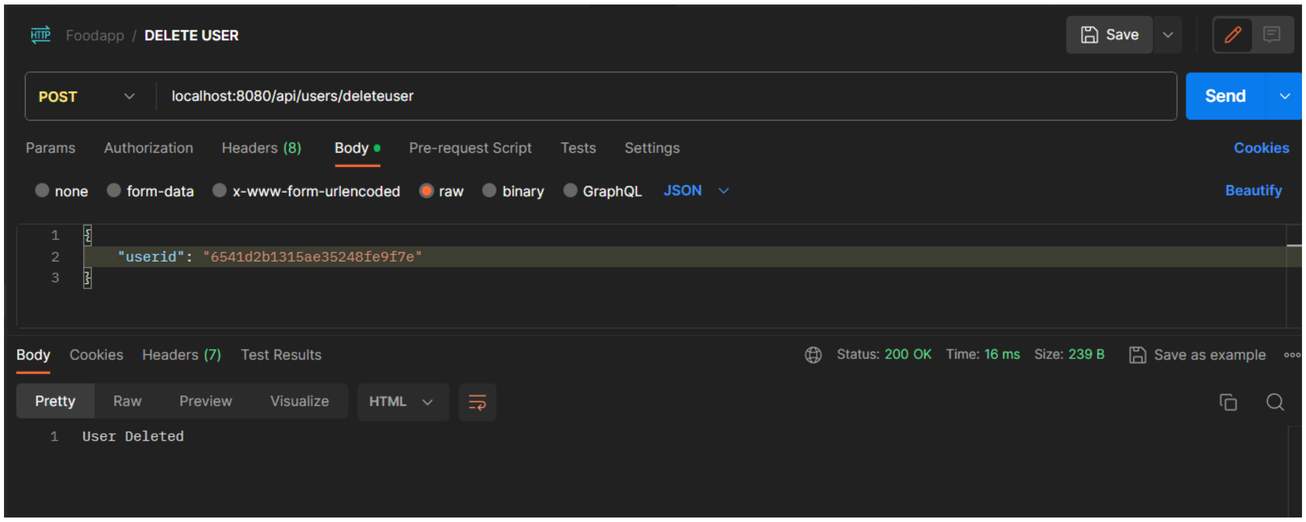
- **DELETE USER**



```
router.post("/deleteuser", async (req, res) => {
  const userid = req.body.userid;
  try {
    await User.findOneAndDelete({ _id: userid });
    res.status(200).send("User Deleted");
  } catch (error) {
    res.status(404).json({ message: error.stack });
  }
});
```

This route is responsible for deleting a user from the database based on the userid provided in the request body.

Postman Testing:



POST localhost:8080/api/users/deleteuser

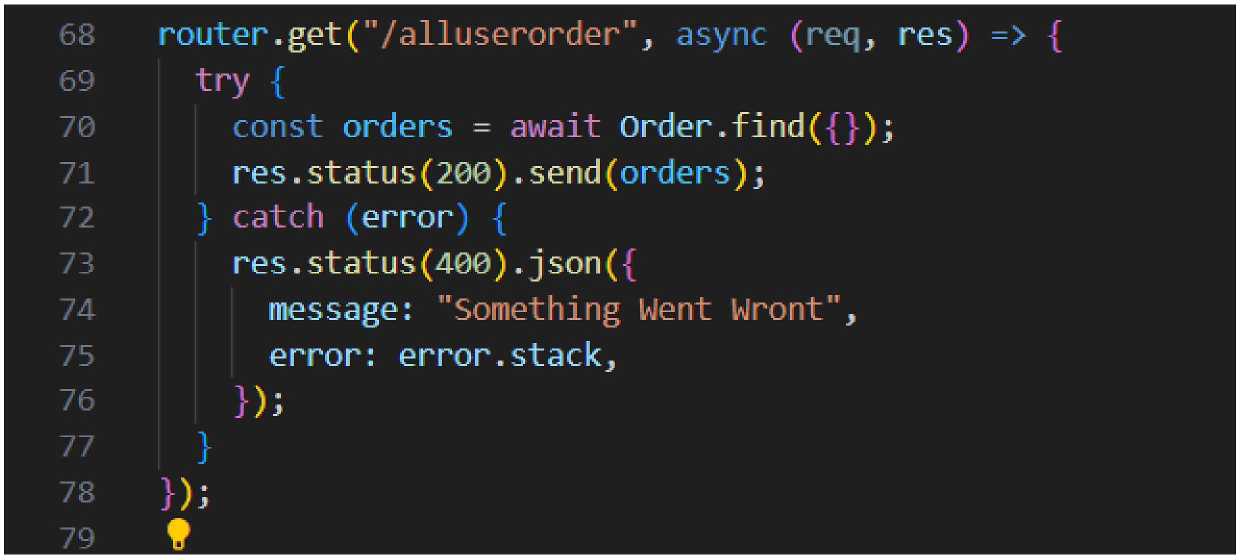
none ○ form-data ○ x-www-form-urlencoded [✓ raw] ○ binary ○ GraphQL JSON

```
{
    "userid": "6541d2b1315ae35248fe9f7e"
}
```

Status: 200 OK Time: 16 ms Size: 239 B

```
User Deleted
```

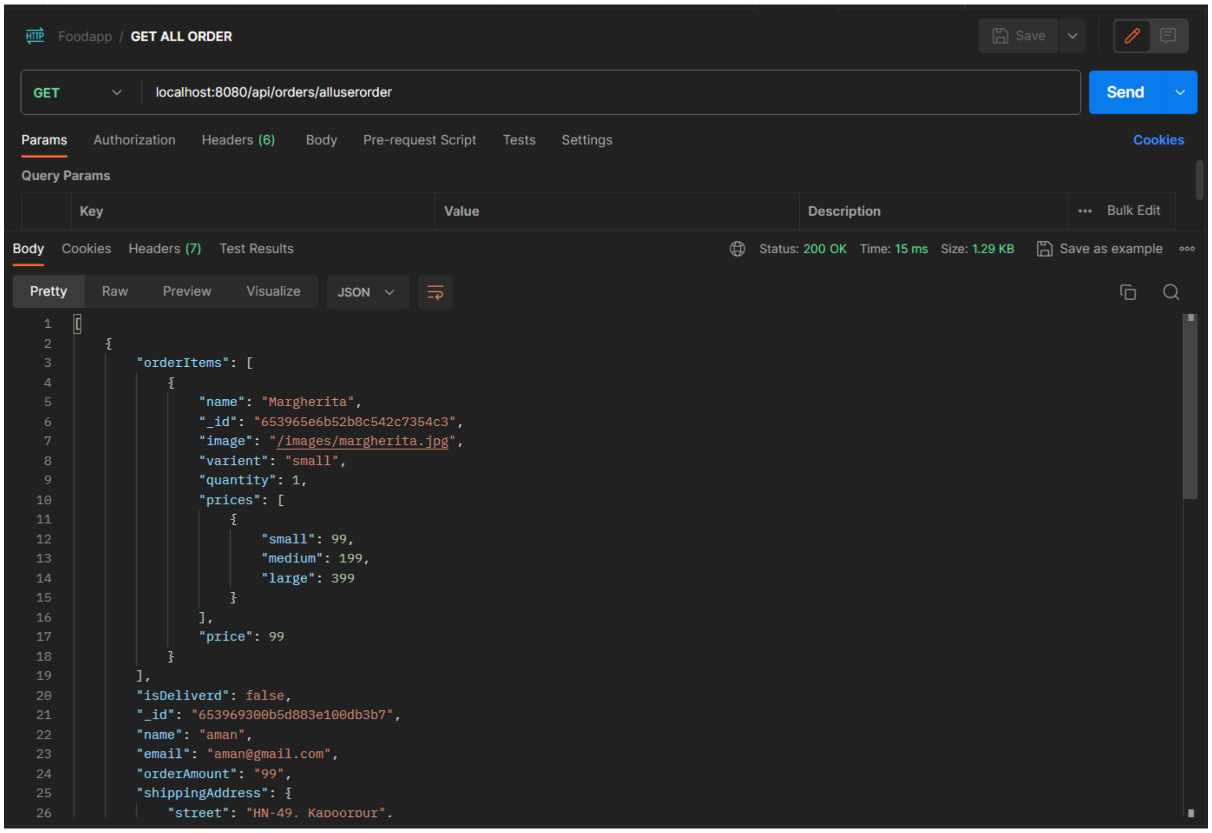
- **GET ALL USER**



```
router.get("/alluserorder", async (req, res) => {
  try {
    const orders = await Order.find({});
    res.status(200).send(orders);
  } catch (error) {
    res.status(400).json({
      message: "Something Went Wront",
      error: error.stack,
    });
  }
});
```

This route is responsible for fetching and returning all user orders from the database.

