4R/2023-24/Roll No: 65

A Seminar Report

On

CREATING A RESTFUL API using Node.js and Express.js : Best Practices and Implementation

Submitted

In partial fulfillment of the requirements for the degree of

Bachelor of Engineering

in

Computer Science and Engineering

Sant Gadge Baba Amravati University

Submitted by

Rudransh Santosh Nemade

Under the esteemed guidance of

Prof. C. M. Mankar

Department of Computer Science and Engineering

Shri Sant Gajanan Maharaj College of Engineering, Shegaon,

Dist.- Buldhana – 444 203 (Maharashtra)

(2023 2024)

SHRI SANT GAJANAN MAHARAJ COLLEGE OF ENGINEERING,

SHEGAON – 444 203 (M.S.)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

`

CERTIFICATE

This is to certify Rudransh Santosh Nemade, student of final year B.E. in the year 2023-24 of Computer Science and Engineering Department of this institute has completed the Seminarreport and presentation entitled "**CREATING A RESTFUL API using Node.js and Express.js: Best Practices and Implementation**" based on syllabus and has submitted a satisfactory account of his work in this report which is recommended for the partial fulfillment of degree of Bachelor of Engineering in Computer Science and Engineering.

Prof. C. M. Mankar

Guide

Dr. P. K. Bharne

Seminar Coordinator

Dr. J. M. Patil

Head
Department of CSE

Acknowledgement

It is our proud privilege and duty to acknowledge the kind of help and guidance received from several people in preparation of this report. It would not have been possible to prepare this report in this form without their valuable help, cooperation and guidance.

First and foremost, I wish to record our sincere gratitude to Management of this college and to our beloved Principal, **Dr. S. B. Somani,** for his constant support and encouragement in preparation of this report and for making available library and laboratory facilities needed to prepare this report.

Our sincere thanks to **Dr. J. M. Patil**, Head of Department of Computer Science and Engineering, for his valuable suggestions and guidance throughout the period of this report.

I express my sincere gratitude to my guide, **Prof C. M. Mankar**, for guiding me in investigations for this seminar and in carrying out relevant work. Our numerous discussions were extremely helpful. I received his esteem guidance, encouragement, and inspiration.

I sincerely thank to **Dr. P. K. Bharne**, Seminar Coordinator for supporting this seminar work. His contribution and technical support in preparing this report is greatly acknowledged.

Last but not the least, I wish to thank our parents for financing my studies in this college as well as for constantly encouraging me to learn engineering. Their personal sacrifice in providing this opportunity to learn engineering is gratefully acknowledged.

Place: Shegaon Rudransh Nemade (65)

Date: 6-10-2023

Table of Contents

Sr. No.	Title	Page No
	Abstract	i
	List of Figures	ii
	List of Abbreviations	iii
1	Introduction	1
2	Literature Survey	4
3	Research Approach	5
4	Practical Execution	19
5	Results and Discussion	27
6	Conclusion	33
7	Future Scope	34
8	References	35

ABSTRACT

In the rapidly evolving landscape of online commerce, this seminar report unfolds the complexities of crafting a robust API for digital storefronts. Focused on the crucial task of product management. This seminar report explores how to create a powerful API for an online store. It focuses on managing products and uses the latest technologies like Node.js, Express.js, MongoDB, and Mongoose. This studydemonstrates how these technologies work together to build a flexible and responsive API. At its core, this API serves as the backbone of user interaction. TheAPI allows users to easily filter, search, and sort products by their prices, names and any other parameter. It also enables users to pick specific product details, making the shopping experience more personalized. To handle large amounts of data efficiently, the API includes pagination, ensuring quick and smooth access to product information. A significant achievement of this project is the deployment ofthe API on the railway.app platform. This cloud-based hosting service ensures the API is always available and can handle many users at once. The API's design is optimized for handling multiple requests simultaneously, making it suitable for busy online stores. This research provides valuable insights into modern web development, showcasing the potential of technologies like Node.js, Express.js, MongoDB, and Mongoose. By adopting best practices in API design and deployment, this study illustrates how these technologies can transform the online shopping experience.

Keywords— RESTful web service REST, RESTful, Web Services, API, HTTP, Web, CRUD

List of Figures

Fig. No	***Name of the Figures***	***Page No***
Fig 1.1	Restful API Client-Server	2
Fig 3.6.1	Client-API-Server	6
Fig 3.8.1	Breakdown of Create Server Code	8
Fig 3.8.2	Routes	9
Fig 3.8.3	Controllers	10
Fig 3.8.4	Different API Testing by Postman	11
Fig 3.8.5	Connecting Server to Database	12
Fig 3.8.6	MongoDB Hierarchy	13
Fig 5.3	Public APIs since 2005	32
Fig 5.4	Distribution of API protocols and Styles	33

List of Abbreviations

1.	API	Application Programming Interface
2.	REST	Representational State Transfer
3.	HTTP	Hyper Text Transfer Protocol
4.	JSON	Java Script Object Notation
5.	URL	Universal Resource Locator

1.INTRODUCTION

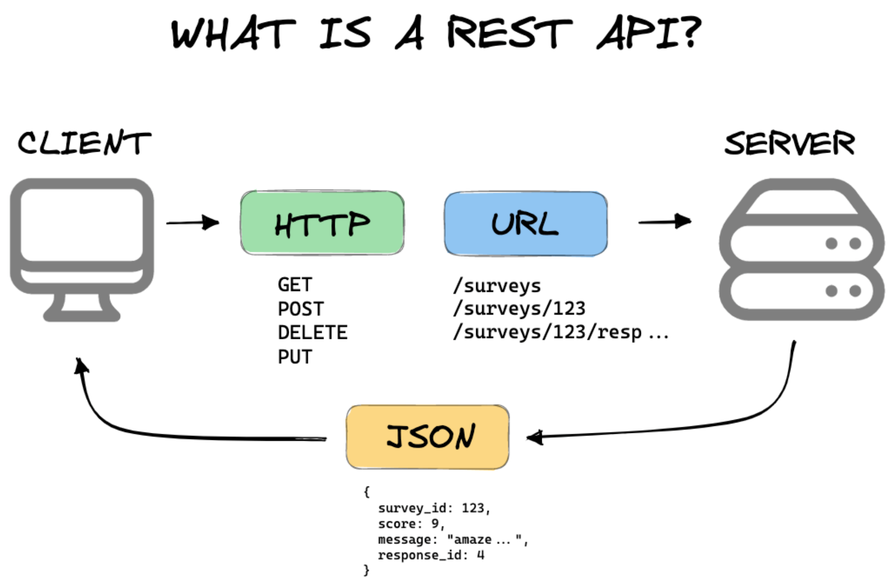
In the dynamic world of web development, the creation of a powerful E-commerce Product API emerges as a vital frontier. This report is an in-depth exploration that merges the essence of modern web technologies with the demands of the bustling digital market. At its core, this study delves into the complexities of building an E-commerce Product API, a foundational element in online retail platforms. Fueled by the fusion of cutting-edge technologies like Node.js, Express.js, MongoDB, and Mongoose, this API not only manages product data but also elevates the user experience in the digital marketplace.

This exploration navigates the technical intricacies of API development, unraveling the nuances of asynchronous operations in Node.js and the art of route handling with Express.js. It delves into MongoDB, showcasing data storage and retrieval techniques, alongside the elegance of schema modeling using Mongoose. The API's functionalities, including user request handling, data filtering, searching, sorting, and pagination, are dissected meticulously, ensuring a deep understanding of each layer. Moreover, the report sheds light on deployment strategies, emphasizing cloud-based hosting platforms such as railway.app. This ensures not just availability but also scalability, making the API capable of accommodating the diverse needs of a thriving online store.

This comprehensive guide serves as a bridge between the technological intricacies and the real-world applications in the realm of E-commerce. By merging the complexities of web development with the dynamic demands of the digital market, this report equips developers and businesses alike with the knowledge to architect APIs that redefine the online shopping experience.

1.1 Introduction to Restful API

A RESTful API, short for Representational State Transfer API, is a set of rules and conventions for building and interacting with web services. It follows the principles of REST, emphasizing a stateless client-server communication where data and functionalities are treated as resources. RESTful APIs use standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on these resources, enabling seamless and scalable interaction between different software systems over the internet. RESTful APIs are known for their simplicity, flexibility, and ease of integration, making them a popular choice for web applications and services.



```
{
  survey_id: 123,
  score: 9,
  message: "amaze ... ",
  response_id: 4
}
```

Fig 1.1 Restful API – Client-Server

1.2 Aim and Objective

Aim: To enhance product data management in digital commerce by implementing efficient RESTful APIs, ensuring seamless data storage, retrieval, and user experience.

Objectives:

1. **Develop a Robust RESTful API:** Create a reliable and secure API framework following REST principles to handle product data operations effectively.
2. **Efficient Data Storage:** Implement data storage mechanisms that ensure secure and organized storage of product information in the database.
3. **Swift Data Retrieval :** Enable quick and accurate retrieval of specific product details using RESTful API endpoints, optimizing response times for user queries.
4. **Enhance User Experience:** Implement advanced functionalities like searching, filtering, and personalized data retrieval to improve user interactions with the digital platform.
5. **Scalability and Flexibility:** Design the API architecture to be scalable, accommodating growing data needs, and flexible, allowing for future enhancements and modifications.

1.3 Scope

This study focuses on implementing and optimizing RESTful APIs for efficient product data management in digital commerce. It encompasses API development, secure data storage, fast retrieval, and user experience enhancements. The scope also includes scalability, security measures, performance optimization, and thorough documentation, ensuring seamless integration and improved functionality for online businesses.

