# INTERDISCIPLINARY PROJECT REPORT

**at**

**Sathyabama Institute of Science and Technology**

**(Deemed to be University)**

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering

By

**DEVIREDDY SRAVANI LAKSHMI DURGA**

**Reg. No.: 40110312**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SCHOOL OF COMPUTING**

**SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY JEPPIAAR NAGAR, RAJIV GANDHI SALAI,**

**CHENNAI – 600119, TAMILNADU**

**APRIL 2023**

**SATHYABAMA**

**INSTITUTE OF SCIENCE AND TECHNOLOGY**

### (DEEMED TO BE UNIVERSITY)

**Accredited with Grade “A” by NAAC**

**(Established under Section 3 of UGC Act, 1956)**

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI– 600119**

[**www.sathyabamauniversity.ac.in**](http://www.sathyabamauniversity.ac.in/)

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the bonafide work of **DEVIREDDY SRAVANI LAKSHMI DURGA (40110312)** who carried out the project entitled “**QnA Backend Service**” under my supervision from FEB 2023 to APRIL 2023.

**Internal Guide**

**Dr.KALAIRASI**

**COMPUTER SCIENCE AND ENGINEERING**

**Head of the Department**

**Dr. L.** **Lakshmanan, M.E., Ph.D.**

**COMPUTER SCIENCE AND ENGINEERING**



## Submitted for Viva voce Examination held on

**Internal Examiner External Examiner**

**DECLARATION**

I **DEVIREDDY SRAVANI LAKSHMI DURGA** hereby declare that the Project Report entitled **“QnA Backend Service”** done by me under the guidance of **Dr.KALAIRASI**  is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

## DATE:

**PLACE: CHENNAI SIGNATURE OF THE CANDIDATE**

**ACKNOWLEDGEMENT**

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D.**, **Dean**, School of Computing, **Dr. L. Lakshmanan, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.KALAIRASI** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

# ABSTRACT

The QnA Backend Service for Ecommerce is a software system designed to provide a question-and-answer (QnA) service for ecommerce websites. The purpose of the system is to enable customers to ask questions about products and receive answers from either the ecommerce site's support team or other customers who have already purchased the product. The QnA Backend Service consists of a backend server and an API that allows ecommerce websites to integrate the QnA service into their product pages. The backend server is responsible for storing and managing the questions and answers, while the API enables the ecommerce website to communicate with the backend server. The QnA service includes features such as real-time notifications, spam detection, and moderation tools to ensure that questions and answers are relevant, accurate, and appropriate. Customers can also rate the helpfulness of each answer, which helps to improve the overall quality of the QnA service. By providing a QnA service, ecommerce websites can improve customer engagement, reduce customer support costs, and increase customer satisfaction. The QnA Backend Service for Ecommerce is a valuable tool for any ecommerce website looking to enhance their customer experience and drive sales.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Title** | **Page no.** |
|  |  |  |
|  | **Abstract** | 5 |
|  |  |  |
| **1** | **INTRODUCTION** |  |
| 1.1 | E-Commerce | 7 |
| 1.2 | Overview | 8 |
| 1.3 | Background Information | 9 |
| 1.4 | Importance | 10 |
| 1.5 | Target Audience | 11 |
|  |  |  |
| **2** | **AIM AND SCOPE OF PRESENT INVESTIGATION** |  |
| 2.1 | Aim | 12 |
| 2.2 | Scope | 12 |
|  |  |  |
| **3** | **USED TECHNOLOGIES** |  |
| 3.1 | React | 13 |
| 3.2 | Node.js | 14 |
| 3.3 | Express.js | 15 |
| 3.4 | MongoDB | 16 |
| 3.5 | Postman | 17 |
|  |  |  |
| **4** | **RESULT AND PERFORMANCE ANALYSIS** |  |
| 4.1 | Frontend | 18 |
| 4.2 | Backend | 25 |
|  |  |  |
| **5** | **SUMMARY AND CONCLUSIONS** | **30** |
|  |  |  |
|  | **REFERNCES** | **31** |
|  | **APPENDIX** | 32 |
|  | Source Code | 32 |

# CHAPTER 1 – INTRODUCTION

## 1.1 E-COMMERCE

Ecommerce websites typically rely on customer support teams to answer questions about products and resolve issues that customers may have. This can be time-consuming and costly for ecommerce businesses, particularly those with a large customer base.

To address this challenge, many ecommerce websites have implemented question-and-answer (QnA) services to enable customers to ask and answer questions about products. This not only helps customers find the information they need but also reduces the workload on customer support teams.

However, implementing a QnA service can be challenging, particularly for smaller ecommerce websites with limited resources. That's where a QnA backend service comes in. By providing a ready-made solution that can be integrated into an ecommerce website, a QnA backend service can help ecommerce businesses save time and money while improving the customer experience.

The QnA backend service for ecommerce that you're creating aims to provide a robust and reliable solution for ecommerce websites looking to implement a QnA service. By offering features such as real-time notifications, spam detection, and moderation tools, your service can help ecommerce businesses provide a high-quality QnA service to their customers, enhancing the overall customer experience and driving sales.

## 1.2 OVERVIEW OF THE PROJECT

The QnA Backend Service for Ecommerce is a software system that provides a question-and-answer (QnA) service for ecommerce websites. The purpose of the system is to enable customers to ask questions about products and receive answers from either the ecommerce site's support team or other customers who have already purchased the product.

The QnA Backend Service consists of a backend server and an API that allows ecommerce websites to integrate the QnA service into their product pages. The backend server is responsible for storing and managing the questions and answers, while the API enables the ecommerce website to communicate with the backend server.

The QnA service includes features such as real-time notifications, spam detection, and moderation tools to ensure that questions and answers are relevant, accurate, and appropriate. Customers can also rate the helpfulness of each answer, which helps to improve the overall quality of the QnA service.

The QnA Backend Service for Ecommerce is a valuable tool for any ecommerce website looking to enhance their customer experience and drive sales. By providing a QnA service, ecommerce websites can improve customer engagement, reduce customer support costs, and increase customer satisfaction.

The key objectives of this project are to develop a scalable and reliable backend server for the QnA service, to implement an API that can be easily integrated into ecommerce websites, and to provide a user-friendly interface for customers to ask and answer questions. The project also aims to incorporate features such as real-time notifications, spam detection, and moderation tools to ensure the quality and accuracy of the QnA service.

## 1.3 BACKGROUND INFORMATION

As ecommerce has become increasingly popular, ecommerce websites have recognized the need to provide customers with a robust and reliable QnA service. Customers want to be able to ask questions about products before making a purchase, and ecommerce businesses want to provide them with the information they need to make informed decisions.

However, implementing a QnA service can be challenging for ecommerce businesses, particularly those with limited resources. Building and maintaining a QnA system requires a significant amount of time, effort, and technical expertise. Furthermore, ensuring that the QnA service is accurate, relevant, and up-to-date can be a daunting task.

To address these challenges, a number of third-party QnA services have emerged, offering ecommerce websites an easy and affordable way to implement a QnA service. These services typically provide a backend server and an API that ecommerce websites can use to integrate the QnA service into their product pages.

The QnA Backend Service for Ecommerce that you're creating is one such solution. By providing a scalable and reliable backend server and an API that can be easily integrated into ecommerce websites, your service aims to help ecommerce businesses provide a high-quality QnA service to their customers. This can lead to increased customer satisfaction, improved customer engagement, and ultimately, increased sales for ecommerce businesses.

## 1.4 IMPORTANCE OF THE PROJECT

Here are some reasons why this project is important:

1. Enhanced Customer Experience: By providing a QnA service, ecommerce websites can enhance the customer experience by allowing customers to get their questions answered quickly and easily. This can help customers make informed purchasing decisions and ultimately lead to increased sales for the ecommerce business.
2. Reduced Customer Support Costs: A QnA service can reduce the workload on customer support teams by allowing customers to find the information they need without having to contact customer support. This can save ecommerce businesses time and money while still providing a high level of customer support.
3. Improved Customer Engagement: A QnA service can help to improve customer engagement by allowing customers to interact with each other and share their experiences with products. This can foster a sense of community around the ecommerce website and lead to increased customer loyalty.
4. Competitive Advantage: Offering a QnA service can give ecommerce businesses a competitive advantage by differentiating them from other ecommerce websites that do not offer such a service. This can help to attract and retain customers, particularly those who value the ability to ask and answer questions about products.
5. Valuable Insights: The QnA service can also provide valuable insights into customer behavior and preferences. By analyzing the questions and answers, ecommerce businesses can gain a better understanding of their customers and use this information to improve their products and services.

## 1.5 TARGET AUDIENCE

The target audience of this project includes:

1. Ecommerce businesses: The primary target audience for this project is ecommerce businesses of all sizes that sell products online. These businesses may be looking to enhance their customer experience, reduce customer support costs, and improve customer engagement by providing a QnA service on their product pages.
2. Developers: Another target audience for this project is developers who work for ecommerce businesses or third-party development firms. These developers may be responsible for integrating the QnA service into ecommerce websites using the API provided by the backend server.
3. Customers: While not a direct target audience for the project, customers who shop on ecommerce websites that use the QnA service will benefit from it. They will be able to ask and answer questions about products, which can help them make informed purchasing decisions and ultimately lead to increased satisfaction with the ecommerce website.
4. Investors: Finally, investors who are interested in the ecommerce industry may also be a target audience for the project. They may be interested in investing in ecommerce businesses that use the QnA service or in the development of the QnA service itself as a standalone product.