Postman testing

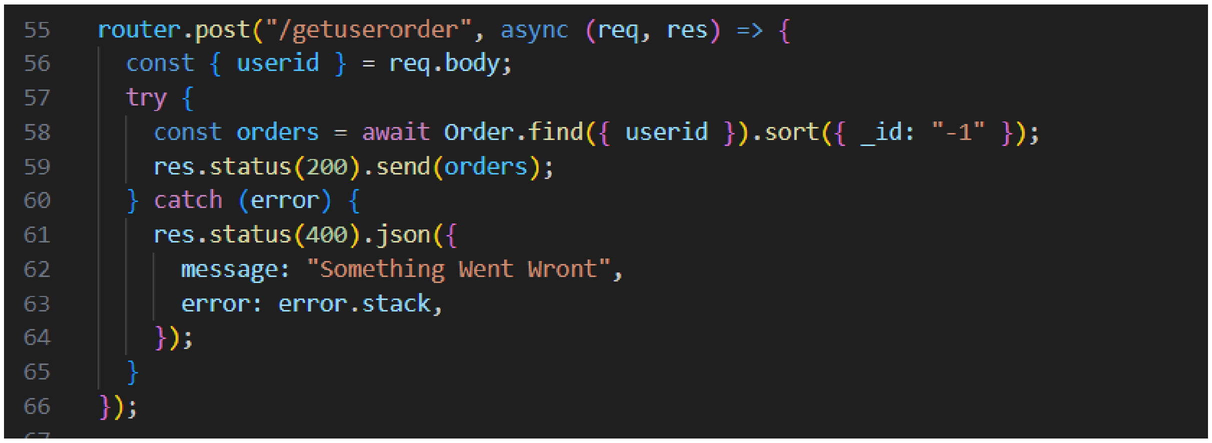


localhost:8080/api/orders/alluserorder

Status: 200 OK Time: 15 ms Size: 1.29 KB

```
[
    {
        "orderItems": [
            {
                "name": "Margherita",
                "_id": "653965e6b52b8c542c7354c3",
                "image": "/images/margherita.jpg",
                "varient": "small",
                "quantity": 1,
                "prices": [
                    {
                        "small": 99,
                        "medium": 199,
                        "large": 399
                    }
                ],
                "price": 99
            }
        ],
        "isDeliverd": false,
        "_id": "653969300b5d883e100db3b7",
        "name": "aman",
        "email": "aman@gmail.com",
        "orderAmount": "99",
        "shippingAddress": {
            "street": "HN-49, Kapoorpur",
```

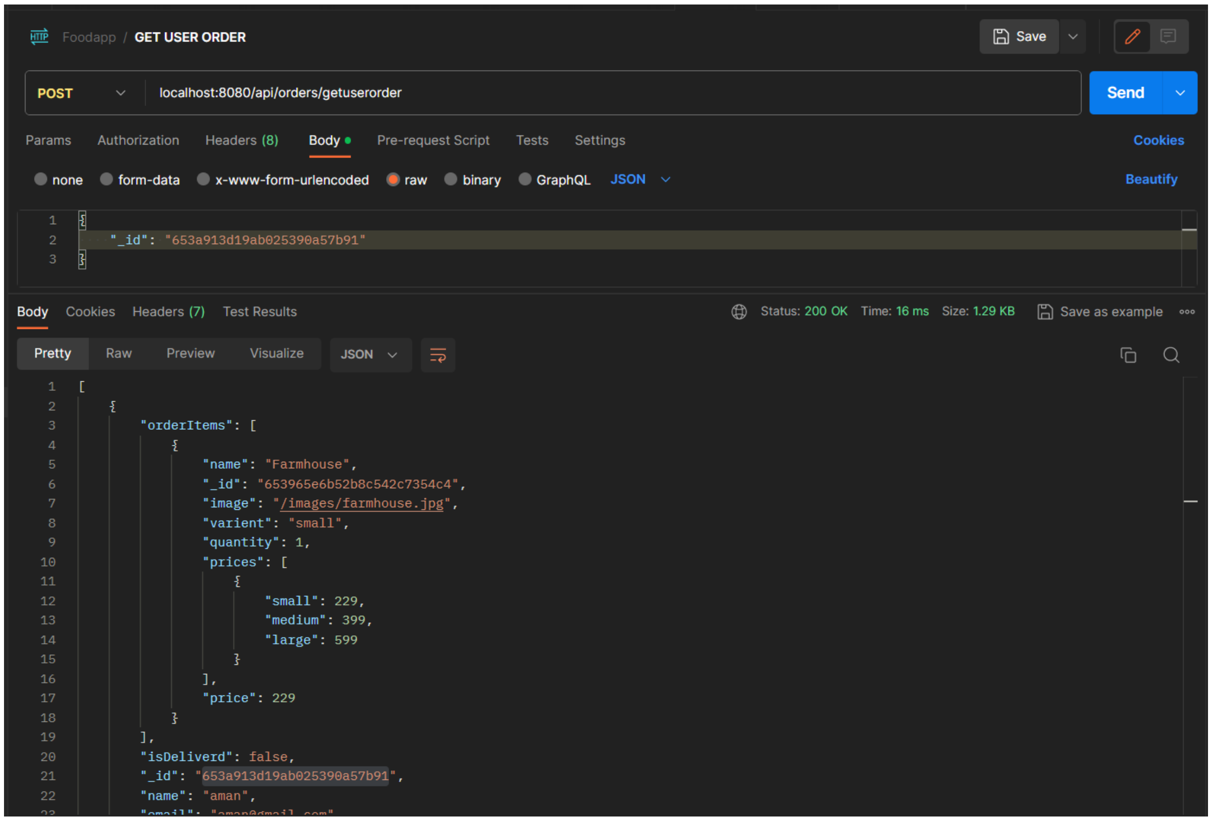
- **GET ORDER BY USER ID**



```
router.post("/getuserorder", async (req, res) => {
  const { userid } = req.body;
  try {
    const orders = await Order.find({ userid }).sort({ _id: "-1" });
    res.status(200).send(orders);
  } catch (error) {
    res.status(400).json({
      message: "Something Went Wront",
      error: error.stack,
    });
  }
});
```

This route is responsible for fetching and returning user-specific orders based on the userid provided in the request body.

Postman testing



POST localhost:8080/api/orders/getuserorder

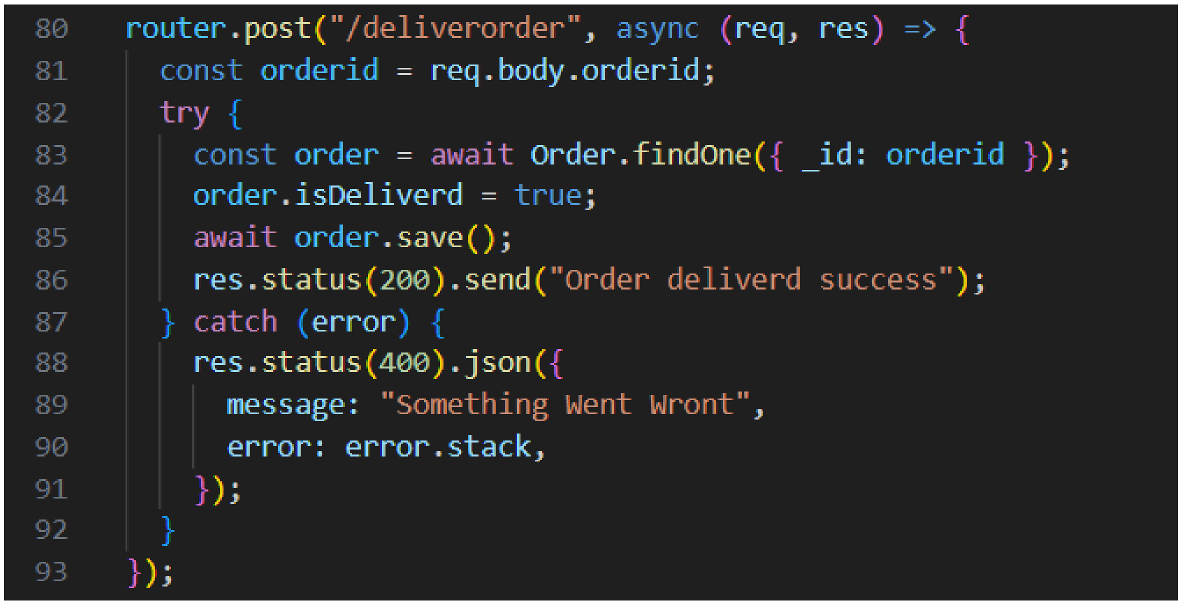
```
{
    "_id": "653a913d19ab025390a57b91"
}
```

Status: 200 OK Time: 16 ms Size: 1.29 KB

JSON

```
[
    {
        "orderItems": [
            {
                "name": "Farmhouse",
                "_id": "653965e6b52b8c542c7354c4",
                "image": "/images/farmhouse.jpg",
                "varient": "small",
                "quantity": 1,
                "prices": [
                    {
                        "small": 229,
                        "medium": 399,
                        "large": 599
                    }
                ],
                "price": 229
            }
        ],
        "isDeliverd": false,
        "_id": "653a913d19ab025390a57b91",
        "name": "aman",
```

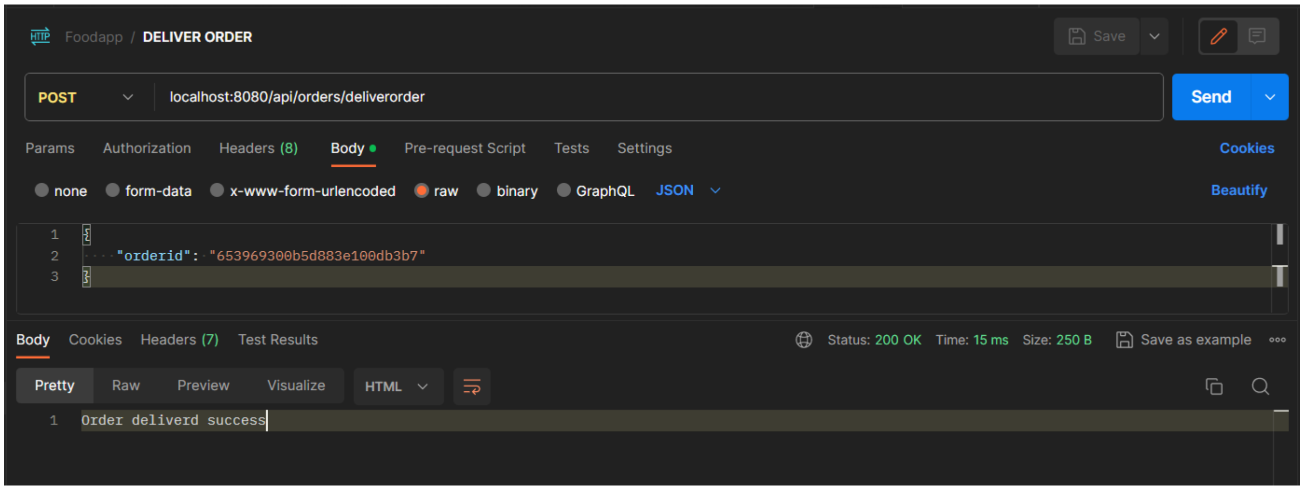
- ## DELIVER ORDER



```
router.post("/deliverorder", async (req, res) => {
  const orderid = req.body.orderid;
  try {
    const order = await Order.findOne({ _id: orderid });
    order.isDeliverd = true;
    await order.save();
    res.status(200).send("Order deliverd success");
  } catch (error) {
    res.status(400).json({
      message: "Something Went Wront",
      error: error.stack,
    });
  }
});
```

This route is responsible for marking an order as delivered in the database based on the orderid provided in the request body.