2.LITERATURE SURVEY

Source	*Title of Research Paper*	*Methodology*	*Key Findings*	*Limitations*
IEEE Xplore: (December 2021)	Implementation of RESTful API Web Services Architecture in Take away Application Development.	The Takeaway application consists of a website as a backend and an Android-based as a frontend..	Comparison of responses and requests to SOAP and REST architectures	The RESTful API architecture in the takeaway application might face limitations in handling a massive influx of users simultaneously, impacting the system's scalability during peak hours..
International Research Journal of Engineering and Technology (IRJET)	Survey Paper: Framework of REST APIs	The framework is created to manufacture an adaptable REST API for Ecommerce to serve any frontend customer.	Introduced many of REST API's designers utilized for making a site or some other framework.This paper additionally introduced the approval procedures like OAuth Authorization.	The paper lack in-depth exploration of specific REST API frameworks due to its broad scope.
Research GATE	REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices	More than 78GB of plain HTTP calls are targeted at REST APIs, and characterize the usage patterns that emerge from the logged data so as to compare them with guidelines and principles.	Provides insight into the compliance of state-of-the-art APIs with theoretical Web engineering principles and guidelines, knowledge that a↲ects how applications should be developed to be scalable and robust.	The study is based on a specific dataset, which could limit the generalizability of the findings to other diverse API environments.

Table 1: Literature Survey

3. Research Approach:

3.1 What is Node.js ?

- Open-source, cross-platform JavaScript runtime environment.
- Built on Chrome's V8 JavaScript engine.
- Enables server-side JavaScript execution.
- Asynchronous and event-driven, enhancing efficiency.
- Ideal for I/O-intensive operations.
- Widely used for building real-time applications, APIs, and microservices.
- Non-blocking and lightweight, ensuring scalability.

3.2 What is Express.js?

- Web application framework for Node.js.
- Simplifies server-side application development.
- Provides a robust set of features and tools.
- Streamlines route handling and middleware management.
- Enables the creation of RESTful APIs and web applications.
- Highly customizable and flexible.

3.3 What is Mongoose?

- Object Data Modeling (ODM) library for MongoDB and Node.js.
- Simplifies interactions with MongoDB databases.
- Provides a schema-based solution for data modeling.
- Offers validation, casting, and other powerful features.
- Enhances code readability and maintainability.
- Allows defining data models using JavaScript objects.
- Facilitates querying and manipulating MongoDB data.

3.4 What is MongoDB?

- NoSQL, document-oriented database.
- Stores data in flexible, JSON-like BSON format.
- Designed for scalability and high performance.
- Supports dynamic schema for data flexibility.
- Enables complex queries and aggregations.
- Ideal for handling large volumes of unstructured or semi-structured data.

3.5 What are Routes?

- Path Definitions: Routes specify URLs (endpoints) within a web application.
- Request Handling: Determine how the server responds to client requests at specific endpoints.
- Mapped Functions: Routes link to specific functions (handlers) to process incoming requests.
- HTTP Methods: Handle different methods like GET, POST, PUT, DELETE for data operations.
- Essential in APIs: Integral part of APIs, defining interaction points for clients and servers.

3.6 Client And Server

Client: A client is a computer program or device that requests services or resources from a server over a network, typically through a web browser or application.

Server: A server is a computer program or device that provides services or resources to clients over a network, fulfilling requests made by clients and delivering data or processing results.

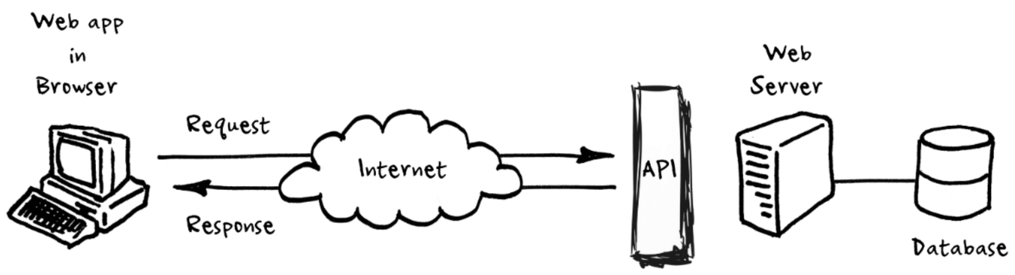


Fig 3.6.1 Client API Server

3.7 Architectural Constraints of RESTful API:

There are six architectural constraints which makes any web service are listed below:

A. Uniform Interface:

A resource in the system should have only one logical URL, and that should provide a way to fetch related or additional data having uniform interface make it easy to be operated with different operating system and different interfaces.

B. Stateless:

Client-server interacts stateless. The client server architecture that is mainly used in the REST API does not contain the information of the any HTTP request and the history of the calls and the session details.

C. Cacheable:

Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side. Client will return the data from its cache for any subsequent request and there would be no need to send the request again to the server. A well-managed caching partially or completely eliminates some client–server interactions, further improving availability and performance.

D. Client-Server:

REST application should have a client-server architecture. A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server, and server is someone who holds the resources and are not concerned with the user interface or user state. They can evolve independently. Client doesn’t need to know anything about business logic and server doesn’t need to know anything about frontend UI.

E. Layered system:

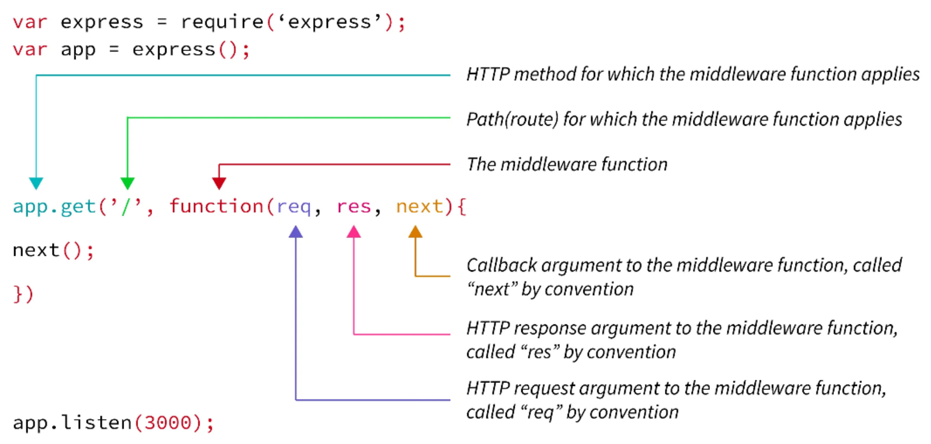
An application architecture needs to be composed of multiple layers. Each layer doesn’t know anything about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server. Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches.

F. Code on demand:

It is an optional feature. According to this, servers can also provide executable code to the client. The examples of code on demand may include the compiled components such as Java Servlets and Server-Side Scripts such as JavaScript.

3.8 Creating a RESTful API for Products: A Step-by-Step Guide

3.8.1 Create a Server with Express.js



```
var express = require('express');
var app = express();

app.get('/', function(req, res, next){
next();
})

app.listen(3000);
```

HTTP method for which the middleware function applies
Path(route) for which the middleware function applies
The middleware function
Callback argument to the middleware function, called "next" by convention
HTTP response argument to the middleware function, called "res" by convention
HTTP request argument to the middleware function, called "req" by convention

Fig 3.8.1 Breakdown of create server code

1. Importing Express: Begin by importing the Express module using require('express').

```
const express = require('express');
```

2. Initializing the Express Application: Create an instance of the Express application using express().

```
const app = express();
```

3. Starting the Server: Use the listen() method to specify a port number and start the server. The server will now listen for incoming HTTP requests on the specified port.

```
app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

3.8.2 Setting Routes in Express.js

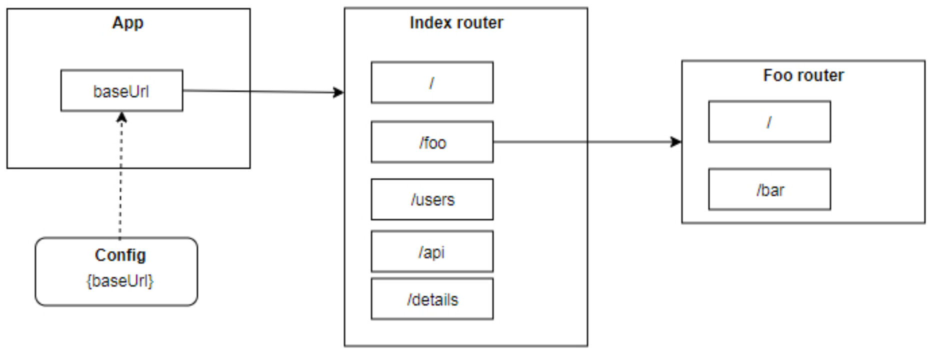


Fig 3.8.2 Routes

1. Defining Routes:

Routes are defined using HTTP methods (get, post, put, delete, etc.) on the Express application object. Each method handles a specific type of request.

```
app.get('/', (req, res) => {
  res.send('This is the homepage');
});
```

2. Route Parameters:

Express supports route parameters, allowing dynamic URL patterns. Parameters are indicated using a colon : in the route definition.

```
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User ID: ${userId}`);
});
```

3. Route Middleware:

Middleware functions can be added to routes. They execute specific tasks before the route handler is invoked, providing flexibility for authentication, logging, and other operations.