In summary, the target audience for this project includes ecommerce businesses, developers, customers, and investors who are interested in the ecommerce industry.

# CHAPTER 2 – AIM AND SCOPE OF PRESENT INVESTIGATION

## 2.1 AIM

The aim of the project is to provide a scalable, reliable, and easy-to-use QnA service for ecommerce businesses. The service will allow customers to ask and answer questions about products on ecommerce websites, enhancing the customer experience and ultimately leading to increased sales for the ecommerce businesses.

## 2.2 SCOPE

The scope of the project includes the development of a backend server that can handle large volumes of requests and provide accurate and relevant responses to customer questions. The backend server will also include a database to store the questions and answers and an API that ecommerce websites can use to integrate the QnA service into their product pages.

Additionally, the project will involve the development of a user interface for managing the QnA service, including the ability to moderate questions and answers and to provide analytics on customer behavior and preferences.

The scope of the project also includes testing and quality assurance to ensure that the QnA service is reliable and accurate, as well as documentation to help ecommerce businesses and developers integrate the service into their websites.

Overall, the scope of this project is to provide a comprehensive and effective solution for ecommerce businesses that want to implement a QnA service on their product pages.

# CHAPTER 3 – USED TECHNOLOGIES

## 3.1 REACT

React is a popular open-source library for building user interfaces. Developed by Facebook, React allows you to create reusable UI components and build complex user interfaces using a declarative approach.

React works by using a virtual DOM (Document Object Model), a lightweight in-memory representation of the actual DOM. By using a virtual DOM, React can minimize the number of actual DOM manipulations required to update the UI, which improves performance.

React uses a component-based architecture, where each UI element is defined as a reusable component. A component is a JavaScript class or function that encapsulates the HTML, CSS, and JavaScript needed to render a UI element.

React provides a powerful set of tools for managing component state and handling user events. Components can maintain their own internal state, and React provides a set of lifecycle methods that allow you to perform actions when a component is created, updated, or destroyed.

React can be used to create a responsive and dynamic user interface for your QnA service. You can create components for displaying questions and answers, and use React's powerful state management and event handling features to provide a smooth user experience. Additionally, you can use React Router to handle client-side routing, allowing users to navigate between product pages and QnA pages without the need for a page refresh.

## 3.2 NODE.JS

Node.js is an open-source, cross-platform, JavaScript runtime built on the V8 JavaScript engine, which is also used by Google Chrome. Node.js allows you to run JavaScript code outside of the browser, on the server-side, and can be used to build fast, scalable, and high-performance network applications.

Node.js provides a non-blocking, event-driven I/O model that allows it to handle large numbers of concurrent connections with low overhead. This makes Node.js particularly well-suited for building real-time applications that require fast and responsive performance.

One of the key benefits of Node.js is that it uses a single-threaded event loop to handle incoming requests. This means that Node.js can handle multiple requests simultaneously without creating a new thread for each request, as is typical in other server-side technologies. This results in low memory usage and improved scalability.

Node.js also has a built-in module system that allows you to reuse code across multiple applications. Node.js modules can be written in JavaScript or other languages that can be compiled to JavaScript, such as TypeScript. The Node.js package manager, npm, provides a large repository of third-party modules that you can use in your applications.

Node.js is often used with other technologies such as Express, a web application framework, and MongoDB, a NoSQL database, to create full-stack web applications. With Node.js, you can build RESTful APIs, web servers, real-time chat applications, and much more.

Overall, Node.js is a powerful and versatile technology that allows you to build fast, scalable, and high-performance applications using JavaScript on the server-side. Its non-blocking I/O model, single-threaded event loop, and built-in module system make it a popular choice for building modern web applications.

## 3.3 EXPRESS.JS

Express.js is a popular open-source framework for building web applications in Node.js. It provides a set of features and utilities for building robust and scalable web applications and APIs.

Express.js allows you to define routes for your application, which map HTTP requests to specific functions or handlers that are responsible for generating the response. Express.js supports a variety of HTTP methods, such as GET, POST, PUT, DELETE, and more, and allows you to handle requests and responses in a flexible and efficient way.

In addition, Express.js provides a middleware architecture that allows you to plug in third-party modules and custom functions that can modify the request or response objects, or perform additional processing before or after the main handler function. Middleware functions can be chained together, and are executed in the order in which they are defined in the middleware stack.

Express.js also provides a powerful template engine, called Pug, that allows you to generate dynamic HTML pages and views using a simple syntax. Pug templates can include variables, control structures, and functions, and can be easily customized and extended to fit your specific needs.

Overall, Express.js is a powerful and flexible framework for building web applications and APIs in Node.js. Its routing system, middleware architecture, and template engine provide a solid foundation for building robust and scalable applications, and its large and active community provides a wealth of resources and modules that can help you get started quickly and easily.

## 3.4 MONGODB

MongoDB is a popular open-source, NoSQL document-oriented database that is designed to handle large amounts of data in a flexible and scalable manner. Unlike traditional relational databases, MongoDB stores data in flexible and dynamic documents that can be easily updated and expanded as needed.

MongoDB documents are stored in collections, which are similar to tables in a traditional database. However, MongoDB documents can contain complex nested data structures and arrays, making it easy to store and query data in a natural way.

MongoDB uses a query language called the MongoDB Query Language (MQL), which allows you to perform complex queries on your data. MQL supports a variety of query operators, such as $and, $or, $in, and more, and allows you to filter, sort, and aggregate your data in a flexible and efficient way.

One of the key benefits of MongoDB is its ability to scale horizontally, allowing you to add more servers to your cluster as your data grows. MongoDB supports sharding, which allows you to distribute your data across multiple servers, and replication, which allows you to create redundant copies of your data for increased availability and reliability.

MongoDB also provides a rich set of features and tools for working with data, including a built-in aggregation framework, text search, geospatial queries, and more. In addition, MongoDB integrates well with other technologies such as Node.js, Express.js, and React, making it a popular choice for building full-stack web applications.

Overall, MongoDB is a flexible, scalable, and high-performance NoSQL database that allows you to store and query large amounts of data in a natural and efficient way. Its support for complex data structures, flexible schema, and horizontal scalability make it a popular choice for building modern web applications and APIs.

## 3.5 POSTMAN

Postman is a popular tool used for testing and interacting with APIs. In my project, I used Postman to test the endpoints of my backend API and to verify that my API was functioning correctly.

Using Postman, I was able to create requests for different HTTP methods, such as GET, POST, PUT, and DELETE, and to send those requests to my backend API. I was also able to specify various parameters for my requests, such as headers, query parameters, and request bodies, and to view the responses returned by my API.

One of the key benefits of Postman is its ability to save and organize requests and collections, which allowed me to easily reuse requests and share them with others. I was able to create collections of requests for different parts of my API, such as user management, product management, and order management, and to organize those collections into folders for easy access.

In addition to testing my API endpoints, I also used Postman to simulate different scenarios, such as sending invalid data or testing error handling, in order to ensure that my API was robust and secure. I was able to use Postman's scripting capabilities to automate some of these tests and to generate reports for my testing results.

Overall, Postman was a valuable tool for my project, allowing me to test and verify the functionality of my backend API, to organize and save my requests, and to simulate different scenarios in order to ensure the quality and security of my API.