Postman Testing



POST localhost:8080/api/orders/deliverorder

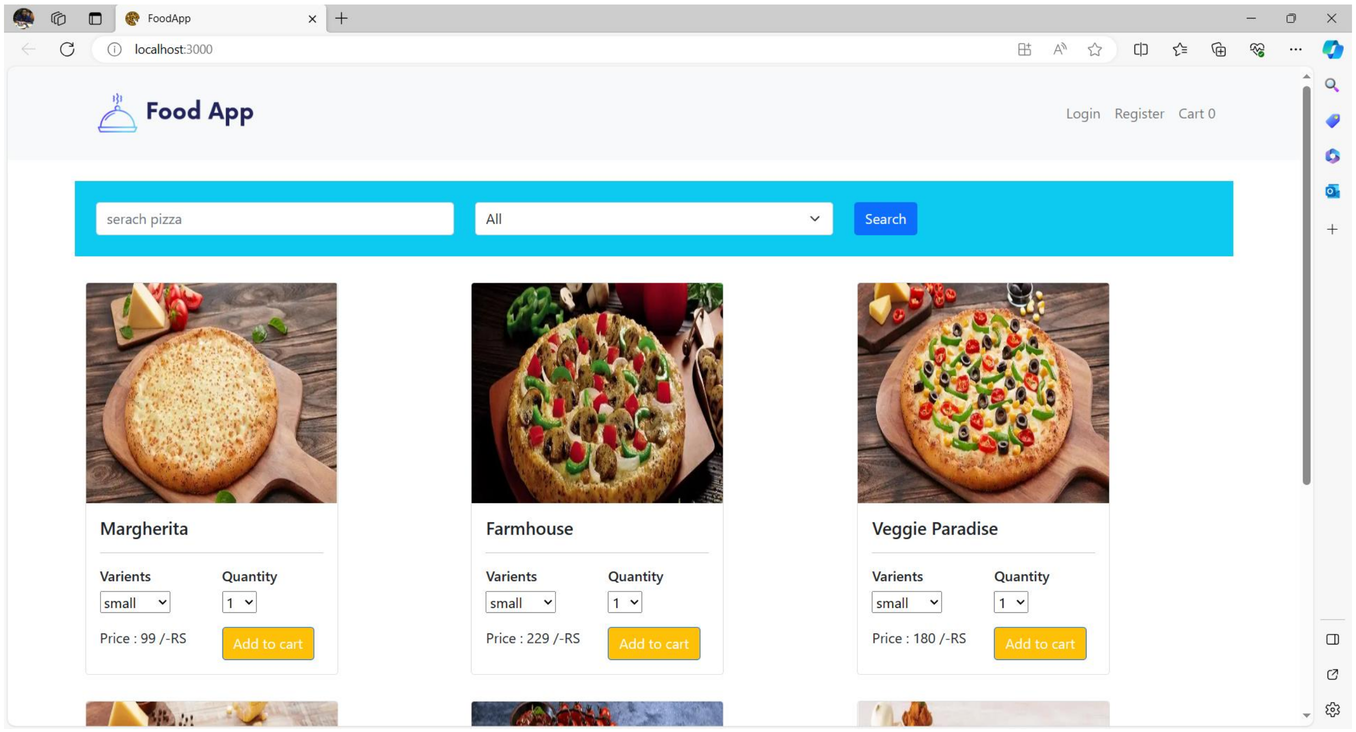
```
{
    "orderid": "653969300b5d883e100db3b7"
}
```