```
app.get('/admin', isAdmin, (req, res) => {
  res.send('Welcome to the admin panel');
});
```

3.8.3 Setting Controllers of Routes in Express.js

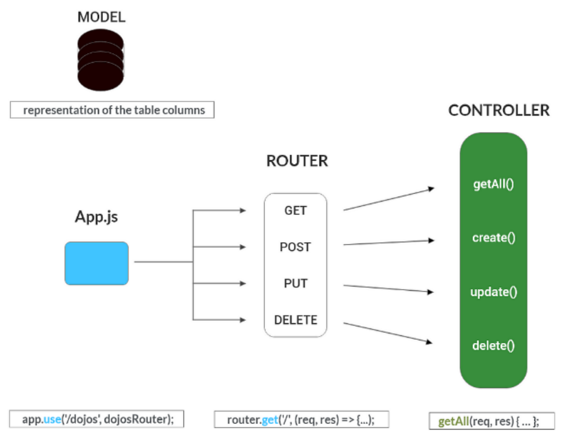

MODEL
representation of the table columns
CONTROLLER
ROUTER
App.js
GET
POST
PUT
DELETE
getAll()
create()
update()
delete()
app.use('/dojos', dojosRouter);
router.get('/', (req, res) => {...});
getAll(req, res) { ... };

Fig 3.8.3 Controllers

1. **Separation of Concerns:** Controllers in Express.js handle the application logic, separating it from route definitions for clarity and maintainability.

2. **Organizing Code:** Controllers encapsulate functions that handle specific route actions, such as retrieving data from databases or processing form submissions.

3. **Modular Structure:** Controllers are modular, allowing developers to reuse code across different routes and keep the application organized and scalable.

Code

```
// Controller File (e.g., userController.js)
exports.getUser = (req, res) => {
  // Logic to retrieve user data from the database
  res.send('User data');
};

// Route File (e.g., userRoutes.js)
const express = require('express');
const router = express.Router();
const userController = require('./userController');

router.get('/user', userController.getUser);
module.exports = router;
```

3.8.4 Postman

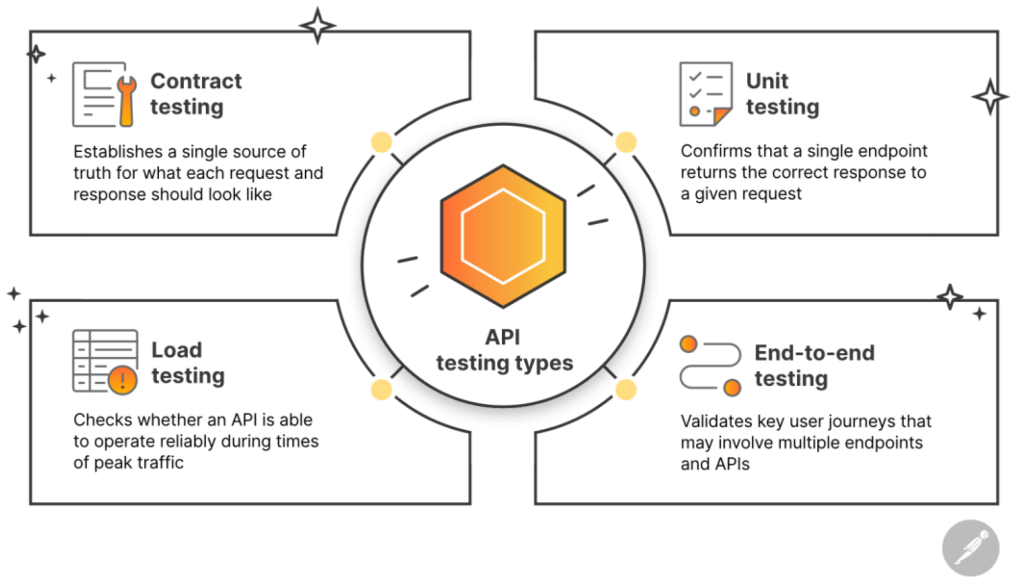

Contract testing
Establishes a single source of truth for what each request and response should look like
Unit testing
Confirms that a single endpoint returns the correct response to a given request
API testing types
Load testing
Checks whether an API is able to operate reliably during times of peak traffic
End-to-end testing
Validates key user journeys that may involve multiple endpoints and APIs

Fig 3.8.4 Differnet API Testing by Postman

- Postman: Postman is a popular API testing tool that allows developers to create, test, and automate API requests.
- Importance: It ensures API endpoints work as expected, validating responses, handling various HTTP methods, and assessing different scenarios.
- Features: Postman supports various request types (GET, POST, PUT, DELETE), parameters, headers, authentication methods, and response validations.
- Workflow: Developers input API endpoints, parameters, and headers in Postman, then send requests to the server. Postman displays server responses, allowing developers to verify API functionality and troubleshoot errors quickly.
- Automation: Postman enables automation of API tests, making it a powerful tool for continuous integration and ensuring API reliability during development and deployment.

3.8.5 Connecting Server to Database (MongoDB)

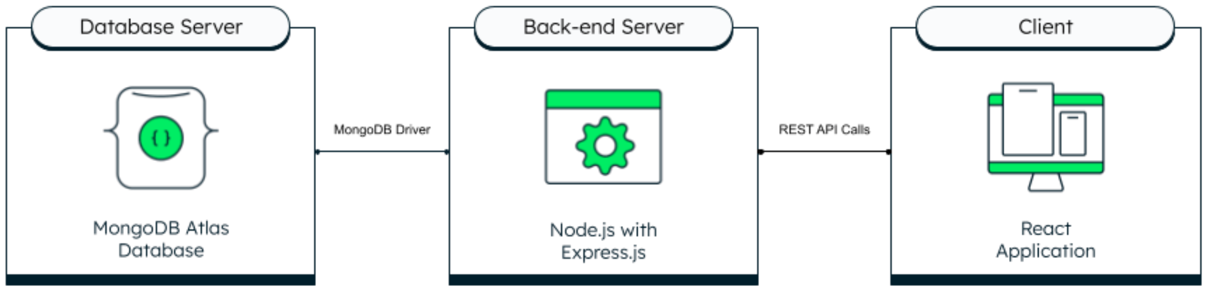


Fig 3.8.5 Connecting Server to Database

1. **MongoDB Driver:** Use a MongoDB driver like Mongoose to connect Express.js with MongoDB database.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});
```

2. **Define Schema and Create Model:**
   - **Schema**: A schema in MongoDB is a blueprint that defines the structure and data types for documents in a collection. It specifies the fields, types, and any validation rules for the data.
   - **Model**: A model is a constructor compiled from a Schema. It represents documents in a MongoDB collection and provides an interface for querying and manipulating the data.

Code

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const userSchema = new Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  age: { type: Number, min: 18 }
});

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

3.8.6 Storing JSON Data into the Database

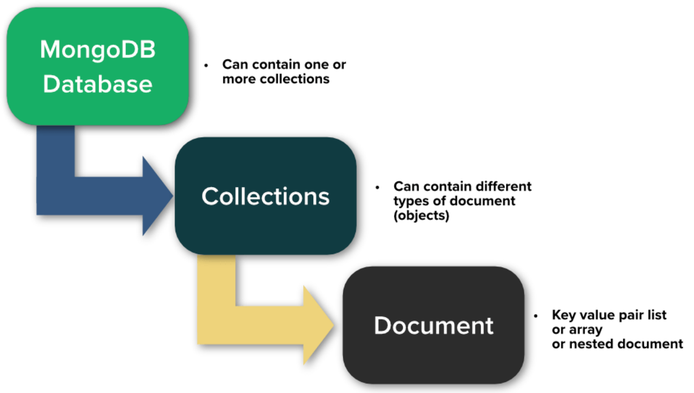

MongoDB Database
Can contain one or more collections
Collections
Can contain different types of document (objects)
Document
Key value pair list or array or nested document

Fig 3.8.1 MongoDB Hierarchy

1. JSON (JavaScript Object Notation):

- Data Format: JSON is a lightweight data interchange format.
- Syntax: Uses key-value pairs and data types like strings, numbers, arrays, and objects.
- Human-Readable: Easy for humans to read and write.
- Data Exchange: Commonly used for exchanging data between a server and a web application.
- Example: {"name": "John", "age": 30, "isEmployed": true}

2. Saving JSON Data Using Model.create Method:

Model.create() is a Mongoose method used to save multiple documents to the database in a single step. It accepts an array of objects (JSON data) and creates corresponding documents in the associated collection.

Code

```
Model.create(dataArray, (err, result) => {
  if (err) {
    console.error('Error saving data:', err);
  } else {
    console.log('Data saved successfully:', result);
  }
});
```

3.8.7 Reading Data from Database Using Model.find()

Reading Data from Database Using Model.find() (Short Theory):

Purpose: Model.find() is a Mongoose method used to retrieve multiple documents from a MongoDB collection based on specified criteria.

Query Conditions: You can pass an object to filter documents. For instance, { name: 'Alice' } retrieves documents where the 'name' field is 'Alice'.

Array of Documents: Model.find() returns an array of documents matching the query conditions. If no conditions are provided, it returns all documents from the collection**.**

```
const User = require('./userModel'); // Assuming 'userModel' is your Mongoose model

// Finding users where age is greater than or equal to 18
User.find({ age: { $gte: 18 } }, 'name age', (err, users) => {
  if (err) {
    console.error('Error fetching users:', err);
  } else {
    console.log('Users:', users);
  }
});
```

3.8.8 Adding Filtration & Searching Functionality with Query Props

*request.query is an object containing the parsed query parameters from the URL.
*When a client sends data through a URL query string (for example, ?key1=value1&key2=value2), Express automatically parses these parameters and makes them available in the request.query object.

3.8.9 Adding Advanced Search Functionality to Your REST API

Adding advanced search functionality to your REST API allows users to perform more complex queries using various criteria. You can achieve this by enhancing your route handling logic.