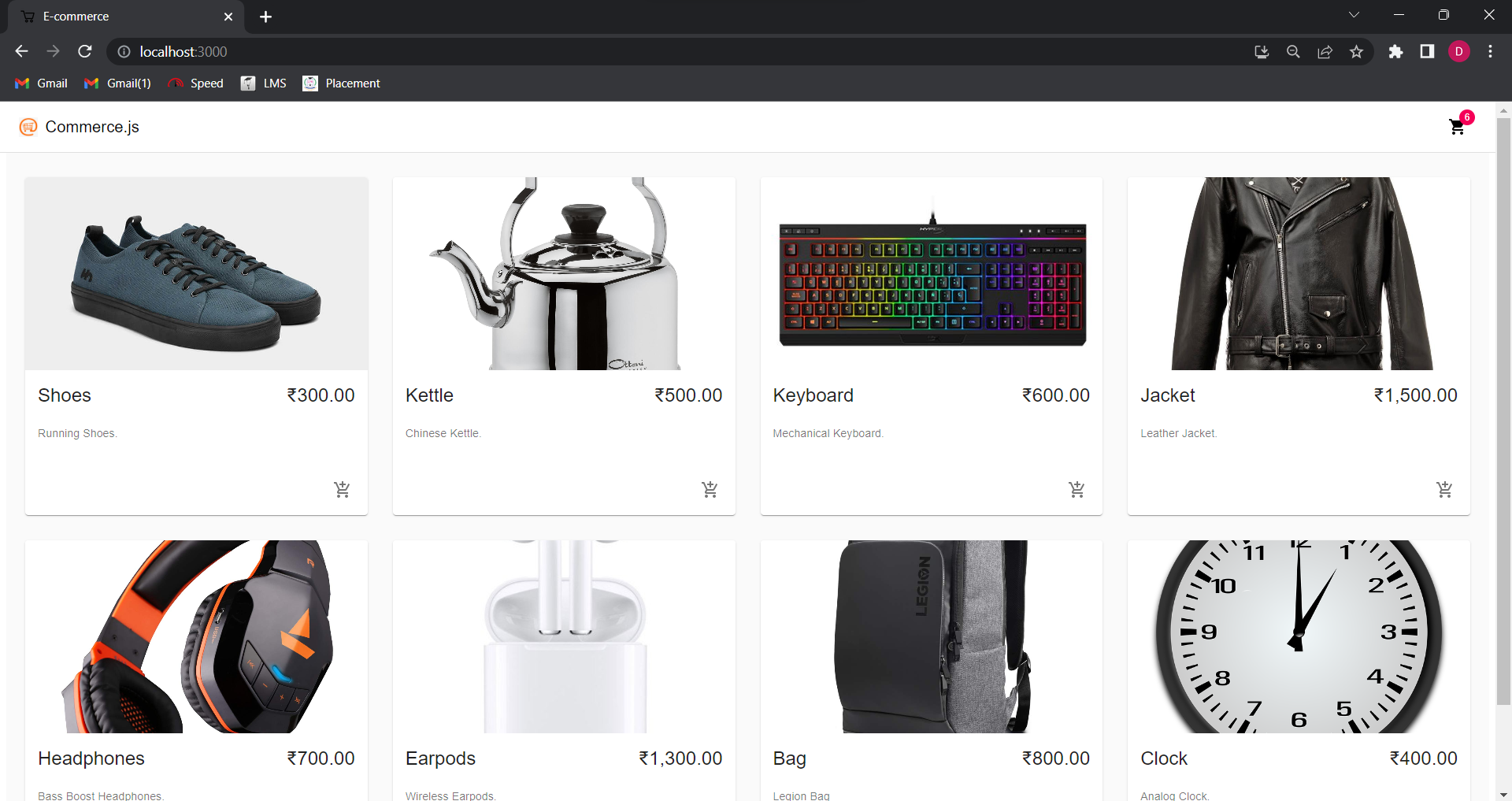
# CHAPTER 4 – RESULT AND PERFORMANCE ANALYSIS

## 4.1 FRONTEND

***4.1.1 Home Page***

The home page of this ecommerce website contains eight static products. The products are stored and managed by Commerce.js APIs.

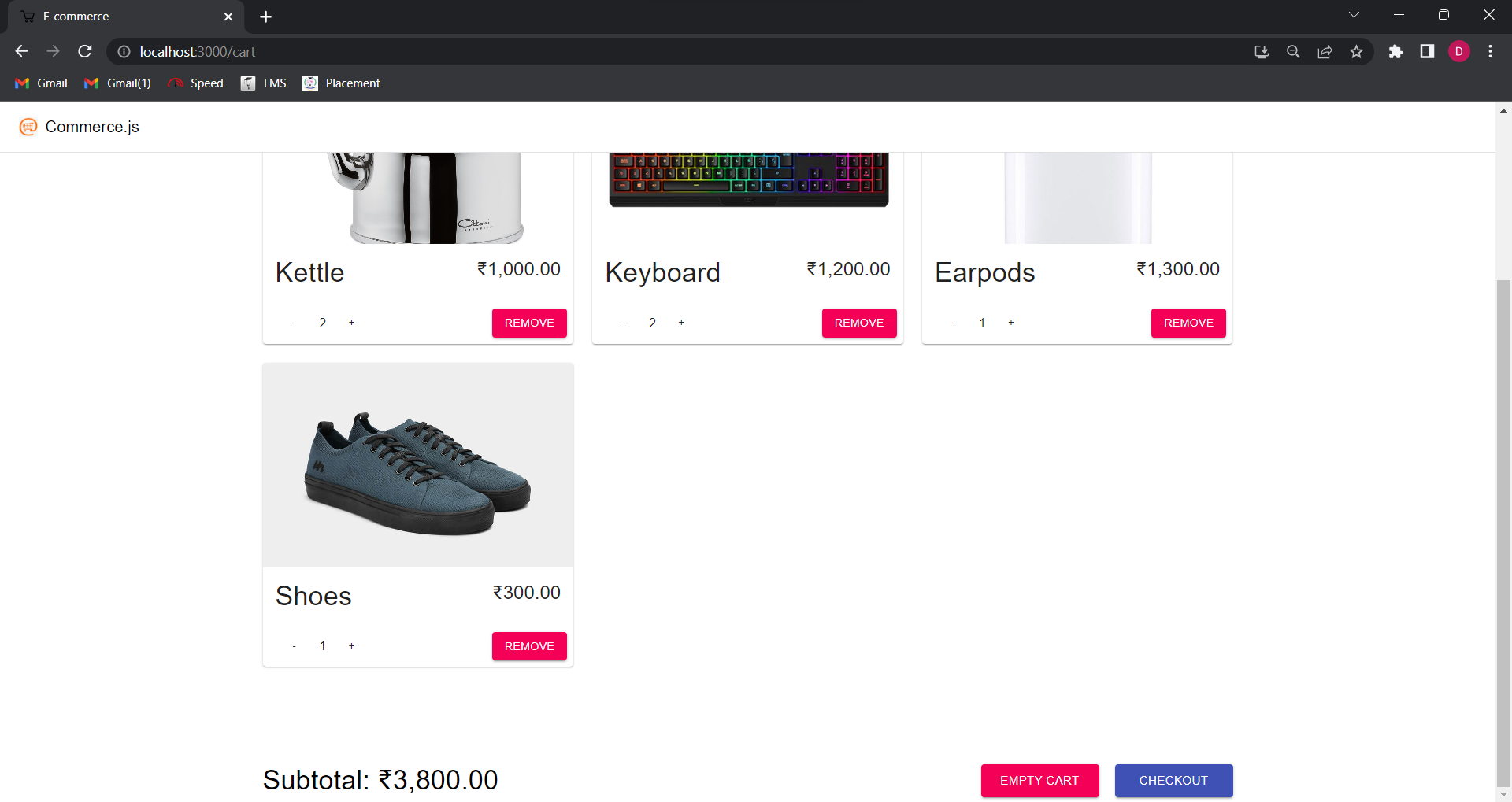
The Commerce.js API provides a set of functions that you can use in your frontend code to retrieve data and perform actions like adding products to the cart and processing orders. You can use the commerce object to access these functions in your JavaScript code. For example, you can use commerce.products.list() to retrieve a list of all products, and commerce.cart.add() to add a product to the cart.



*Fig 4.1 Home Page*

**4.1.2 Cart Page**

The cart page is a crucial part of any ecommerce website as it is the final step in the purchasing process. This page provides customers with an overview of the items they have added to their cart and allows them to make any changes before proceeding to checkout. It is important to design this page in a way that is easy to understand and navigate so that customers can complete their purchase with ease.