Status: 200 OK Time: 15 ms Size: 250 B

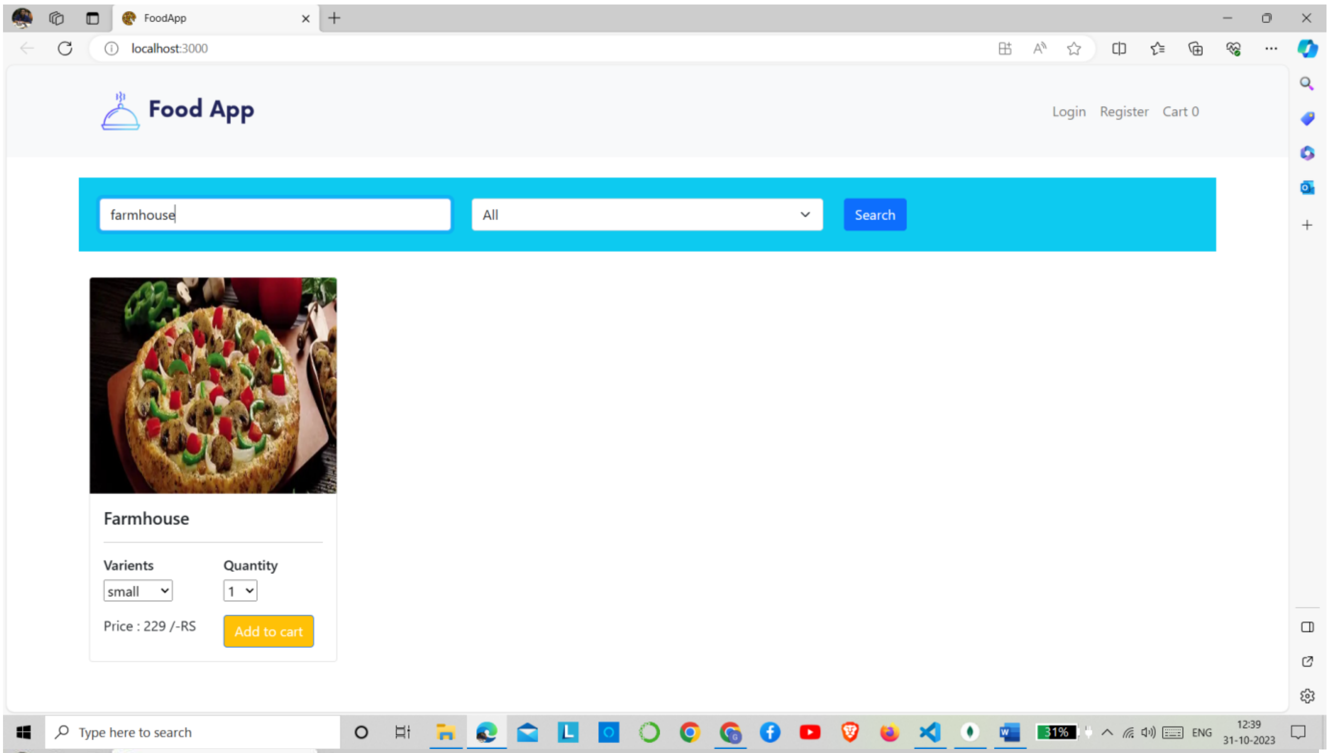
```
Order deliverd success
```

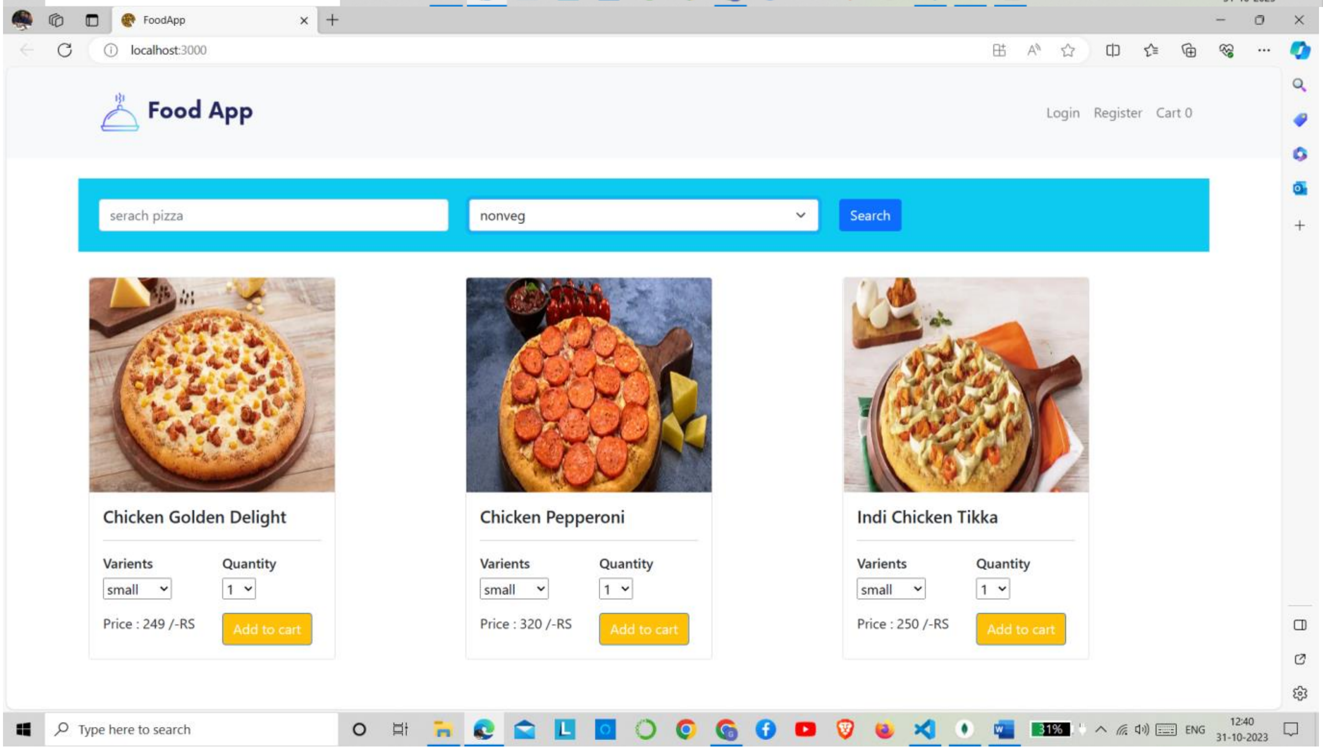
FRONTEND OF OUR WEBSITE

- **HOME PAGE**



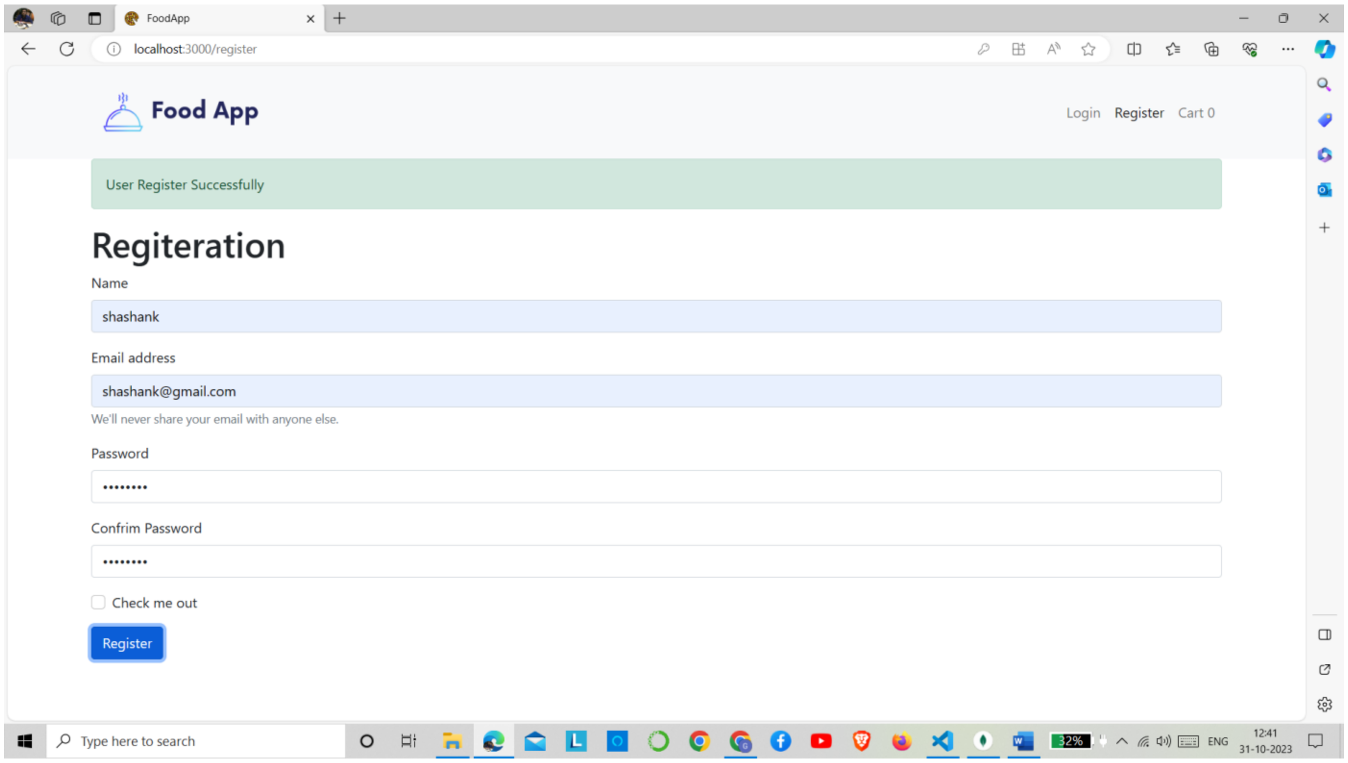
- **SEARCHING**





We can search the food items and category we want.

- **REGISTERING USER**



localhost:3000/register

Food App

Login Register Cart 0

User Register Successfully

Regiteration

Name: shashank

Email address: shashank@gmail.com

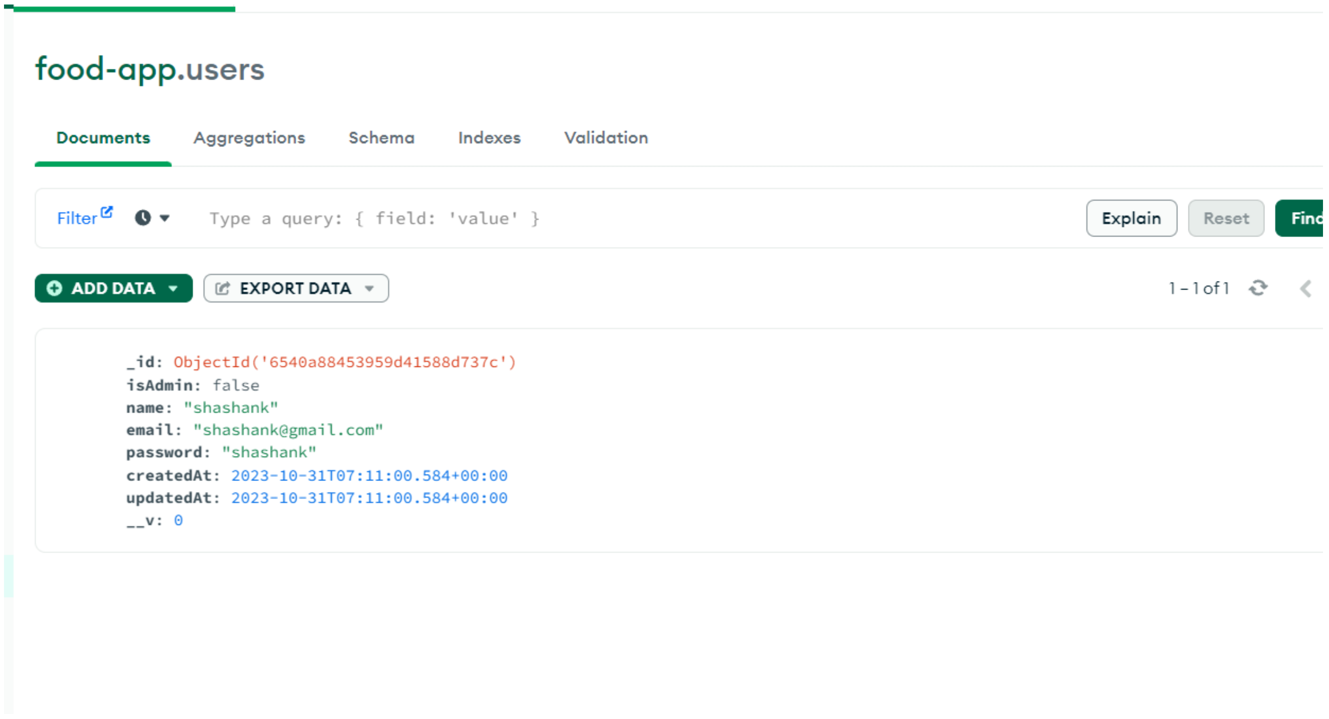
We'll never share your email with anyone else.

Password: ••••••••

Confrim Password: ••••••••

Check me out

Register



food-app.users

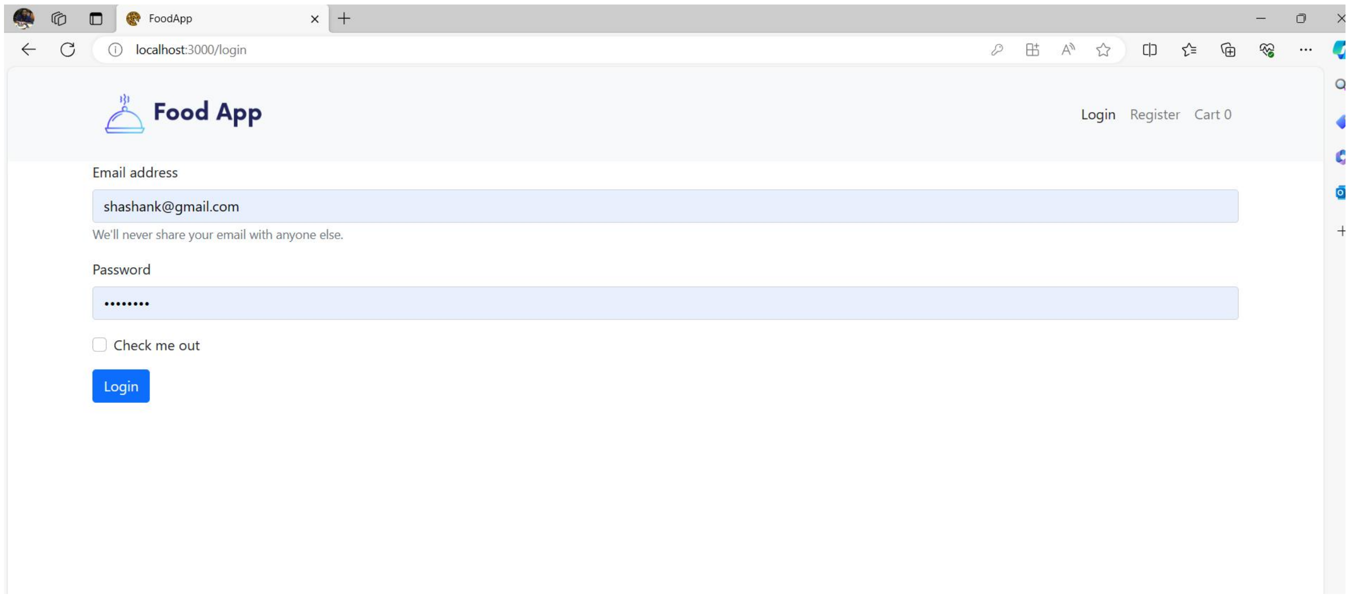
Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } Explain Reset Find

ADD DATA EXPORT DATA 1 - 1 of 1

```
_id: ObjectId('6540a88453959d41588d737c')
isAdmin: false
name: "shashank"
email: "shashank@gmail.com"
password: "shashank"
createdAt: 2023-10-31T07:11:00.584+00:00
updatedAt: 2023-10-31T07:11:00.584+00:00
__v: 0
```

- **LOGIN USER**


FoodApp

localhost:3000/login

Food App

Login Register Cart 0

Email address

shashank@gmail.com

We'll never share your email with anyone else.