Code

```
const express = require('express');
const router = express.Router();
const Product = require('../models/Product'); // Assuming Product model is defined

// API endpoint to handle advanced product search requests
router.get('/products', (req, res) => {
  const { category, price_min, price_max, rating_min } = req.query;
  const query = {};

  if (category) {
    query.category = category;
  }

  if (price_min && price_max) {
    query.price = { $gte: parseInt(price_min), $lte: parseInt(price_max) };
  }

  if (rating_min) {
    query.rating = { $gte: parseInt(rating_min) };
  }

  Product.find(query, (err, products) => {
    if (err) {
      res.status(500).send('Error fetching products');
    } else {
      res.json(products);
    }
  });
});

module.exports = router;
```

3.8.10 Adding SORT functionality in Rest API

Sets the sort order
If a string is passed, it must be a space delimited list of path names. The sort order of each path is ascending unless the path name is prefixed with - which will be treated as descending.

```
query.sort('field -test');
```

3.8.11 Return Specific Document Fields using SELECT in Mongoose

The select() method to specify which fields you want to include or exclude from the query results. Here's how you can use the select() method to return specific document fields:
query.sort('field -test');

Code
```
Model.find()
  .select('-fieldToExclude')
  .exec((err, results) => {
    if (err) {
      console.error('Error:', err);
    } else {
      console.log('Results excluding specified field:', results);
    }
  });
```

3.8.12 Adding Pagination in REST API Using Node & Mongoose

Adding pagination to your API allows you to handle large datasets efficiently by dividing the results into manageable chunks. Clients can request specific portions of data, improving response times and reducing server load.

Query Parameters: Use query parameters like page and limit to allow clients to specify the page number and the number of items per page.

Code:
```
const express = require('express');
const router = express.Router();
const Product = require('../models/Product'); // Assuming Product model is defined

// API endpoint to handle paginated product requests
router.get('/products', async (req, res) => {
  const page = parseInt(req.query.page) || 1; // Current page number, default is 1
  const limit = parseInt(req.query.limit) || 10; // Number of items per page, default is 10

  try {
    const products = await Product.find()
      .skip((page - 1) * limit)
      .limit(limit)
      .exec();

    res.json(products);
  } catch (err) {
    res.status(500).send('Error fetching products');
  }
});

module.exports = router;
```

3.8.13 Host REST API on railway.app

- Create a Railway Project:

- Go to the Railway website.
- Sign up for an account if you haven't already.
- Create a new project and connect your repository where your Express.js project is stored (GitHub, GitLab, or Bitbucket).
- Set Environment Variables:

- In your Railway project, go to the Settings or Environment Variables section.
- Add any environment variables your application needs, such as database connection strings or API keys. These variables are crucial for your Express.js application to function correctly.
- Configure Railway Project:

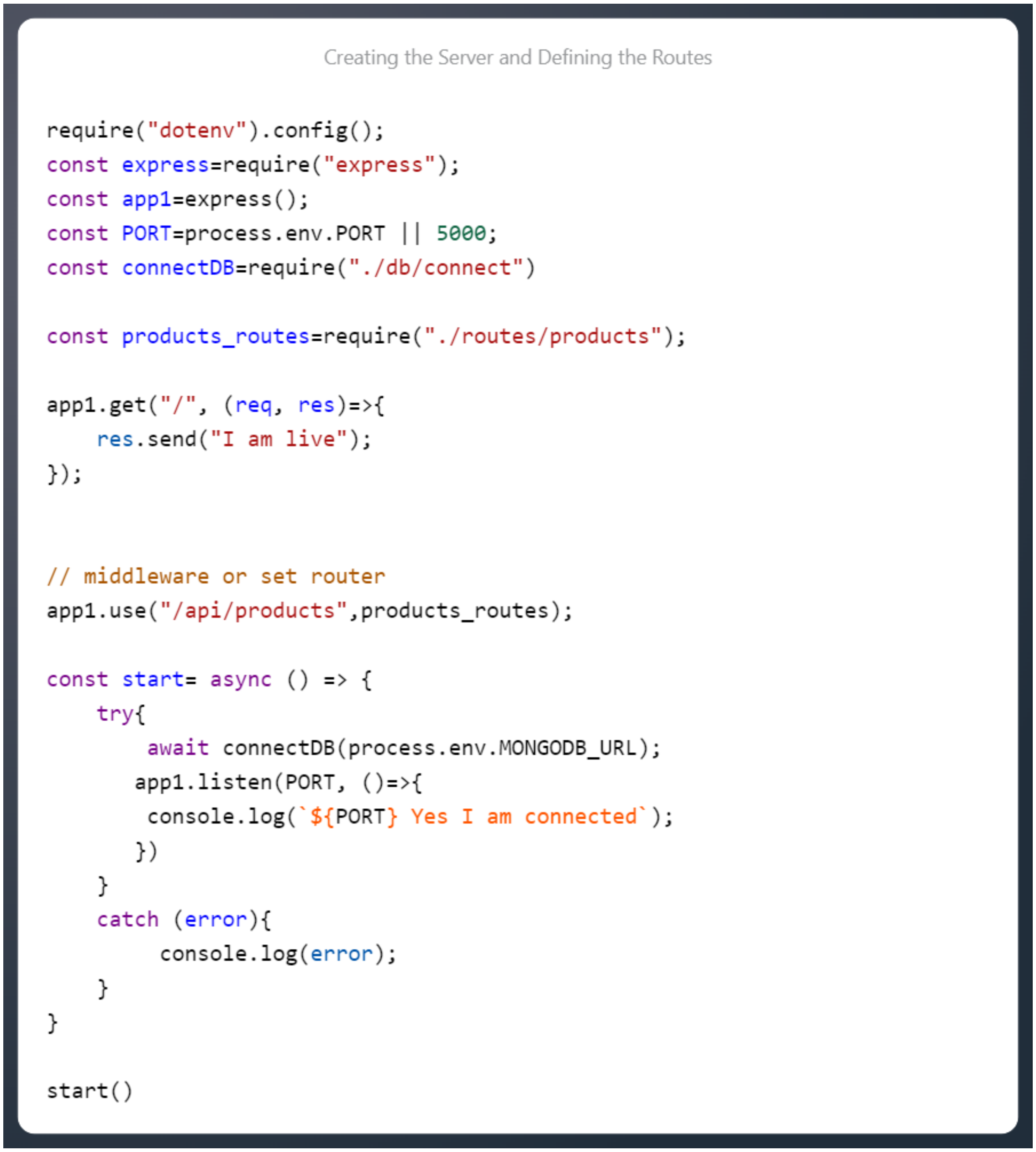
- Set up your deployment configuration in the Railway dashboard. Configure your main branch (usually main or master), the build command, and the start command for your Express.js application.
- Ensure that Railway knows how to build and run your application.
- Deploy Your Application:

- Once your project is configured, trigger the deployment. Railway will automatically build and deploy your Express.js application.
- Wait for the deployment to complete. You'll be able to see the progress in the Railway dashboard.
- Access Your API:

- Once the deployment is successful, Railway will provide you with a URL where your API is hosted.
- Use this URL to make requests to your API endpoints.