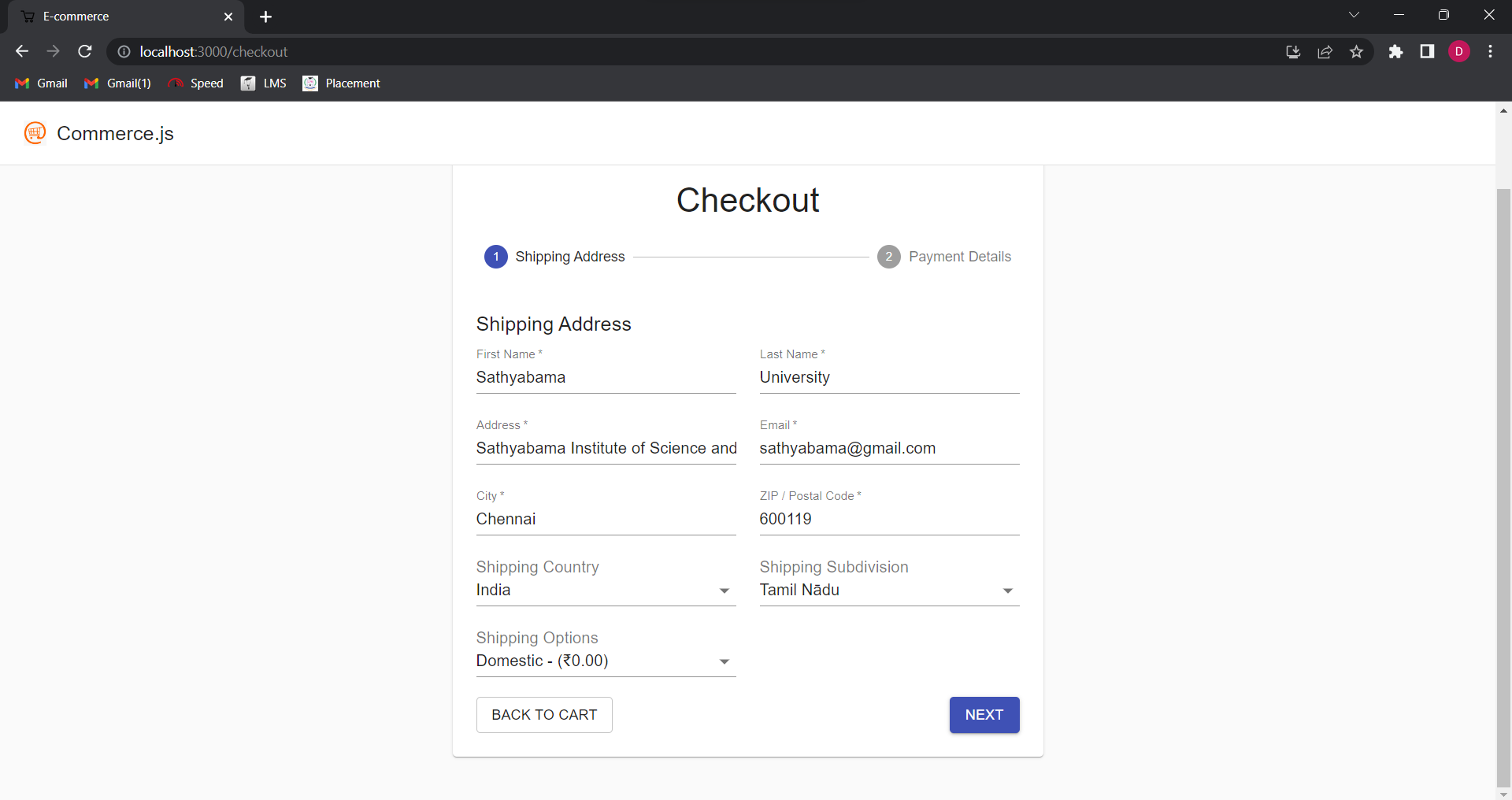
*Fig 4.2 Cart Page*

***4.1.3 Checkout Page***

Stripe.js is a powerful tool that allows businesses to easily accept payments on their websites or mobile apps. One of the key components of any online payment process is the checkout page, which is the final step in the payment process where customers enter their payment information and complete their transaction.

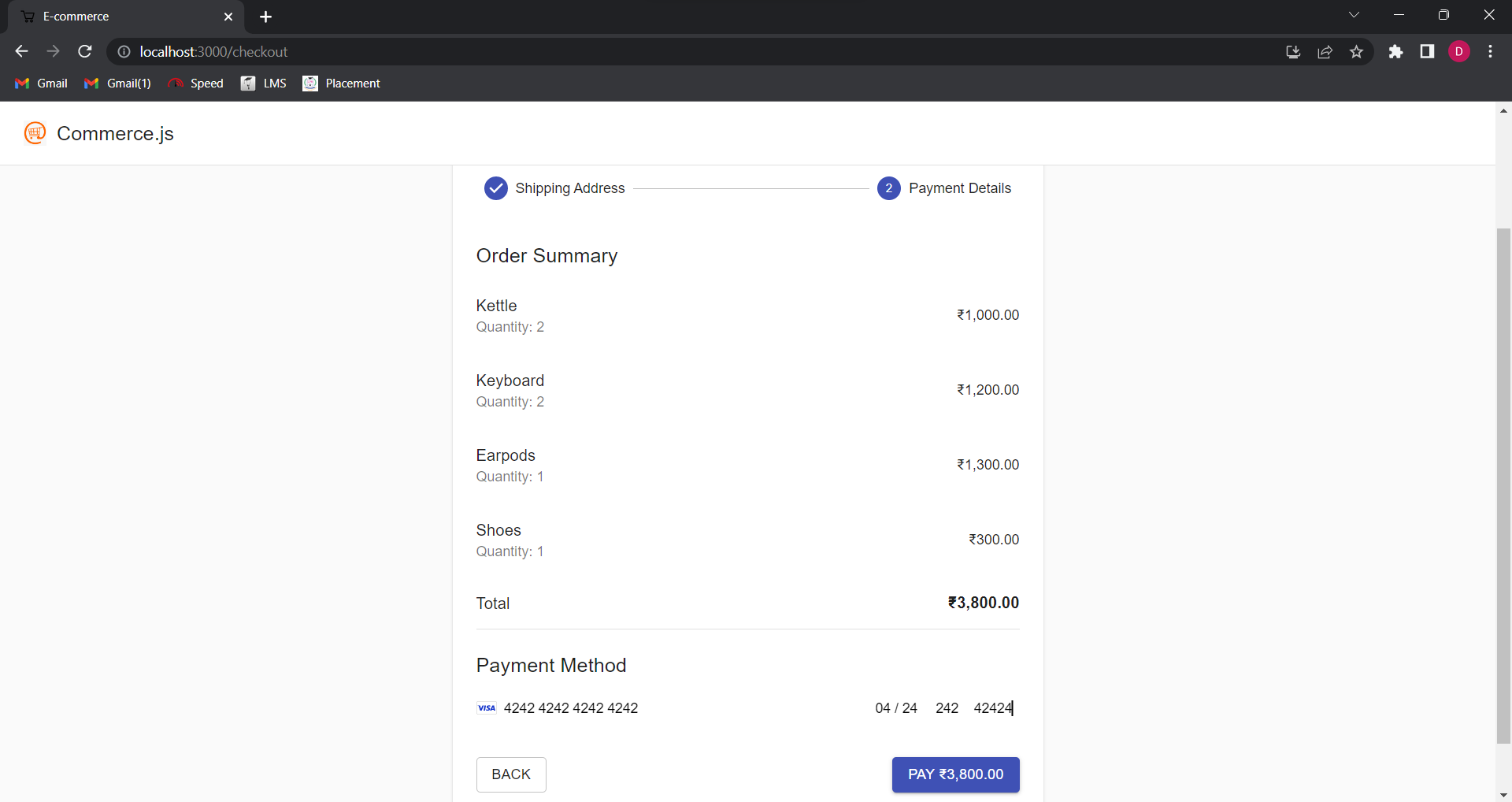
With Stripe.js, implementing a checkout page is made easy through its pre-built checkout form that can be customized to fit the look and feel of your website. The checkout form is designed to provide a seamless payment experience for customers while also ensuring the security of their payment information.

When a customer completes a transaction on the checkout page, Stripe.js securely collects their payment information and processes the payment. Stripe.js then returns a response to your website, indicating whether the payment was successful or not.