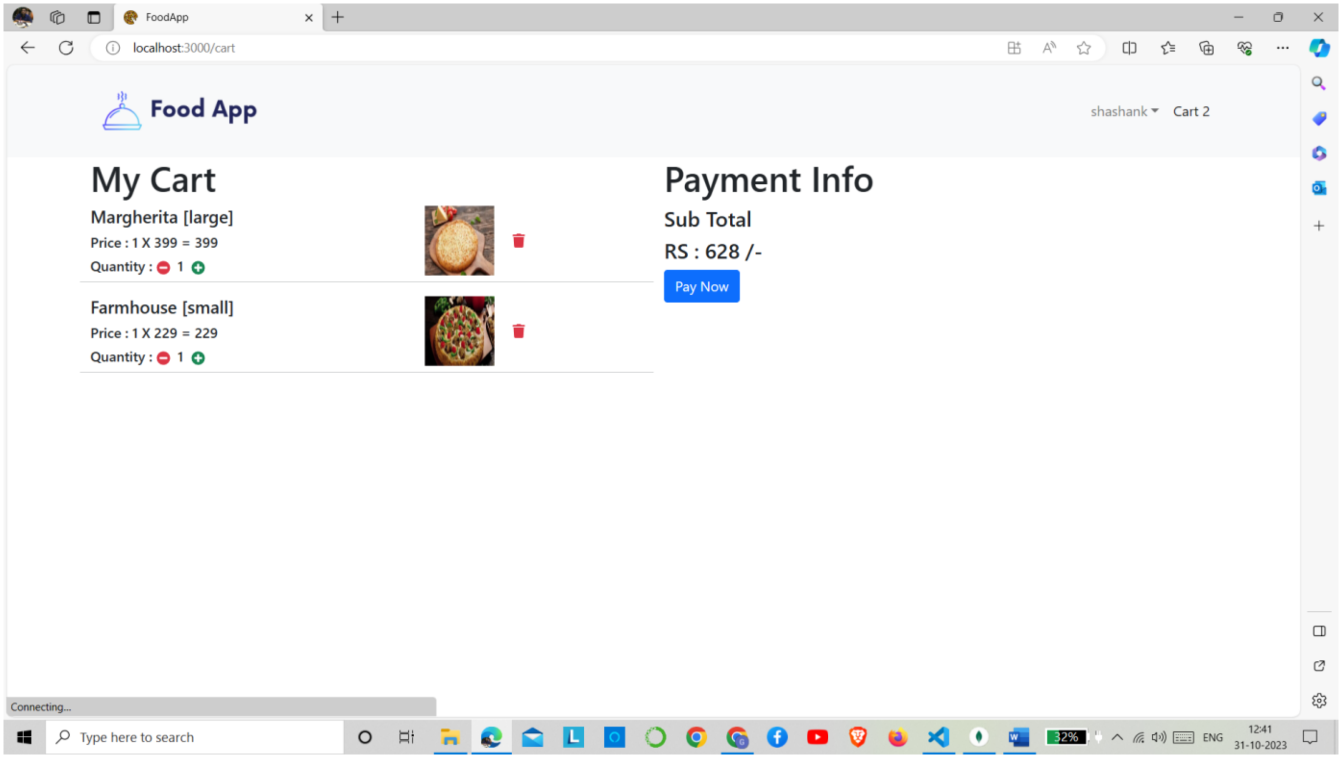
Password

••••••••

Check me out

Login

- **PLACING ORDER**


FoodApp

localhost:3000/cart

Food App

shashank Cart 2

My Cart

Margherita [large]

Price : 1 X 399 = 399

Quantity : 1

Farmhouse [small]

Price : 1 X 229 = 229

Quantity : 1

Payment Info

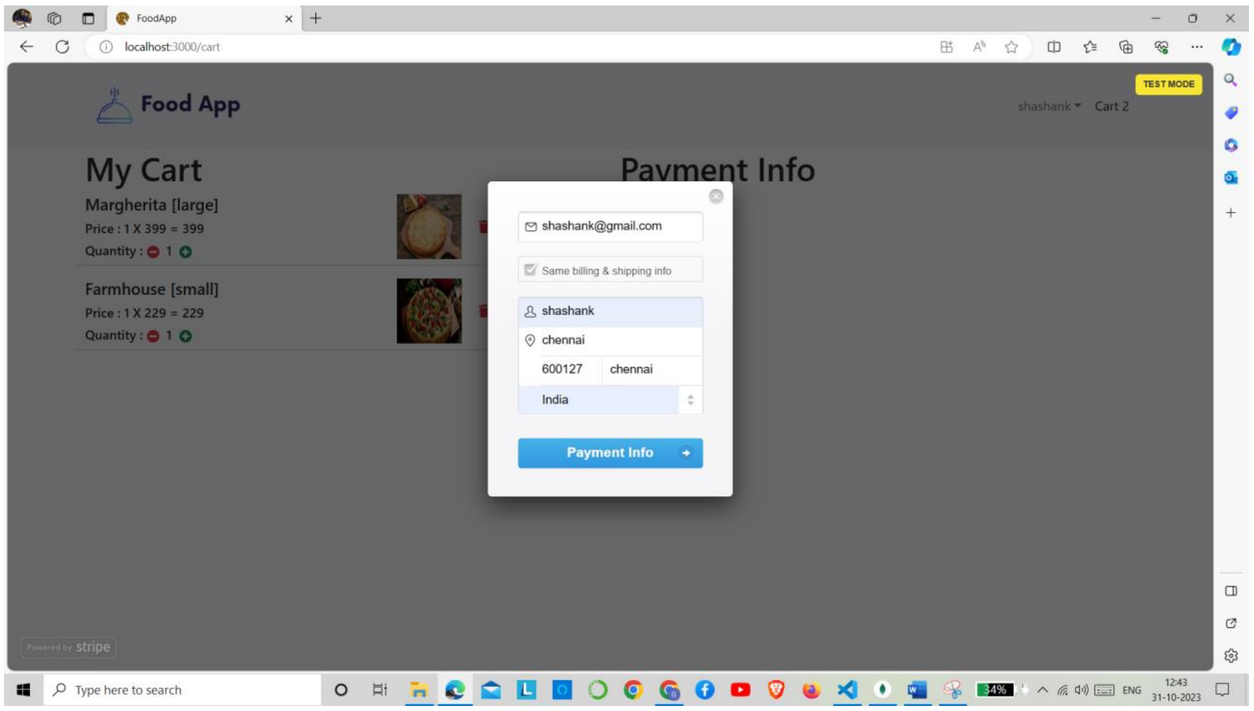
Sub Total

RS : 628 /-

Pay Now

Connecting...

The item is being added into the cart and we are having the pay now option in this Page.