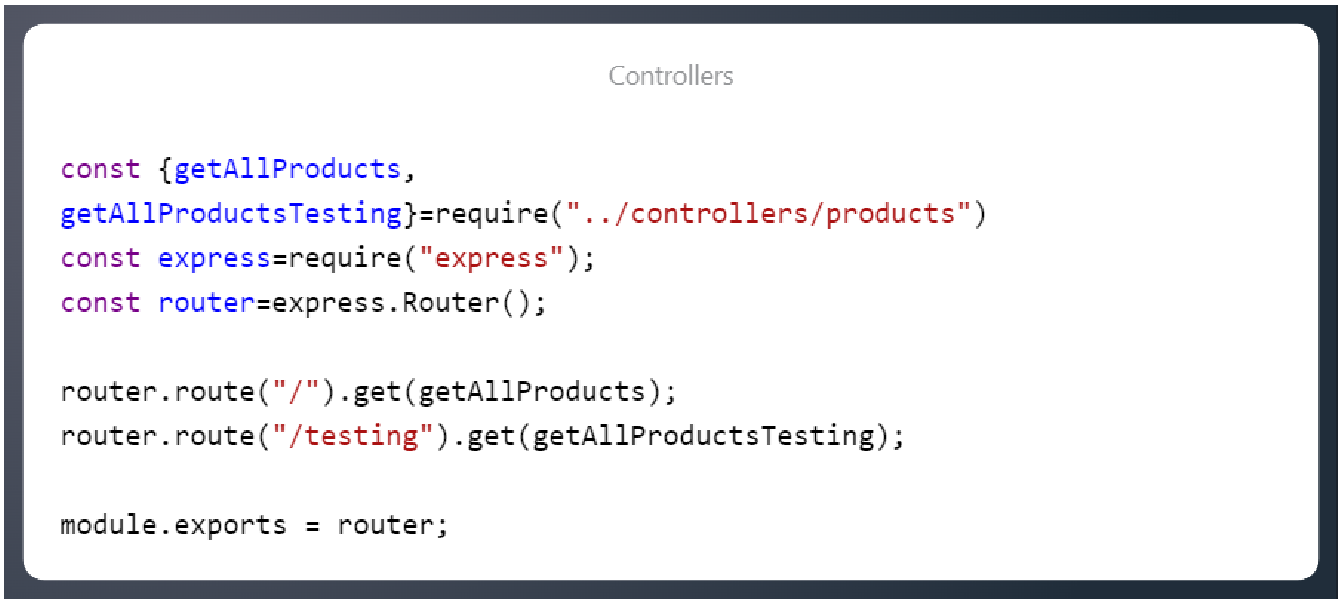
4.PRACTICAL EXECUTION

4.1 Creating the Server and Defining the Routes:


Creating the Server and Defining the Routes

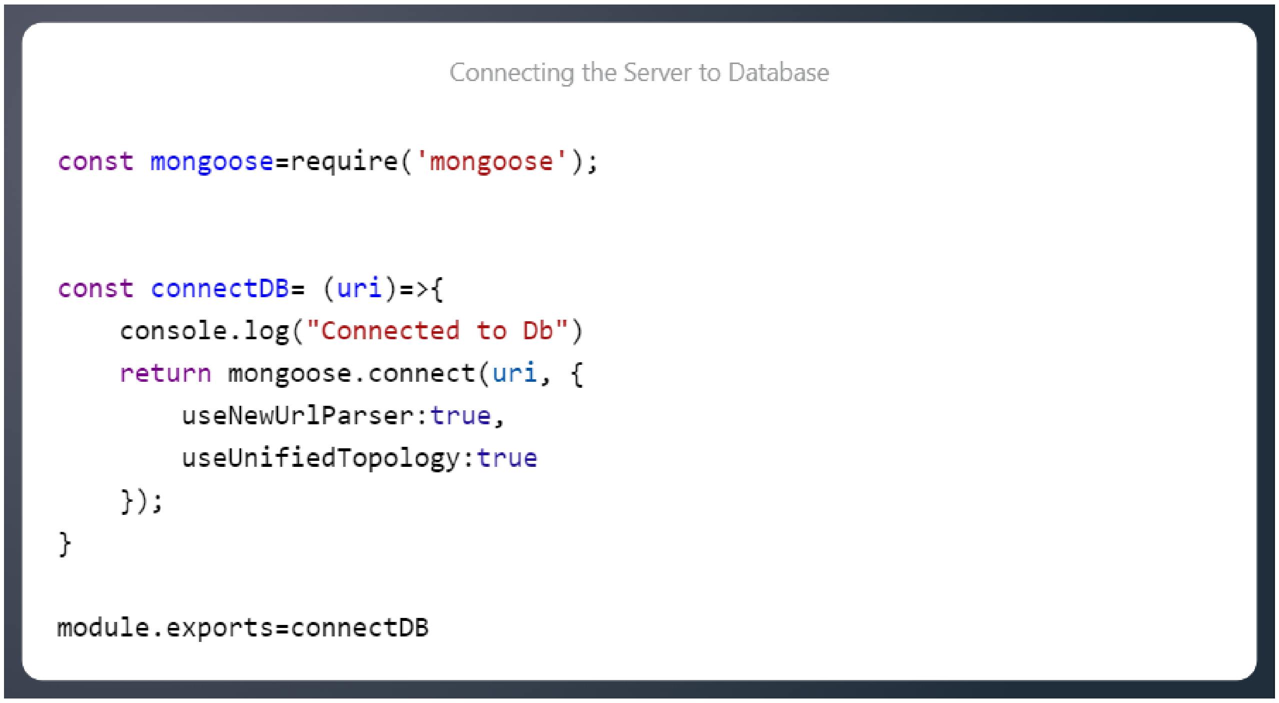
```
require("dotenv").config();
const express=require("express");
const app1=express();
const PORT=process.env.PORT || 5000;
const connectDB=require("./db/connect")

const products_routes=require("./routes/products");

app1.get("/", (req, res)=>{
    res.send("I am live");
});


// middleware or set router
app1.use("/api/products",products_routes);

const start= async () => {
    try{
        await connectDB(process.env.MONGODB_URL);
       app1.listen(PORT, ()=>{
        console.log(`${PORT} Yes I am connected`);
       })
    }
    catch (error){
         console.log(error);
    }
}

start()
```

4.2 Defining Controllers:



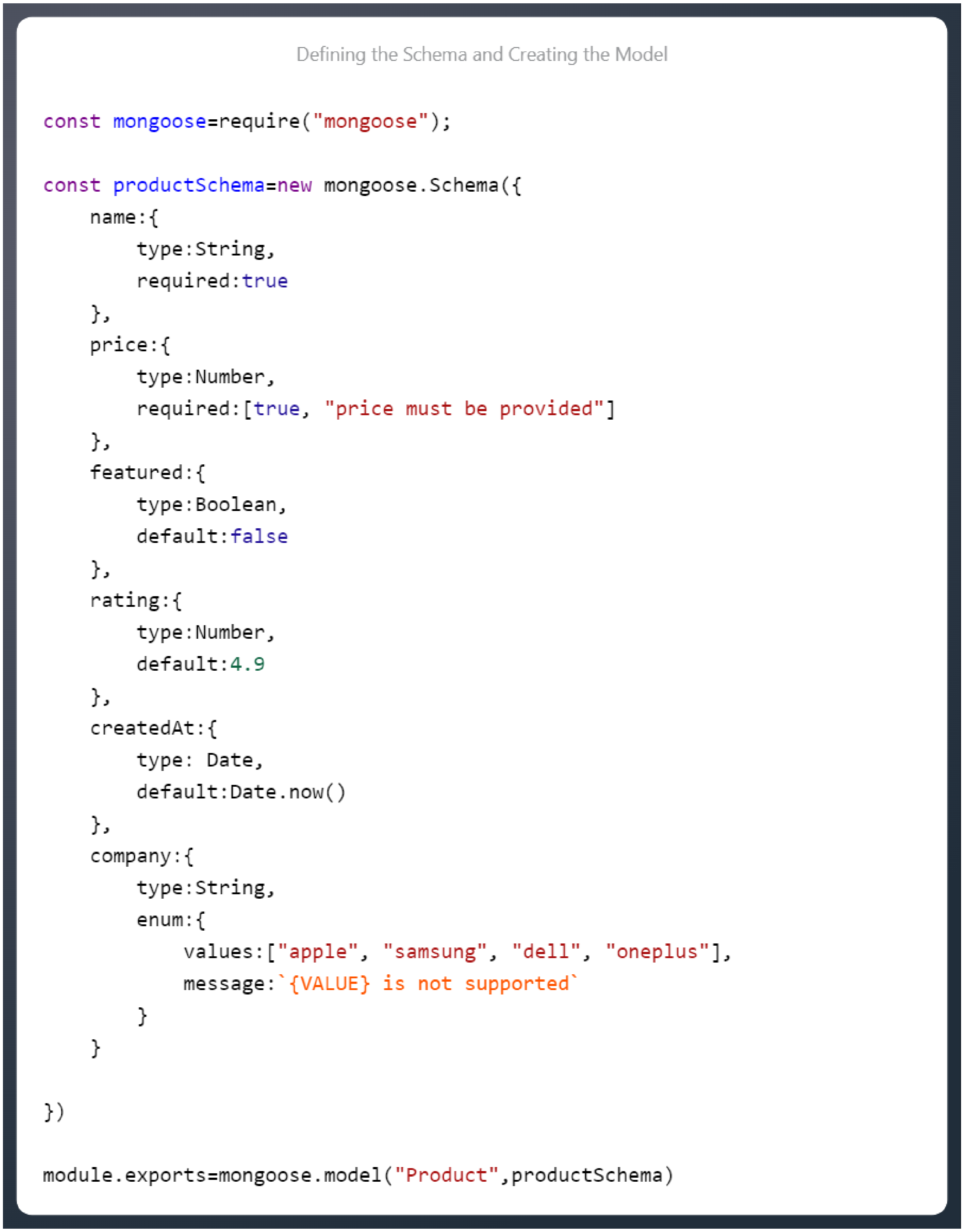
```
const {getAllProducts,
getAllProductsTesting}=require("../controllers/products")
const express=require("express");
const router=express.Router();

router.route("/").get(getAllProducts);
router.route("/testing").get(getAllProductsTesting);

module.exports = router;
```

4.3 Connecting Server to the Database :



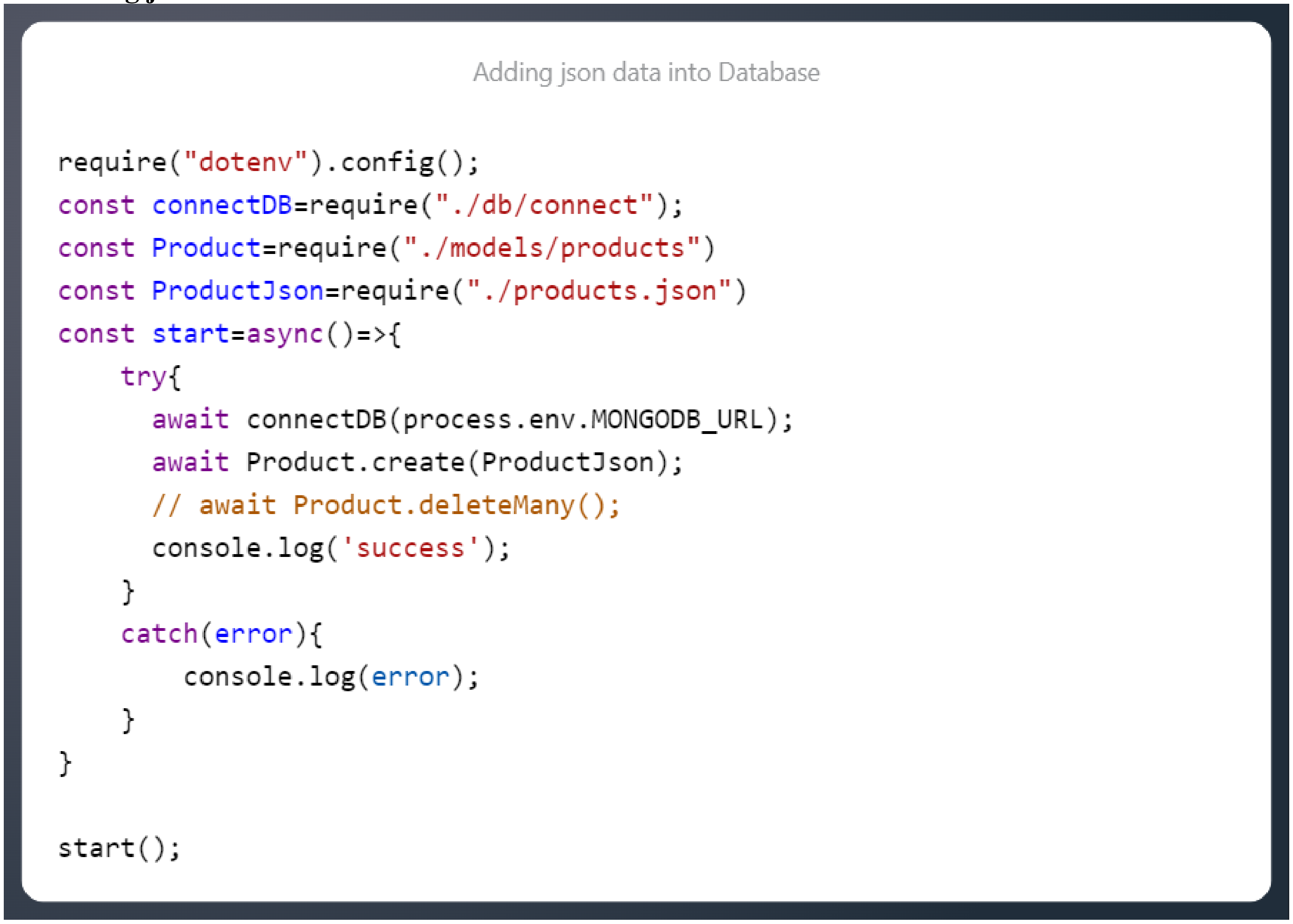
```
const mongoose=require('mongoose');


const connectDB= (uri)=>{
    console.log("Connected to Db")
    return mongoose.connect(uri, {
        useNewUrlParser:true,
        useUnifiedTopology:true
    });
}

module.exports=connectDB
```