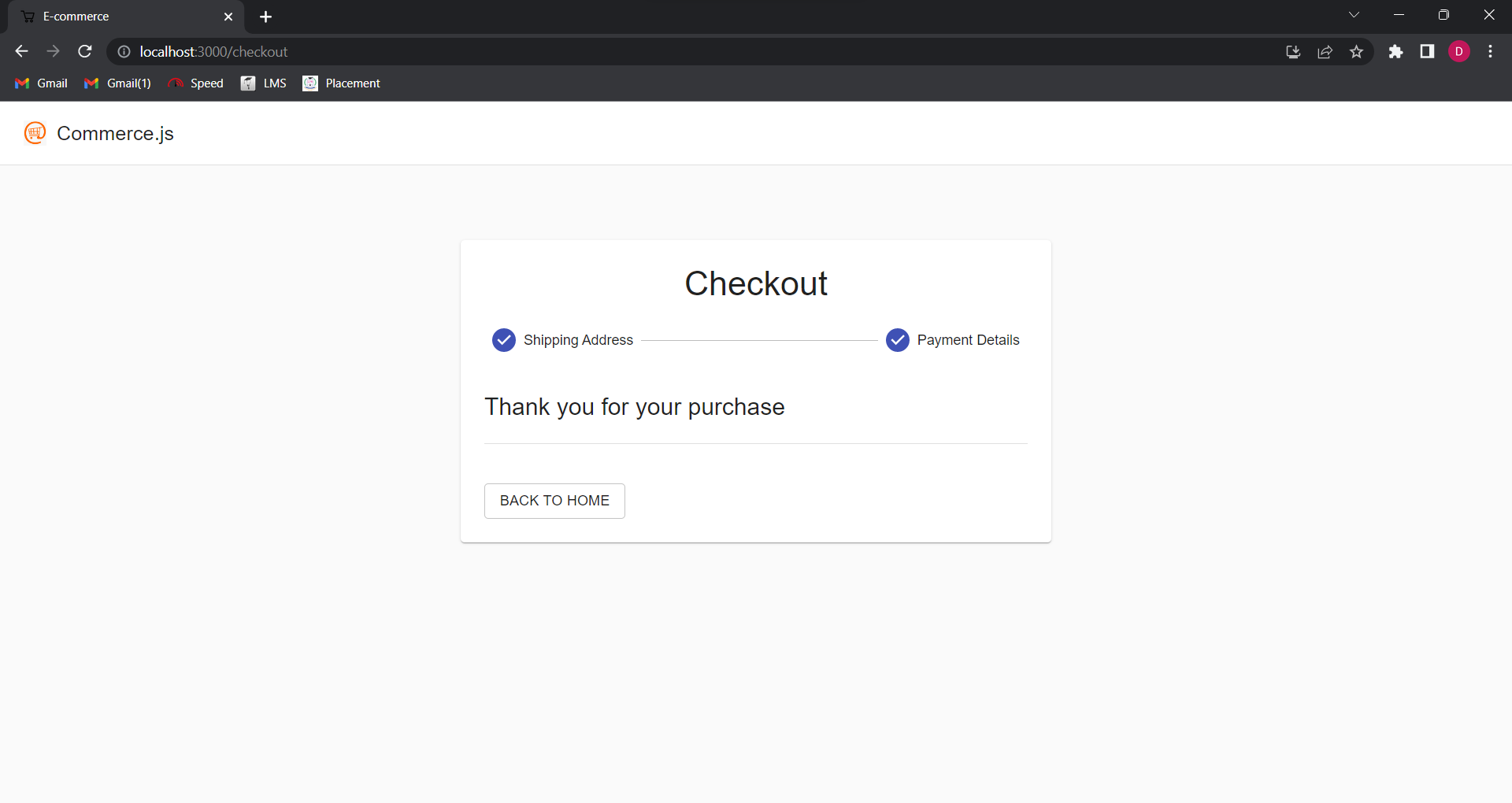


*Fig 4.3 Checkout Page*

Stripe.js provides an easy and convenient way to test payment processing on your website or mobile app before going live with real transactions. This is important because it allows you to ensure that everything is working correctly and that payments are being processed securely.



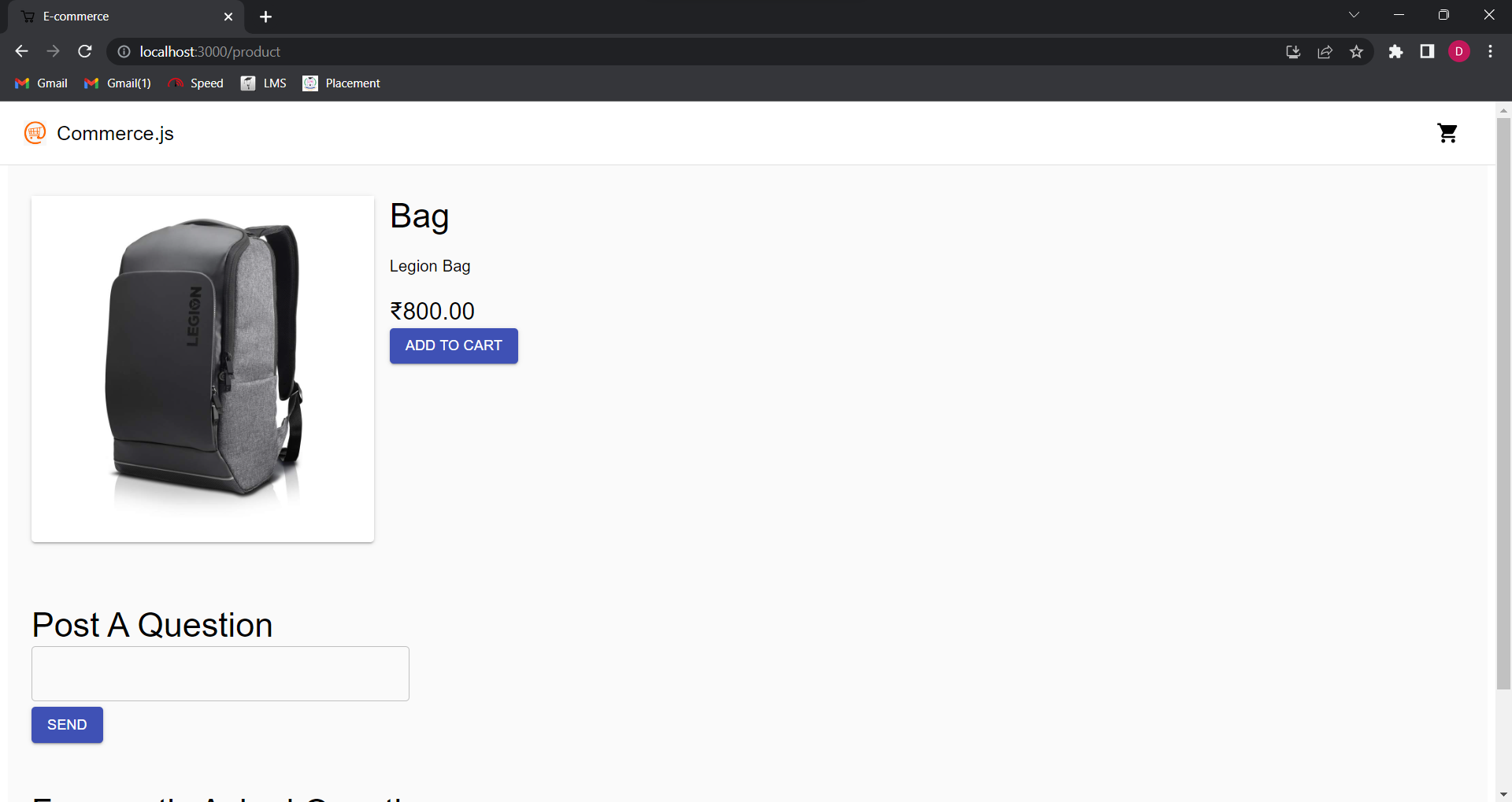
*Fig 4.4 Payment Details*



*Fig 4.5 Confirmation*

***4.1.4 Product’s Page***

A product page is a webpage on a company or business's website that showcases a specific product that they are selling. The page typically includes a product description, pricing information, images or videos of the product, specifications, and any other relevant information about the product. The purpose of a product page is to provide customers with enough information about the product to help them make an informed purchasing decision.

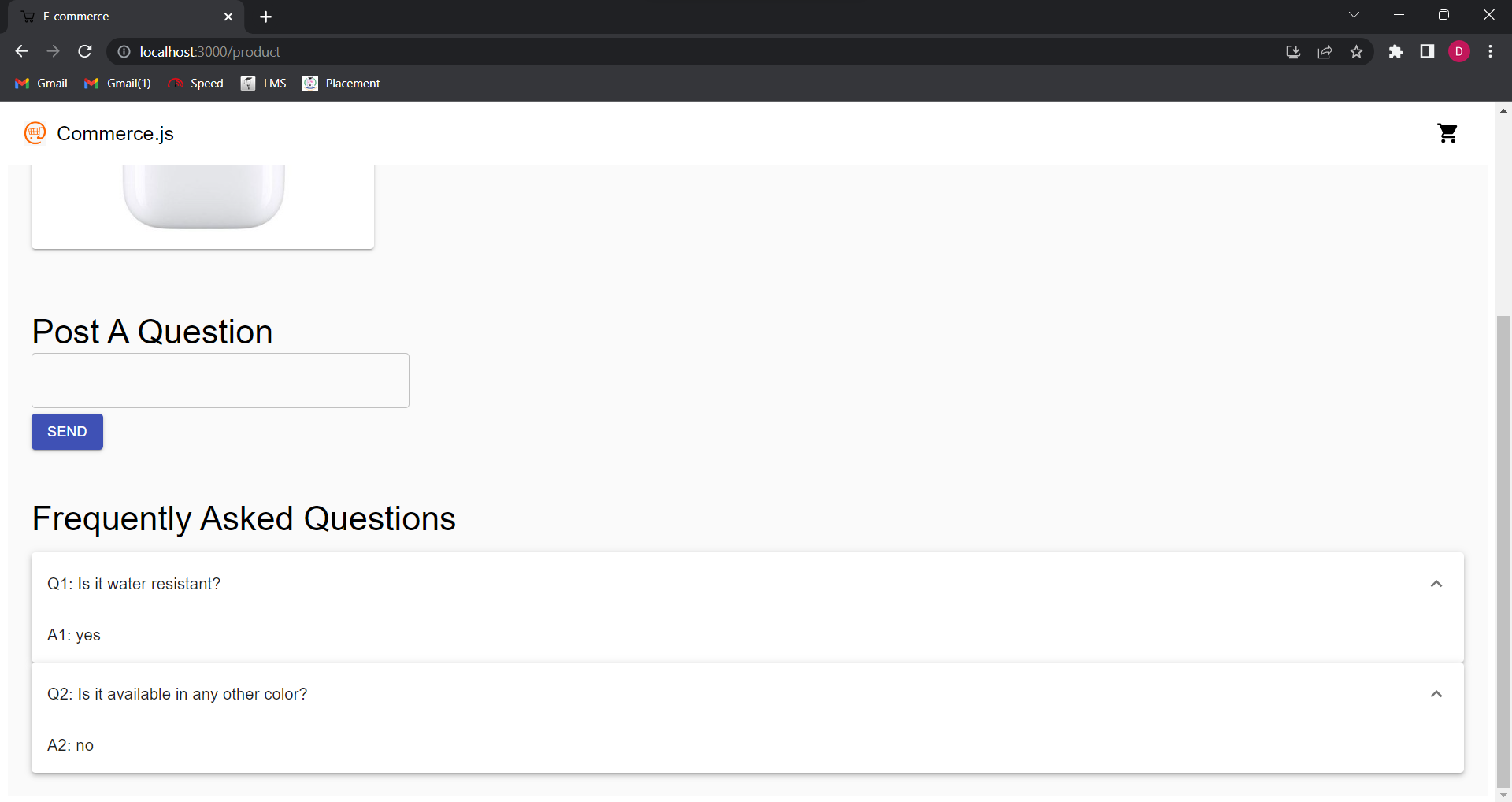


*Fig 4.6 Product’s Page*

***4.1.5 QnA Section***

The Q&A (Question and Answer) section of an e-commerce website is an important feature that allows potential customers to ask questions about a particular product before making a purchase. This section provides a platform for customers to seek clarification on any doubts they may have about the product, and also for existing customers to share their experience or opinion about the product.