FoodApp

localhost:3000/cart

Food App

shashank Cart 2

TEST MODE

My Cart

Margherita [large]
Price : 1 X 399 = 399
Quantity : 1

Farmhouse [small]
Price : 1 X 229 = 229
Quantity : 1

Payment Info

shashank@gmail.com

Same billing & shipping info

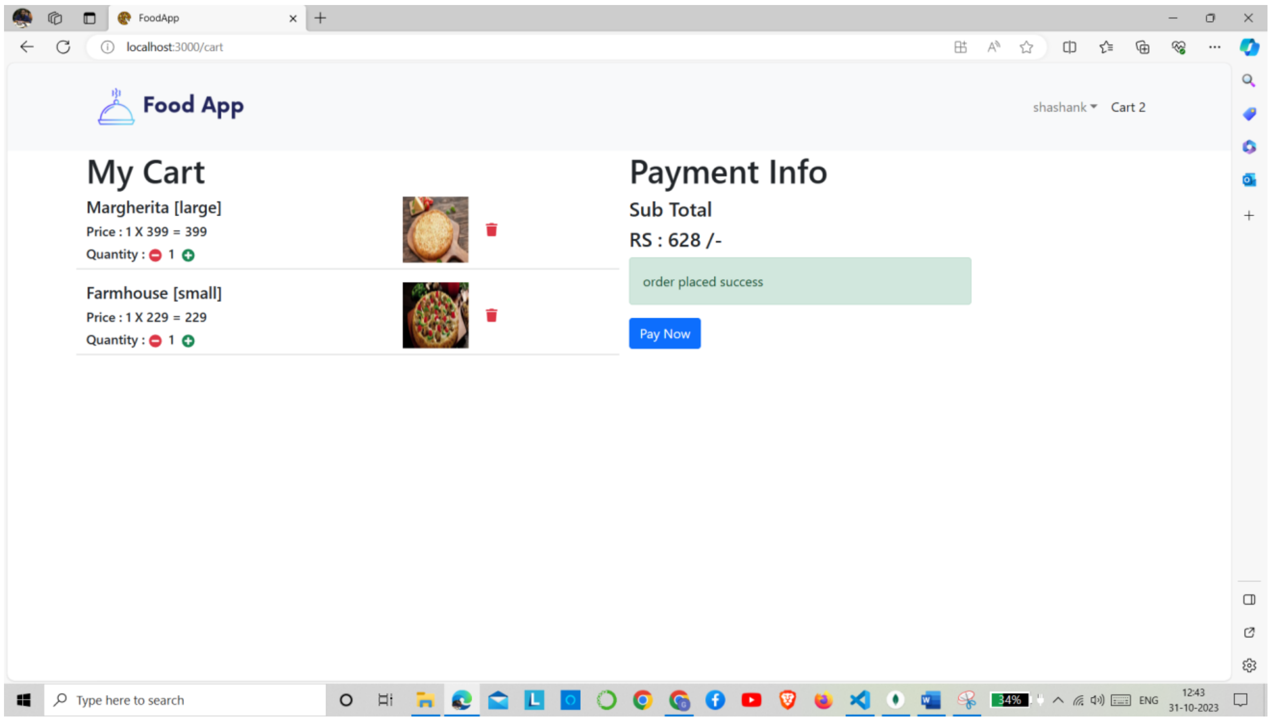
shashank
chennai
600127 chennai
India

Payment Info

Powered by stripe

After clicking on the pay now option the system ask for the email id and the billing address.

- **ORDER PLACED**


FoodApp

localhost:3000/cart

Food App

shashank Cart 2

My Cart

Margherita [large]
Price : 1 X 399 = 399
Quantity : 1

Farmhouse [small]
Price : 1 X 229 = 229
Quantity : 1

Payment Info

Sub Total

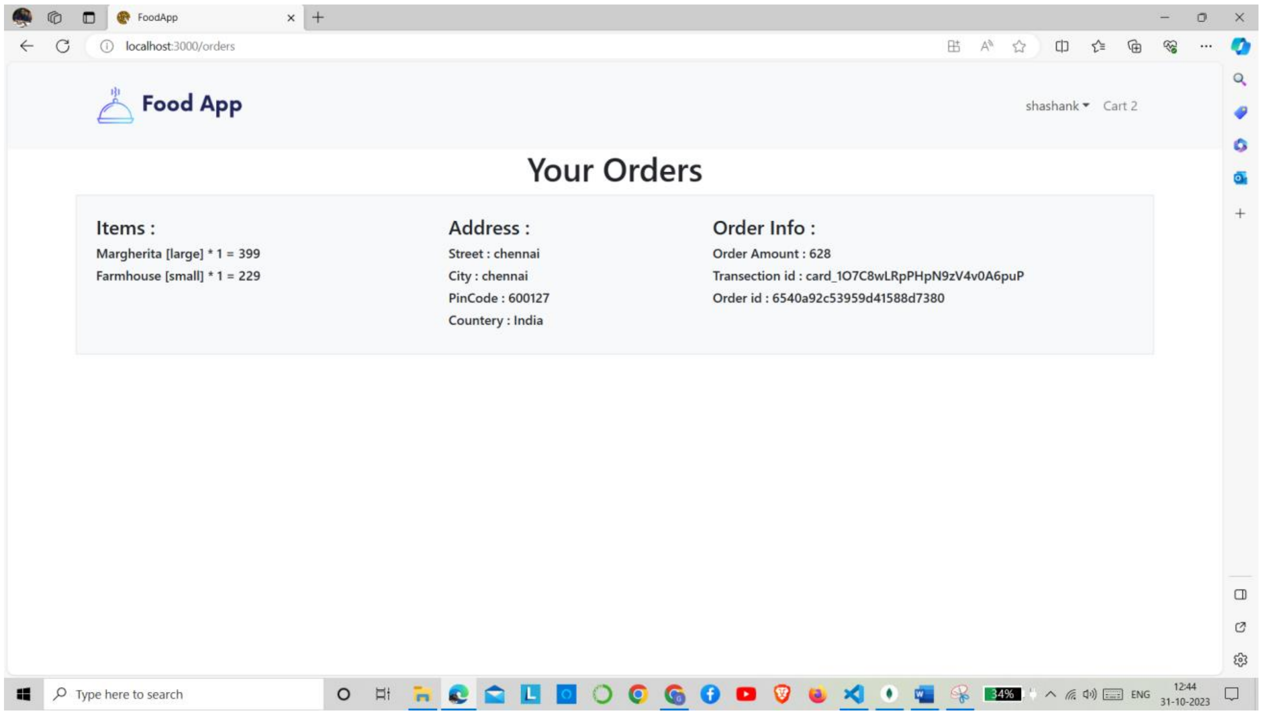
RS : 628 /-

order placed success

Pay Now

After we are done with payment our website will show the order placed success.

- **ORDER HISTORY**



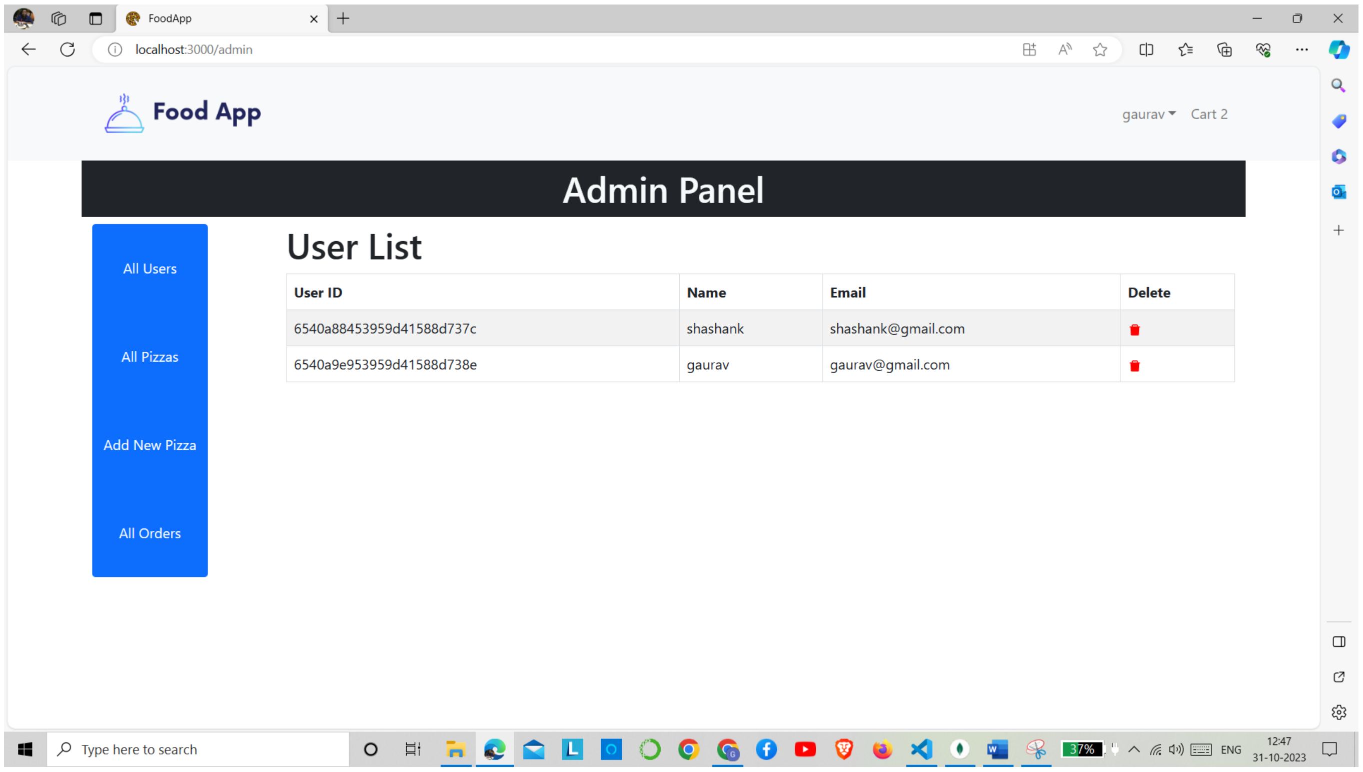
Your Orders

Items :
Margherita [large] * 1 = 399
Farmhouse [small] * 1 = 229

Address :
Street : chennai
City : chennai
PinCode : 600127
Countery : India

Order Info :
Order Amount : 628
Transection id : card_1O7C8wLRpPHpN9zV4v0A6puP
Order id : 6540a92c53959d41588d7380