4.4 Defining the Schema and Creating the Model :


Defining the Schema and Creating the Model

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

const productSchema=new mongoose.Schema({
    name:{
        type:String,
        required:true
    },
    price:{
        type:Number,
        required:[true, "price must be provided"]
    },
    featured:{
        type:Boolean,
        default:false
    },
    rating:{
        type:Number,
        default:4.9
    },
    createdAt:{
        type: Date,
        default:Date.now()
    },
    company:{
        type:String,
        enum:{
            values:["apple", "samsung", "dell", "oneplus"],
            message:`{VALUE} is not supported`
        }
    }

})

module.exports=mongoose.model("Product",productSchema)
```

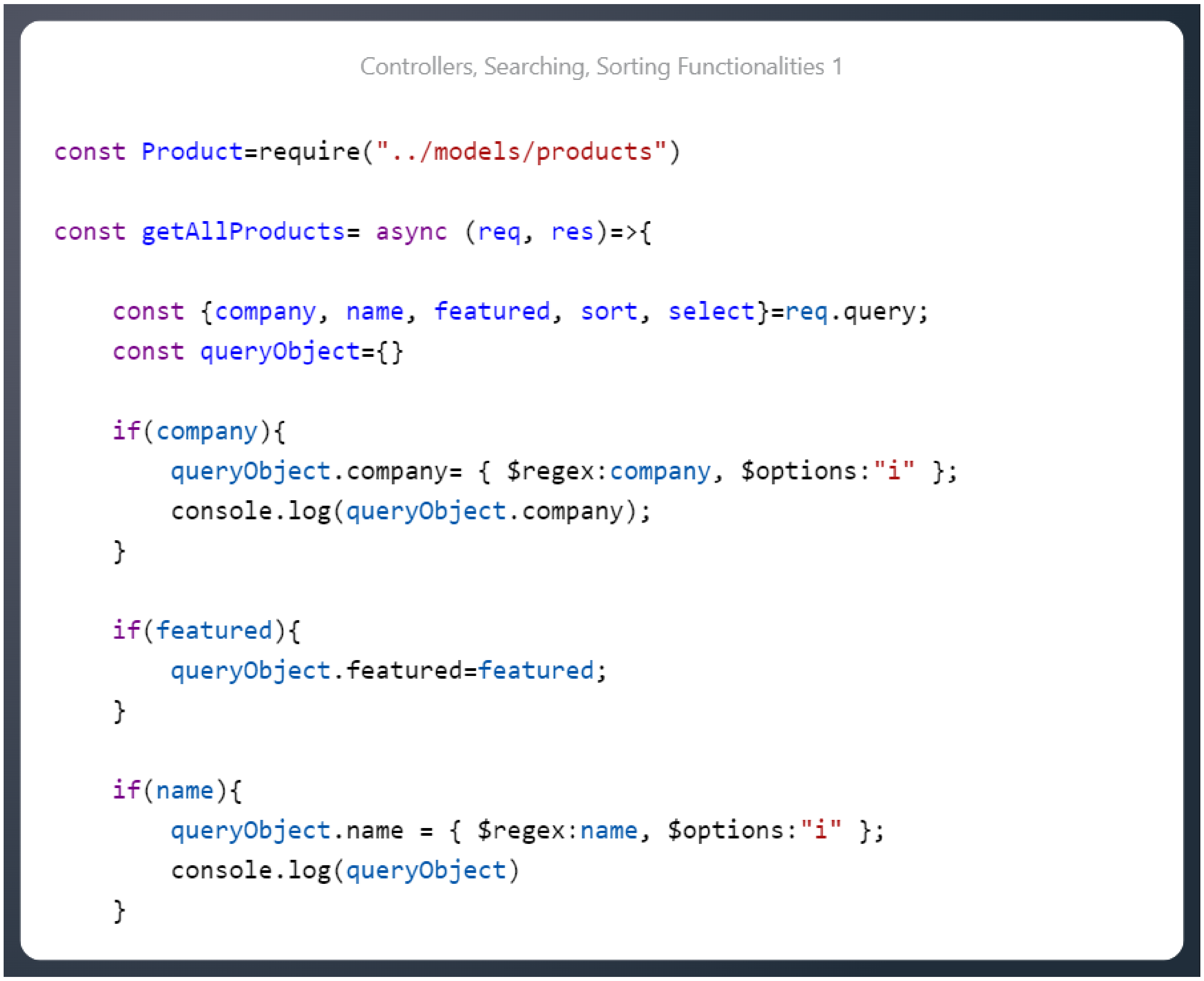
4.5 Adding json Data into Database :



Adding json data into Database

```
require("dotenv").config();
const connectDB=require("./db/connect");
const Product=require("./models/products")
const ProductJson=require("./products.json")
const start=async()=>{
    try{
      await connectDB(process.env.MONGODB_URL);
      await Product.create(ProductJson);
      // await Product.deleteMany();
      console.log('success');
    }
    catch(error){
        console.log(error);
    }
}

start();
```

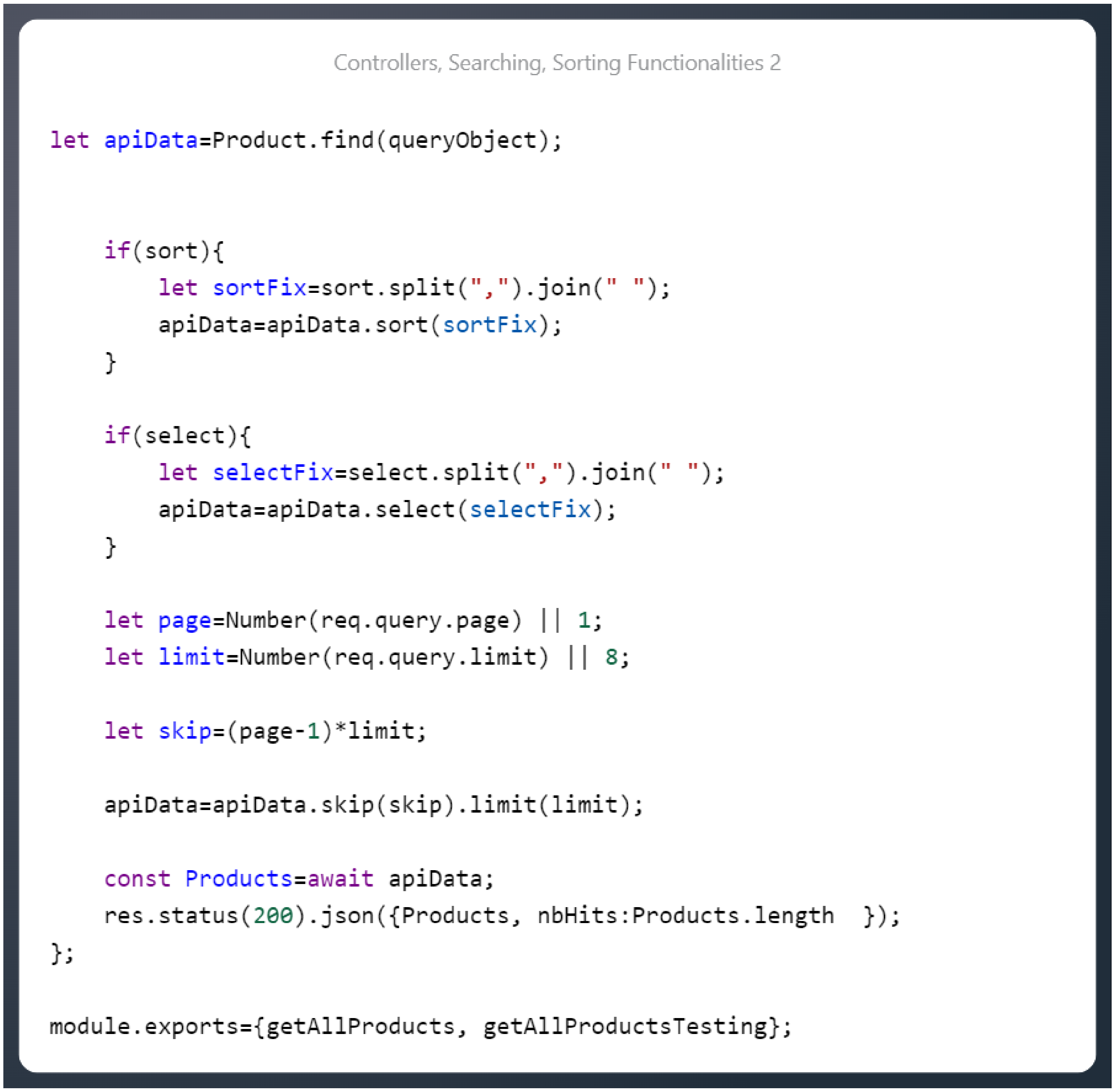
4.6.1 Controllers, Searching, Filtering Functionalities :



Controllers, Searching, Sorting Functionalities 1