A well-managed Q&A section can help increase customer trust in the company and its products, and also helps to reduce the number of customer service inquiries. It is important for the company to have a dedicated team to monitor and respond promptly to the questions asked by customers in this section.



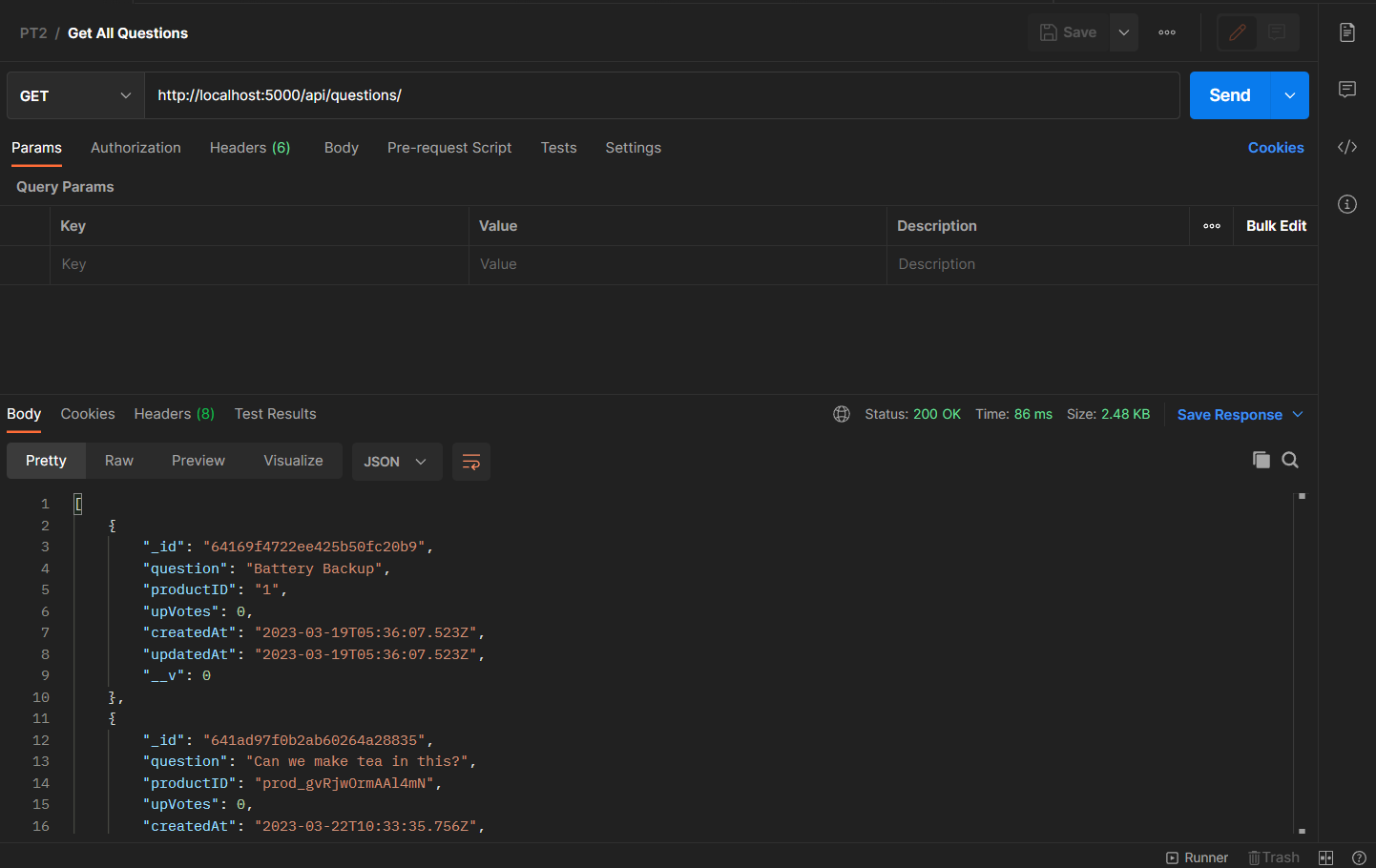
*Fig 4.7 QnA Section*

## 4.2 BACKEND

List of APIs created for QnA Service:

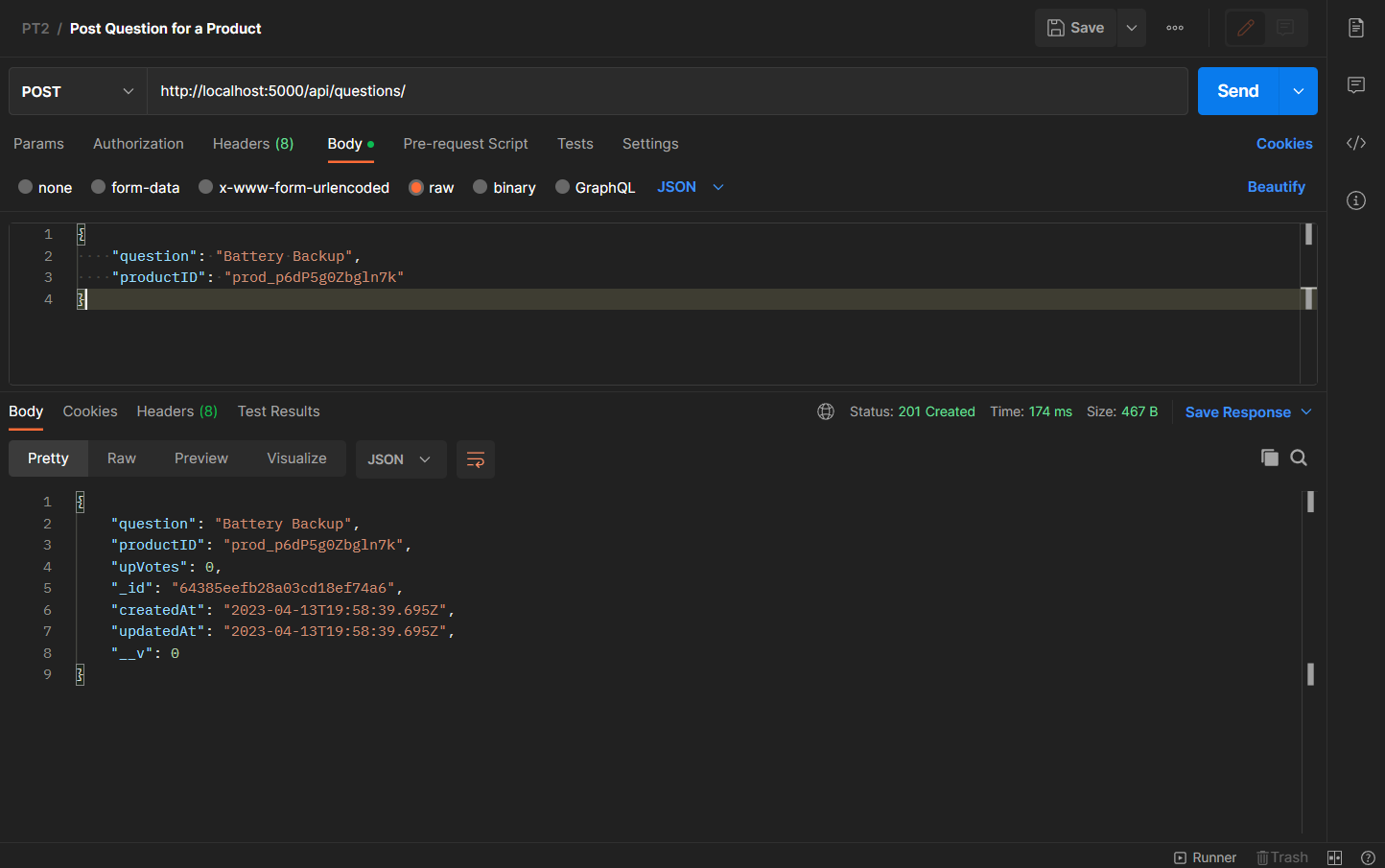
***4.2.1 Get All Questions***

Gives you all the questions posted from the frontend.