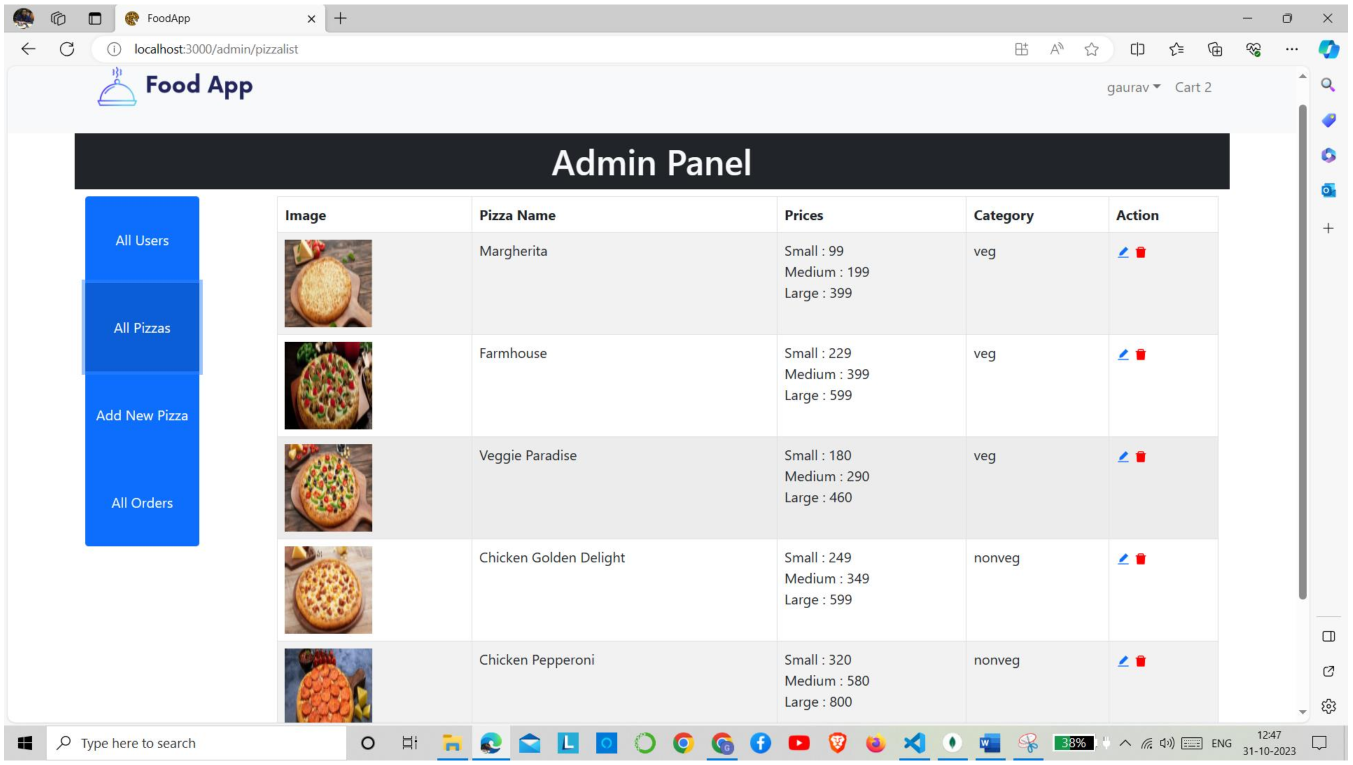
- **ADMIN ACCESS PAGE**



Admin Panel

User List

User ID	Name	Email	Delete
6540a88453959d41588d737c	shashank	shashank@gmail.com	🗑
6540a9e953959d41588d738e	gaurav	gaurav@gmail.com	🗑



Food App

gaurav Cart 2

Admin Panel

All Users

All Pizzas

Add New Pizza

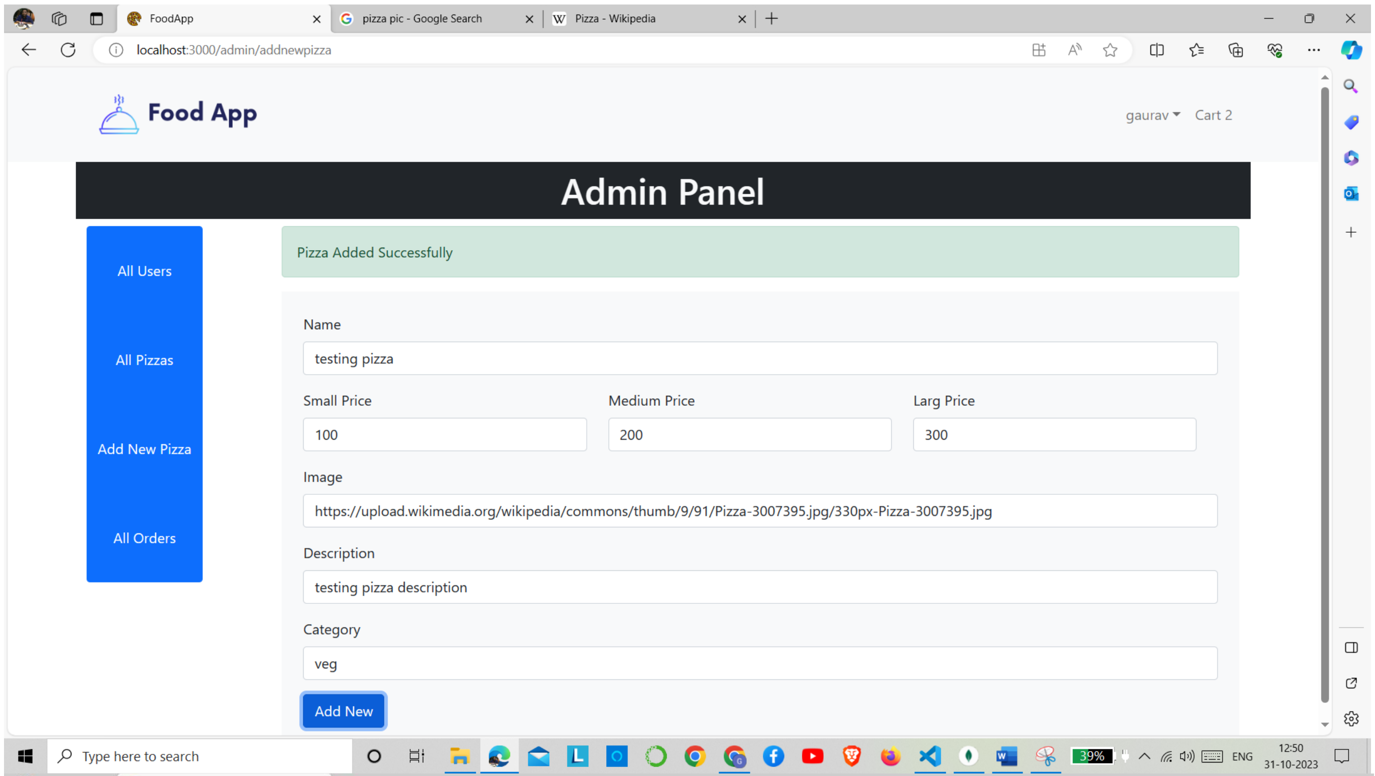
All Orders

Image	Pizza Name	Prices	Category	Action
	Margherita	Small : 99 Medium : 199 Large : 399	veg	
	Farmhouse	Small : 229 Medium : 399 Large : 599	veg	
	Veggie Paradise	Small : 180 Medium : 290 Large : 460	veg	
	Chicken Golden Delight	Small : 249 Medium : 349 Large : 599	nonveg	
	Chicken Pepperoni	Small : 320 Medium : 580 Large : 800	nonveg	

The admin access of the food delivery app website has the following functions:

- Manage users: Add, delete, and view all users.
- Manage pizzas: Add, delete, and view all pizzas.
- Manage orders: View all orders, view the details of each order, and view all orders that contain a particular pizza

- **ADDING PIZZA USING ADMIN PAGE**



Admin Panel

Pizza Added Successfully

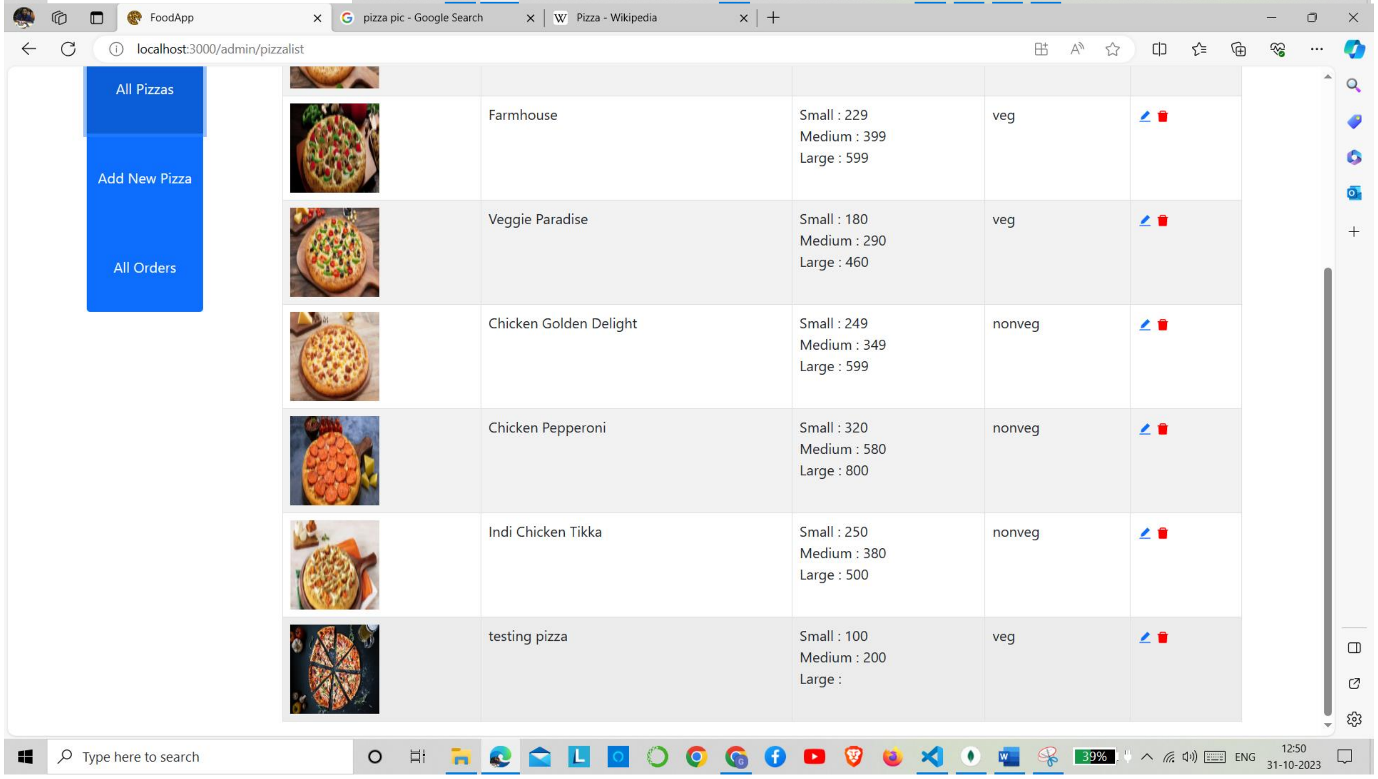
Name: testing pizza

Small Price: 100
Medium Price: 200
Larg Price: 300

Image: https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Pizza-3007395.jpg/330px-Pizza-3007395.jpg