```
const Product=require("../models/products")

const getAllProducts= async (req, res)=>{

    const {company, name, featured, sort, select}=req.query;
    const queryObject={}

    if(company){
        queryObject.company= { $regex:company, $options:"i" };
        console.log(queryObject.company);
    }

    if(featured){
        queryObject.featured=featured;
    }

    if(name){
        queryObject.name = { $regex:name, $options:"i" };
        console.log(queryObject)
    }
```

4.6.2 Sorting Functionalities :


Controllers, Searching, Sorting Functionalities 2

```
let apiData=Product.find(queryObject);


    if(sort){
        let sortFix=sort.split(",").join(" ");
        apiData=apiData.sort(sortFix);
    }

    if(select){
        let selectFix=select.split(",").join(" ");
        apiData=apiData.select(selectFix);
    }

    let page=Number(req.query.page) || 1;
    let limit=Number(req.query.limit) || 8;

    let skip=(page-1)*limit;

    apiData=apiData.skip(skip).limit(limit);

    const Products=await apiData;
    res.status(200).json({Products, nbHits:Products.length  });
};

module.exports={getAllProducts, getAllProductsTesting};
```

4.7 Summary:

Server Creation:

- **Node.js & Express Setup:** Started by setting up a Node.js server using the Express.js framework. Express simplifies the process of building robust APIs.

Route and Controller Setup:

- **Defined Routes:** Established specific routes (or endpoints) within the API. Each route corresponds to a specific functionality, making the API organized and easy to navigate.
- **Controllers:** Developed controller functions corresponding to each route. Controllers handle the logic for processing requests, making the code modular and maintainable. For example, there could be controllers for adding products, retrieving product details, and handling user authentication.

Database Connection and Data Handling:

- **Database Integration:** Integrated the server with MongoDB, a popular NoSQL database, using Mongoose, an Object Data Modeling (ODM) library for MongoDB and Node.js. This connection enables the server to store and retrieve data in a structured manner.
- **Adding Data:** Created controller functions to add JSON data to the MongoDB database. These functions process incoming data and store it in the appropriate database collections.

Data Operations:

- **Finding Data:** Implemented controller functions for finding specific data in the database. Users can request specific information, and the API responds with the relevant data, facilitating targeted searches.
- **Filtering, Sorting, and Pagination:** Enhanced data manipulation capabilities by adding functionalities like filtering (based on criteria such as category or price range), sorting (arranging results alphabetically, numerically, etc.), and pagination (splitting large datasets into manageable pages). These features improve user experience and optimize API performance, especially when dealing with extensive datasets.
- **Pagination:** Implemented pagination logic to divide large sets of data into smaller, more manageable chunks. Users can navigate through these chunks using page numbers, ensuring a smoother browsing experience.

API Deployment:

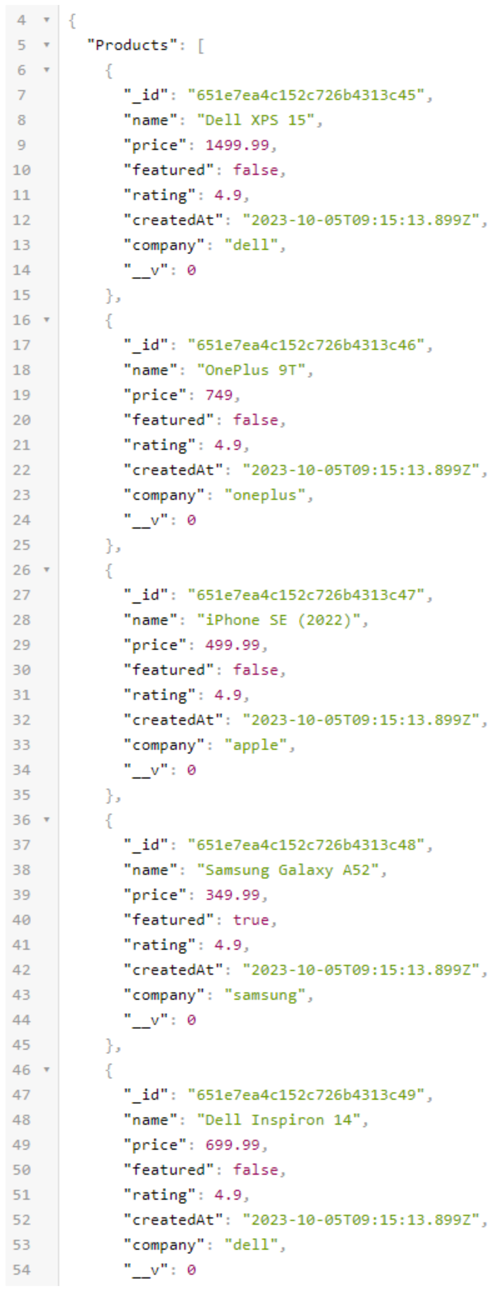
- **Hosting on Railway:** Deployed the API on Railway.app, a cloud hosting platform. Railway.app manages the deployment process, ensuring the API is accessible via a unique URL. This step enables the API to go live, making it publicly available for users to interact with.

5.RESULT & DISCUSSION

5.1 Link of Deployed Restful API

https://srestapi-production.up.railway.app/api/products

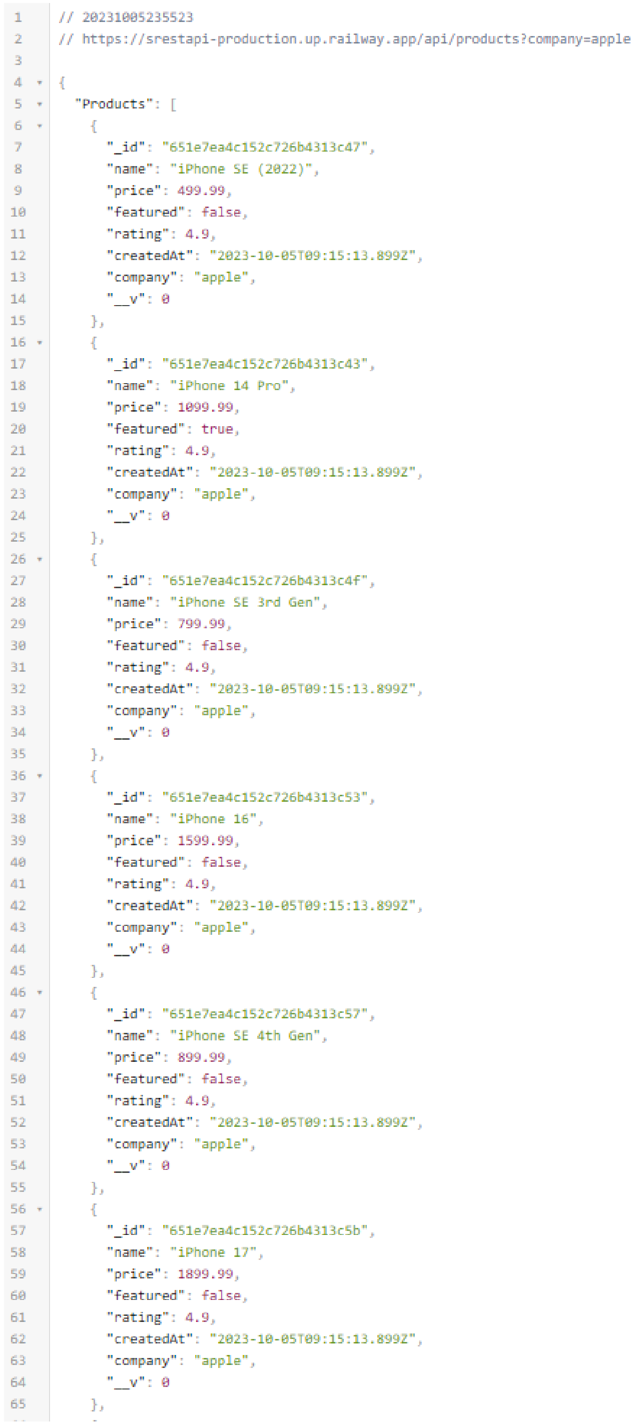
5.2.1 Result for url https://srestapi-production.up.railway.app/api/products