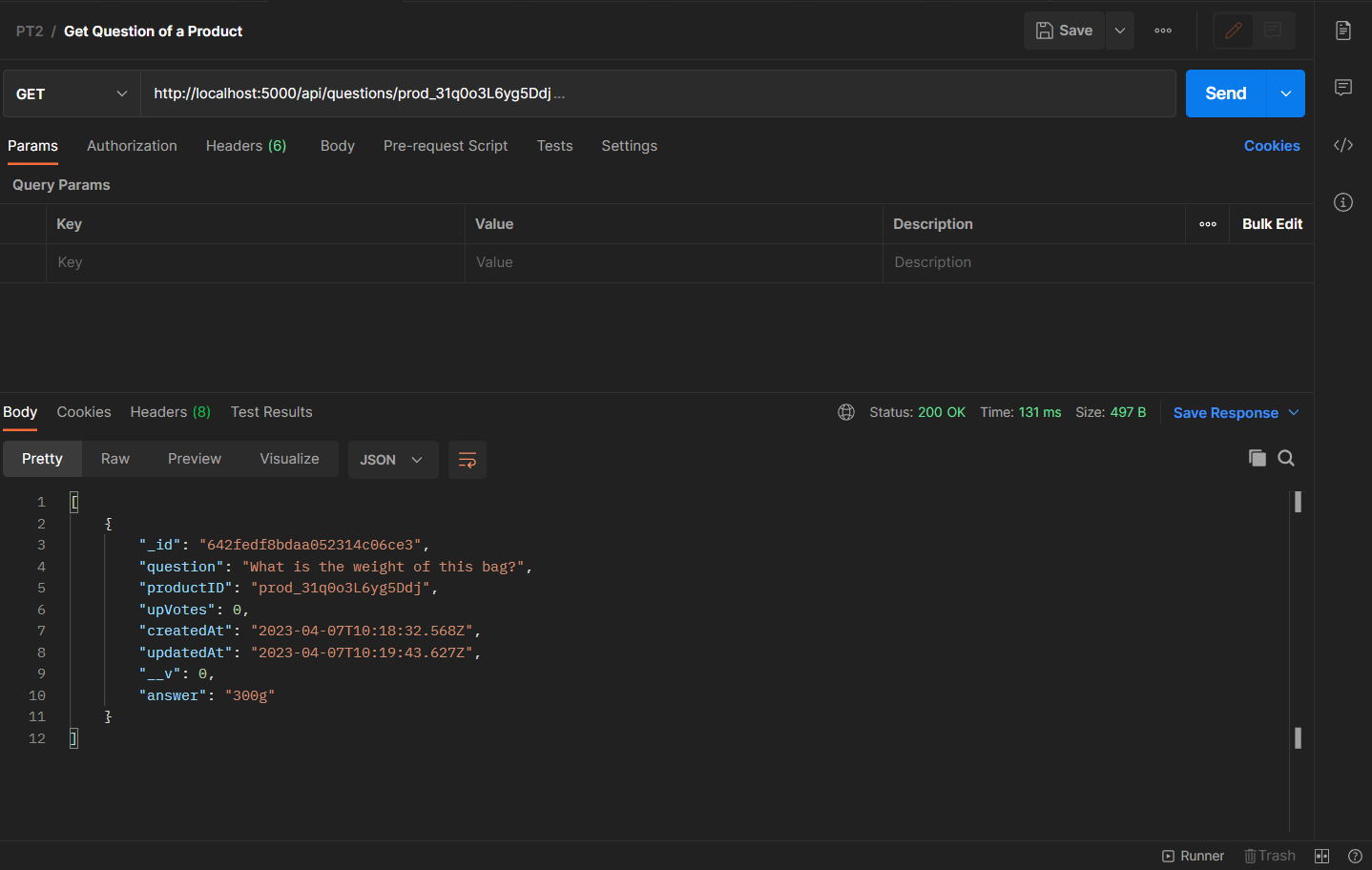


*Fig 4.8 Get all Questions*

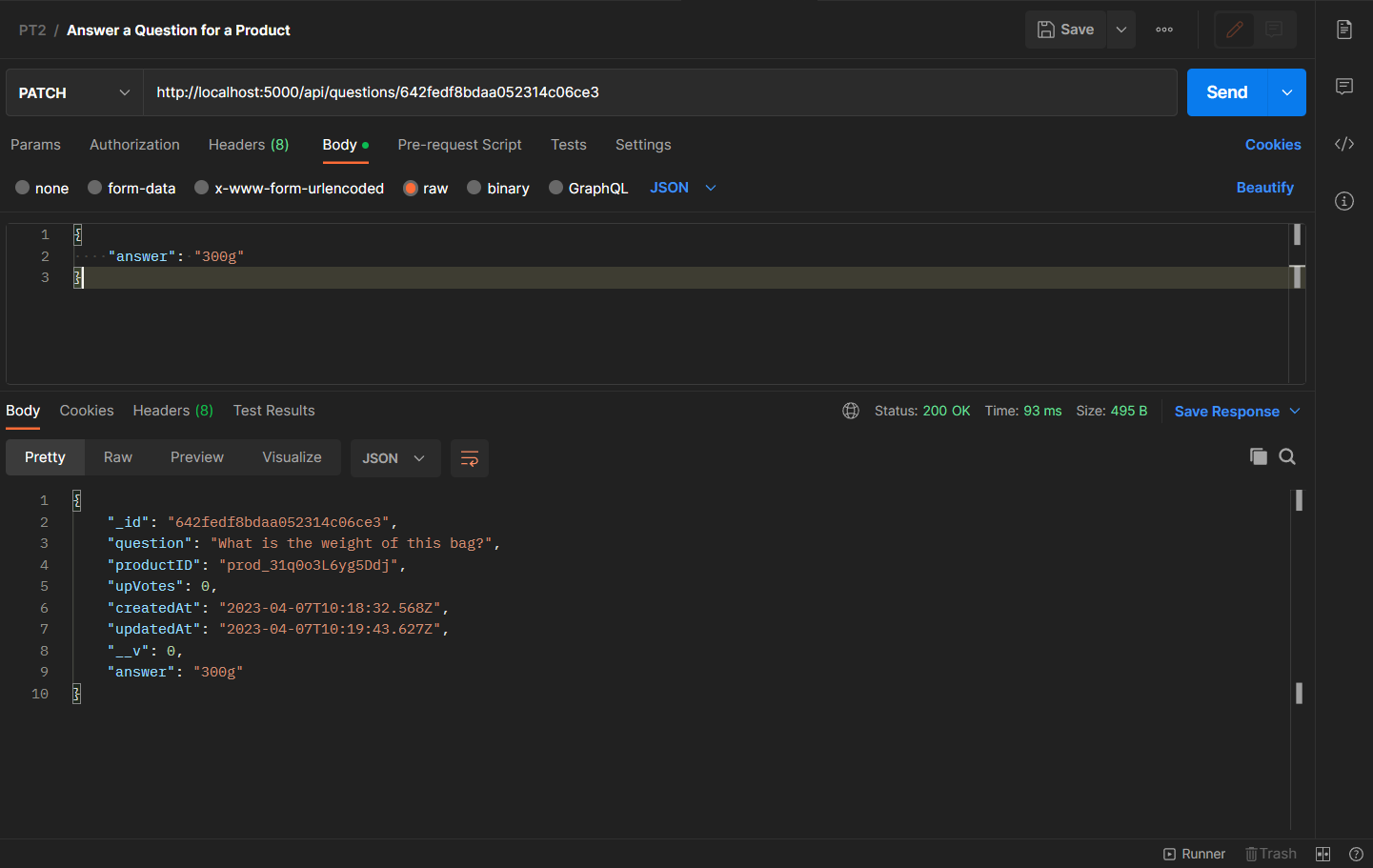
***4.2.2 Post Question for a Product***



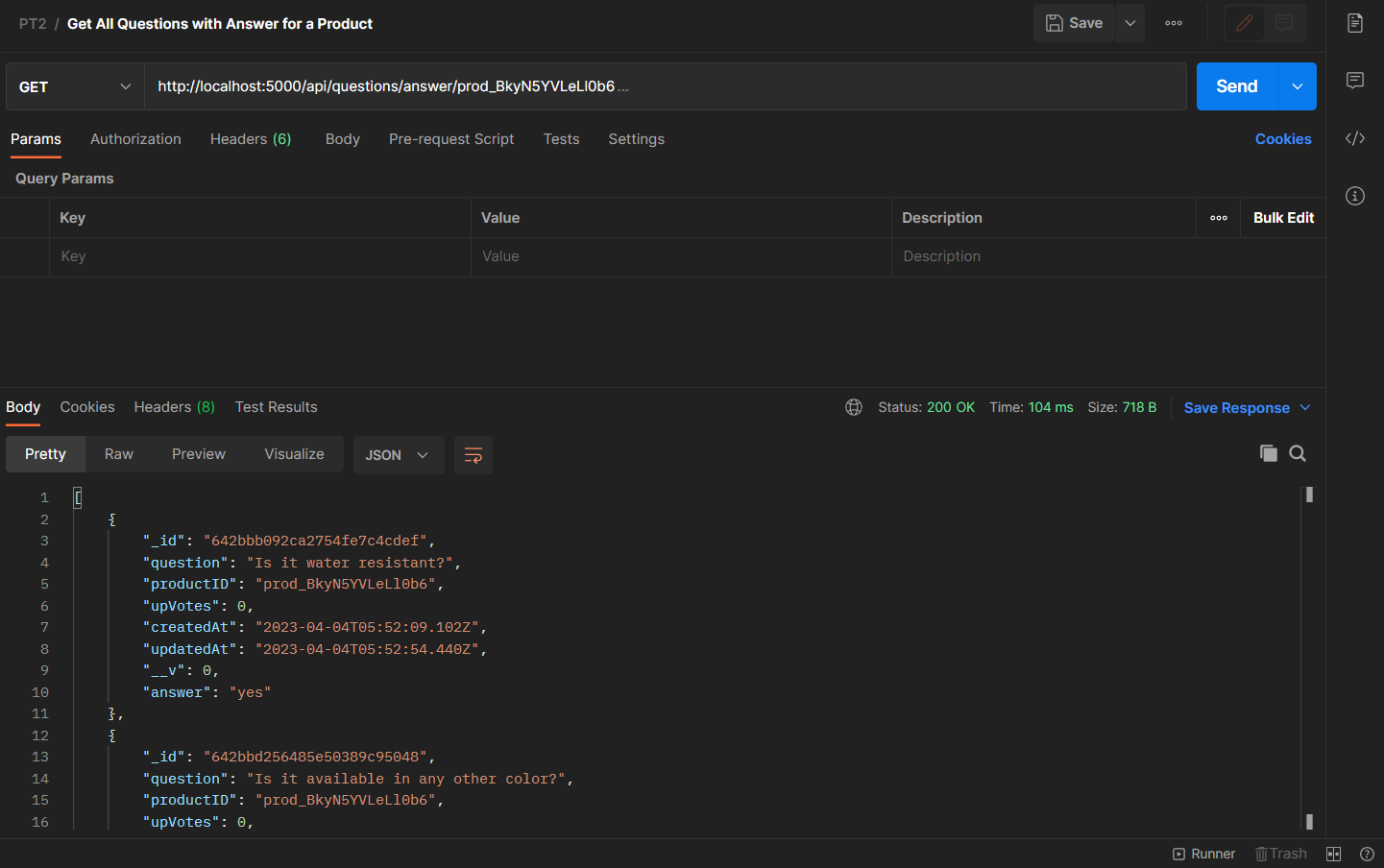
***4.2.3 Get Questions of a Product***

******

***4.2.4 Answer a Question for a Product***



***4.2.5 Get all Question with Answer for a Product***

******

# CHAPTER 5 – SUMMARY AND CONCLUSION

The QnA Backend Service project is a web-based application that provides a question and answer service for customers on an ecommerce platform. The application uses React for the frontend development, and Node.js, Express.js, and MongoDB for the backend development. The aim of the project is to enhance the user experience on an ecommerce platform by providing a platform for customers to ask and answer questions related to products.

The project features an intuitive user interface that allows customers to browse through questions and answers related to products. Customers can also submit new questions or answers and upvote or downvote existing ones. The project also includes a search functionality that allows customers to search for specific questions or products.

The "QnA Backend Service for Ecommerce" project is a valuable tool for ecommerce platforms to enhance their user experience by providing a question and answer service for customers. The project uses modern web technologies like React, Node.js, Express.js, and MongoDB, making it scalable and easy to maintain.

With the project's intuitive user interface and search functionality, customers can easily find answers to their questions and get insights into the products they are interested in. The ability for customers to submit new questions and answers, and upvote or downvote existing ones, makes the project a valuable tool for building a community around products.

Overall, the QnA Backend Service project is an excellent addition to any ecommerce platform that is looking to provide a better user experience for their customers.

# REFERENCES

1. React Documentation. https://reactjs.org/docs/getting-started.html. Accessed April 12, 2023.
2. Node.js Documentation. https://nodejs.org/en/docs/. Accessed April 12, 2023.
3. "Improving Customer Experience on Ecommerce Sites." Nielsen Norman Group. https://www.nngroup.com/articles/ecommerce-experience/. Accessed April 12, 2023.
4. "The Impact of Online Customer Experience on Customer Satisfaction and Loyalty." International Journal of Customer Relationship Marketing and Management, vol. 5, no. 3, 2014, pp. 36-54.
5. "Stripe.js Documentation." https://stripe.com/docs/js. Accessed April 12, 2023.
6. "Building a Simple E-commerce Store with Node.js and Stripe." Scotch.io. https://scotch.io/tutorials/building-a-simple-e-commerce-store-with-nodejs-and-stripe. Accessed April 12, 2023.
7. "Using Express.js to Build a Web Application with Node.js." DigitalOcean. https://www.digitalocean.com/community/tutorials/using-express-to-build-a-web-application-with-node-js. Accessed April 12, 2023.
8. "MongoDB Documentation." https://docs.mongodb.com/. Accessed April 12, 2023.
9. "10 Tips to Improve Your Ecommerce Customer Experience." BigCommerce. https://www.bigcommerce.com/blog/ecommerce-customer-experience/. Accessed April 12, 2023.
10. "React Bootstrap Documentation." https://react-bootstrap.github.io/. Accessed April 12, 2023.

# APPENDIX

## SOURCE CODE

***index.js***

import express from "express";

import cors from "cors";

import mongoose from "mongoose";

import questionRoutes from "./routes/questionRoutes.js"

const app = express();

app.use(cors());

app.use(express.json());

app.use("/api/questions", questionRoutes);

app.get("/", (req, res) => {

    res.send("App is running");

});

const PORT = process.env.PORT || 5000;