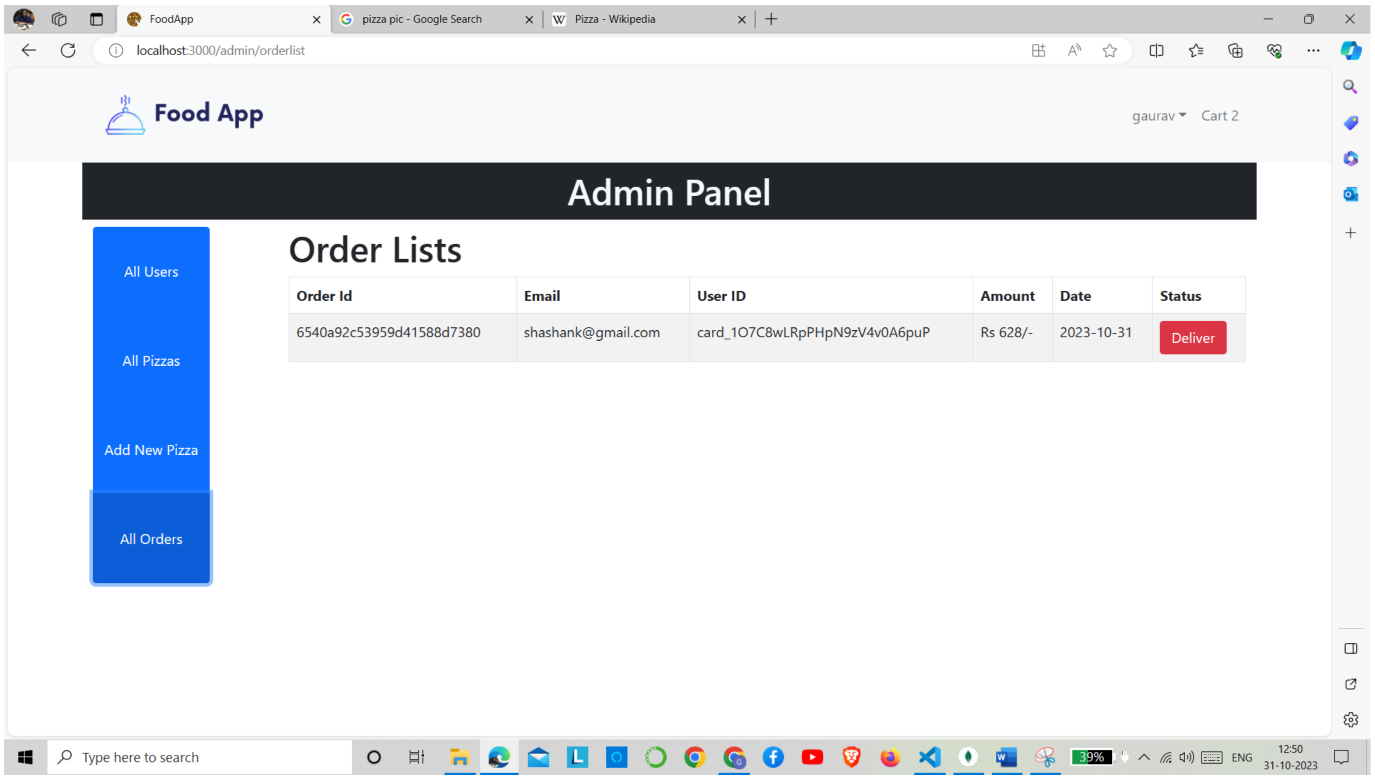
Description: testing pizza description

Category: veg



	Farmhouse	Small : 229 Medium : 399 Large : 599	veg	
	Veggie Paradise	Small : 180 Medium : 290 Large : 460	veg	
	Chicken Golden Delight	Small : 249 Medium : 349 Large : 599	nonveg	
	Chicken Pepperoni	Small : 320 Medium : 580 Large : 800	nonveg	
	Indi Chicken Tikka	Small : 250 Medium : 380 Large : 500	nonveg	
	testing pizza	Small : 100 Medium : 200 Large :	veg	

Our website allows us to modify , delete and add the pizza accordingly .



Admin Panel

Order Lists

Order Id	Email	User ID	Amount	Date	Status
6540a92c53959d41588d7380	shashank@gmail.com	card_1O7C8wLRpPHpN9zV4v0A6puP	Rs 628/-	2023-10-31	Deliver

The order list shows the following information about each order:

The admin can also filter the order list by order status, date, and user ID.

information shown on the website:

- Order ID: A unique identifier for each order.
- Email: The email address of the user who placed the order.
- User ID: The unique identifier for the user who placed the order.
- Amount: The total amount of the order.
- Status: The current status of the order, such as "Pending", "Accepted", "Preparing", "Out for delivery", or "Delivered".
- Date: The date and time the order was placed.

7. Conclusion

1. **Meeting Contemporary Demand:** In a fast-paced society, the Food Delivery Application project successfully addresses the growing demand for efficient and convenient food delivery services by leveraging the power of modern web technologies.

2. **MERN Stack Proficiency:** The project provides a valuable opportunity for developers to gain insights into the MERN (MongoDB, Express.js, React, and Node.js) stack, showcasing best practices in application design and deployment of contemporary web technologies.

3. **User-Centric Approach:** The application prioritizes the user experience, offering features like user registration, dynamic menu browsing, order placement, real-time tracking, and secure payment methods to ensure a gratifying dining experience.

4. **Restaurant Management:** It not only benefits patrons but also empowers restaurant owners with efficient restaurant and menu administration, enhancing their online presence and service quality.

5. **Real-Time Updates**: By combining MongoDB and Node.js, the application ensures real-time updates and scalability, resulting in efficient data management and retrieval.

6. **Functionalities:** The application encompasses a wide range of functionalities, including pizza management, user management, order handling, and admin access, making it a comprehensive solution for online food delivery.

7. **Scalability:** With a strong backend infrastructure and adaptable design, the Food Delivery Application is poised for scalability to accommodate a growing user base and evolving market trends.

8. **Utilization of Postman:** Efficient Testing pf API routes using postman, improving the overall quality and reliability of our application.

8. Future Scope

1. Geographic Expansion: Extend the application's reach to serve a wider geographical area, tapping into new markets and accommodating a larger customer base.

2. Multi-Platform Support: Develop dedicated mobile applications for iOS and Android platforms to enhance accessibility for users on mobile devices.

3. User Reviews and Ratings: Implement a user review and rating system to provide valuable feedback and assist other customers in making informed decisions.

4. Integration with Third-Party Services: Integrate with third-party services, such as weather updates or traffic data, to provide real-time information that can affect food delivery times.

5. Advanced Analytics: Implement robust analytics tools to gain insights into user behavior, preferences, and market trends, allowing for data-driven decision-making.

6. Payment Gateway Expansion: Integrate additional payment gateways and digital wallets to offer users a broader range of payment options.

7. Enhanced Security Features: Invest in enhanced security measures to protect user data and transactions, instilling trust and confidence in the application.

8. Sustainability Initiatives: Incorporate sustainability features, such as eco-friendly packaging and delivery methods, to align with evolving environmental concerns.

9. Blockchain for Transparency: Explore blockchain technology to provide transparency in the food supply chain, ensuring the authenticity and quality of ingredients.

10. Offline Mode: Develop an offline mode that allows users to browse menus and place orders even when they have limited or no internet connectivity.

9. Github Repository

https://github.com/shashankpandey2411/FoodApp-using-MERN

REFERENCES

[1] Joshi, Umesh, et al. "Online Food Ordering System." International Journal of Advanced Research in Computer Science 13 (2022).

[2] He, Zhou, et al. "Evolutionary food quality and location strategies for restaurants in competitive online-to-offline food ordering and delivery markets: An agent-based approach." International Journal of Production Economics 215 (2019): 61-72.

[3] Patel, Mayurkumar. "Online Food Order System for Restaurants." (2015).

[4] Tarun Garg, Ms. Meenu Garg and Dr. Bhoomi Gupta, “Food Ordering Web Application for the Fitness freaks”,

[5] Joseph George , Sreenath P , Siddharth K G , Jeswill Paulose, 2020, Food Ordering Application for Canteen using React Native, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 09, Issue 06 (June 2020).

[6] Pathak, Harsh & Gupta, Naman & Premaker, Dhiren & Garg, Preeti. (2022). Design of Web App for Online Food Services. International Journal for Research in Applied Science and Engineering Technology. 10. 4316-4322.
10.22214/ijraset.2022.43136..

[7] Byali, Ramesh. (2022). Using MongoDB to Understand the Underlying Methods Techniques Encryption in NoSQL database. 862-866.

[8] Chauhan, Anjali. "A review on various aspects of MongoDb databases." Int. J. Eng. Res. Sci. Technol 8.5 (2019): 90-92.

[9] https://doi.org/10.22214/ijraset.2022.39982