const CONNECTION\_URL = "mongodb+srv://dibyanshu1607:pKXxiIV0LHo94dBl@cluster0.movti6w.mongodb.net/?retryWrites=true&w=majority";

mongoose.connect(CONNECTION\_URL, { useNewUrlParser: true })

    .then(() => {

        console.log("DB connected");

        app.listen(PORT, () => console.log(`Server running on PORT ${PORT}`));

    })

    .catch((error) => console.log(error));

***questionRoutes.js***

import express from "express";

import { getQuestions, postQuestion, getQuestionsOfProduct, deleteQuestions, deleteQuestion, answerQuestion, getQuestionsWithAnswerOfProduct } from "../controllers/questionController.js"

const router = express.Router();

router.get("/", getQuestions);

router.get("/:productID", getQuestionsOfProduct);

router.post("/", postQuestion);

router.delete("/", deleteQuestions);

router.delete("/:id", deleteQuestion);

router.patch("/:id", answerQuestion);

router.get("/answer/:productID", getQuestionsWithAnswerOfProduct);

export default router;

***question.js***

import mongoose from "mongoose";

const questionSchema = mongoose.Schema({

    question: {

        type: String,

        required: true,

    },

    productID: {

        type: String,

        required: true,

    },

    answer: String,

    upVotes: {

        type: Number,

        default: 0

    }

}, { timestamps: true });

const Question = mongoose.model("question", questionSchema);

export default Question;

***questionController.js***

import Question from "../models/question.js";

export const getQuestions = async (req, res) => {

    try {

        const result = await Question.find();

        res.status(200).json(result);

    } catch (error) {

        console.log(error);

        res.status(500).json({ message: "Something went wrong" });

    }

};

export const postQuestion = async (req, res) => {

    const { question, productID } = req.body;

    try {

        const result = await Question.create({

            question,

            productID

        })

        res.status(201).json(result);

    } catch (error) {

        console.log(error);

        res.status(500).json({ message: "Something went wrong" });

    }

};

export const getQuestionsOfProduct = async (req, res) => {

    const productID = req.params.productID;

    try {

        const result = await Question.find({ productID });

        res.status(200).json(result);

    } catch (error) {

        console.log(error);

        res.status(500).json({ message: "Something went wrong" });

    }

};

export const deleteQuestions = async (req, res) => {

    try {

        const result = await Question.deleteMany({});

        res.status(200).json(result);

    } catch (error) {

        console.log(error);

        res.status(500).json({ message: "Something went wrong" });

    }

};

export const deleteQuestion = async (req, res) => {

    const id = req.params.id;

    try {

        const result = await Question.findByIdAndDelete(id);

        res.status(200).json(result);

    } catch (error) {

        console.log(error);

        res.status(500).json({ message: "Something went wrong" });

    }

};

export const answerQuestion = async (req, res) => {

    const id = req.params.id;

    try {

        const result = await Question.findByIdAndUpdate(id, req.body);

        res.status(200).json(result);

    } catch (error) {

        console.log(error);

        res.status(500).json({ message: "Something went wrong" });

    }

};

export const getQuestionsWithAnswerOfProduct = async (req, res) => {

    const productID = req.params.productID;

    try {

        const result = await Question.find({ productID });

        const newResult = result.filter(item => item.answer !== undefined);

        res.status(200).json(newResult);

    } catch (error) {

        console.log(error);

        res.status(500).json({ message: "Something went wrong" });

    }

};