```
{
  "Products": [
    {
      "_id": "651e7ea4c152c726b4313c45",
      "name": "Dell XPS 15",
      "price": 1499.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "dell",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c46",
      "name": "OnePlus 9T",
      "price": 749,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "oneplus",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c47",
      "name": "iPhone SE (2022)",
      "price": 499.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "apple",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c48",
      "name": "Samsung Galaxy A52",
      "price": 349.99,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "samsung",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c49",
      "name": "Dell Inspiron 14",
      "price": 699.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "dell",
      "__v": 0
```

5.2.2 Result for url

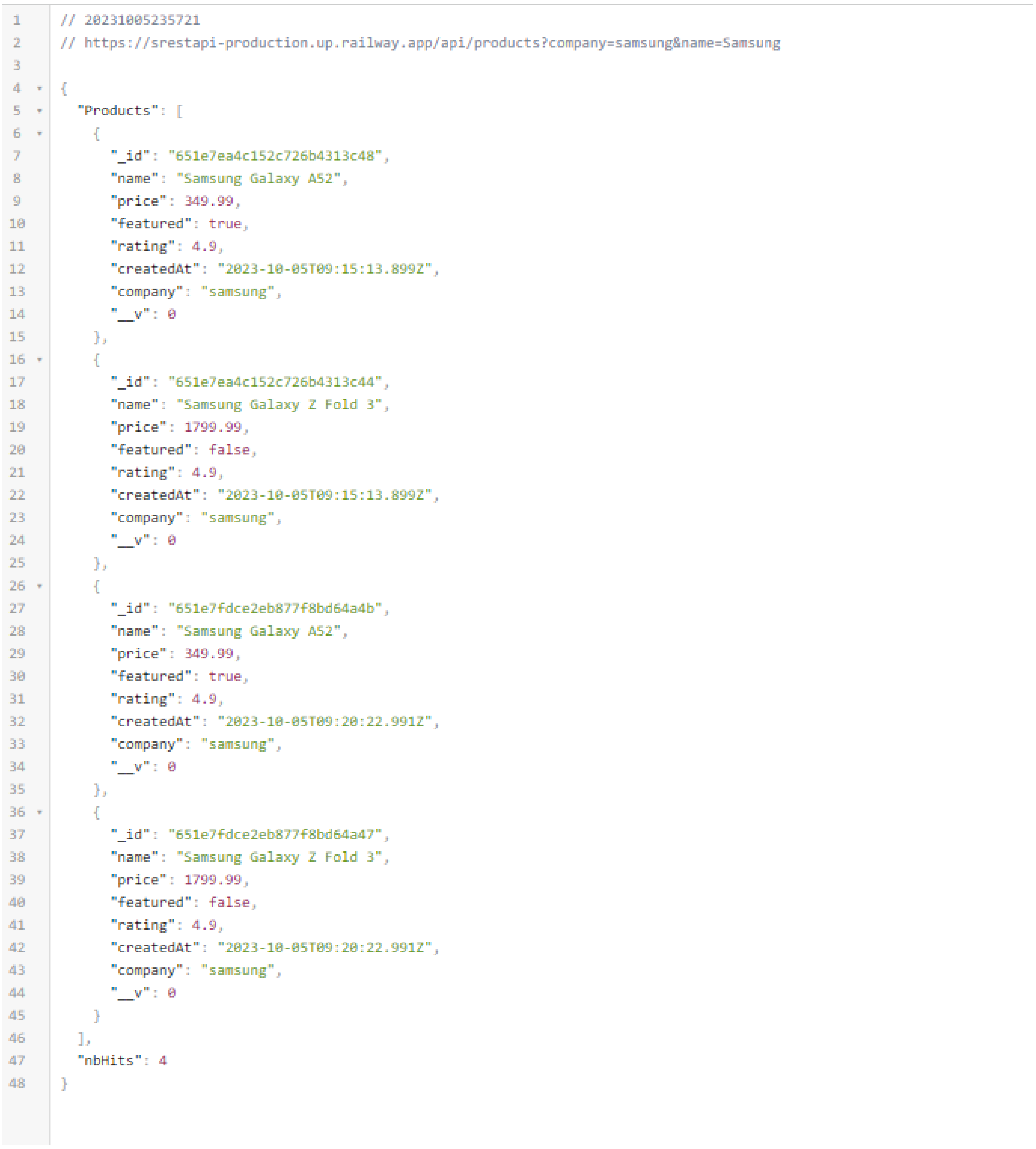
https://srestapi-production.up.railway.app/api/products?company=apple



```
// 20231005235523
// https://srestapi-production.up.railway.app/api/products?company=apple

{
  "Products": [
    {
      "_id": "651e7ea4c152c726b4313c47",
      "name": "iPhone SE (2022)",
      "price": 499.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "apple",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c43",
      "name": "iPhone 14 Pro",
      "price": 1099.99,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "apple",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c4f",
      "name": "iPhone SE 3rd Gen",
      "price": 799.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "apple",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c53",
      "name": "iPhone 16",
      "price": 1599.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "apple",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c57",
      "name": "iPhone SE 4th Gen",
      "price": 899.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "apple",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c5b",
      "name": "iPhone 17",
      "price": 1899.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "apple",
      "__v": 0
    },
```

5.2.3 Result for url

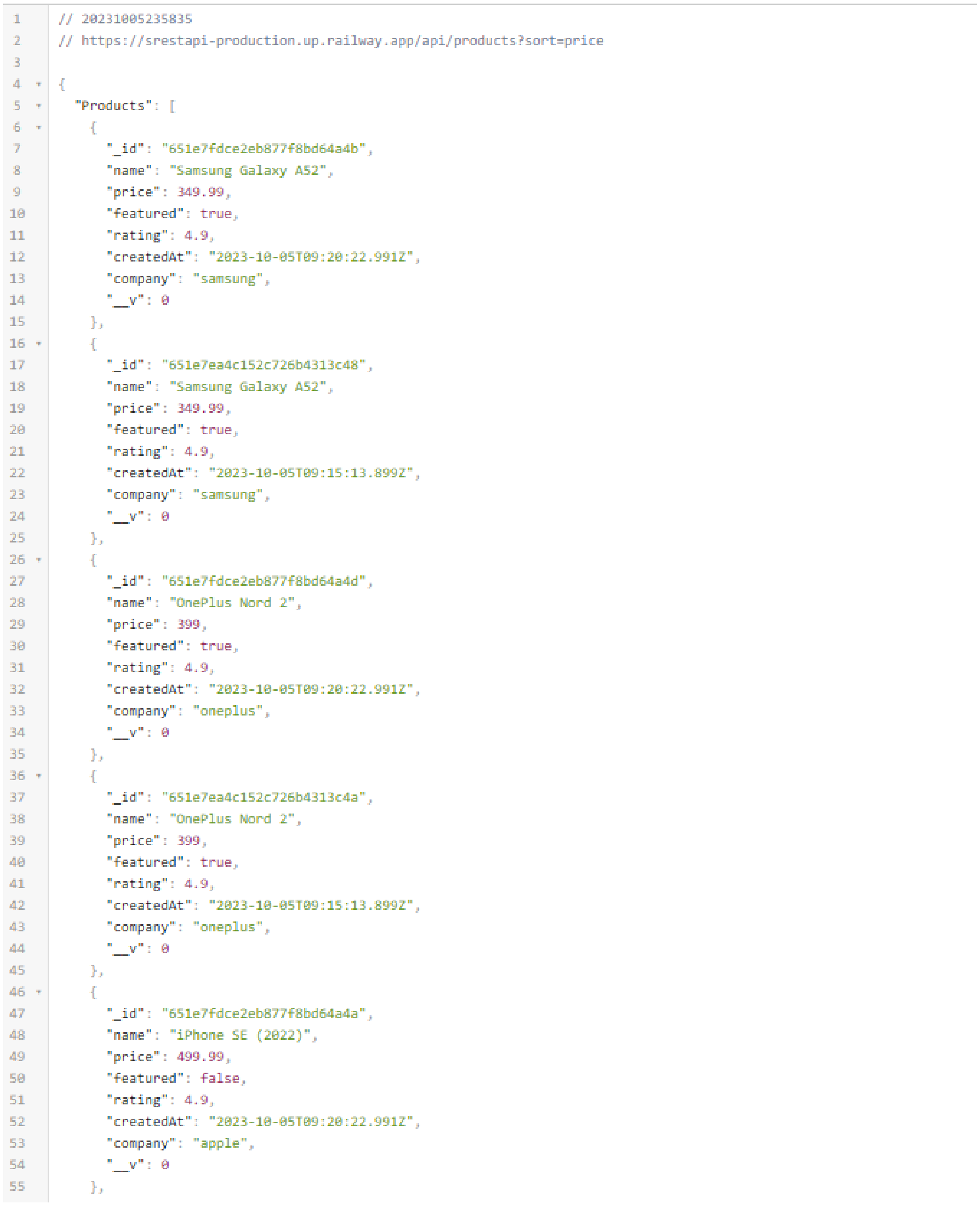
https://srestapi-production.up.railway.app/api/products?company=samsung&name=Samsung



```
// 20231005235721
// https://srestapi-production.up.railway.app/api/products?company=samsung&name=Samsung

{
  "Products": [
    {
      "_id": "651e7ea4c152c726b4313c48",
      "name": "Samsung Galaxy A52",
      "price": 349.99,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "samsung",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c44",
      "name": "Samsung Galaxy Z Fold 3",
      "price": 1799.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "samsung",
      "__v": 0
    },
    {
      "_id": "651e7fdce2eb877f8bd64a4b",
      "name": "Samsung Galaxy A52",
      "price": 349.99,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:20:22.991Z",
      "company": "samsung",
      "__v": 0
    },
    {
      "_id": "651e7fdce2eb877f8bd64a47",
      "name": "Samsung Galaxy Z Fold 3",
      "price": 1799.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:20:22.991Z",
      "company": "samsung",
      "__v": 0
    }
  ],
  "nbHits": 4
}
```

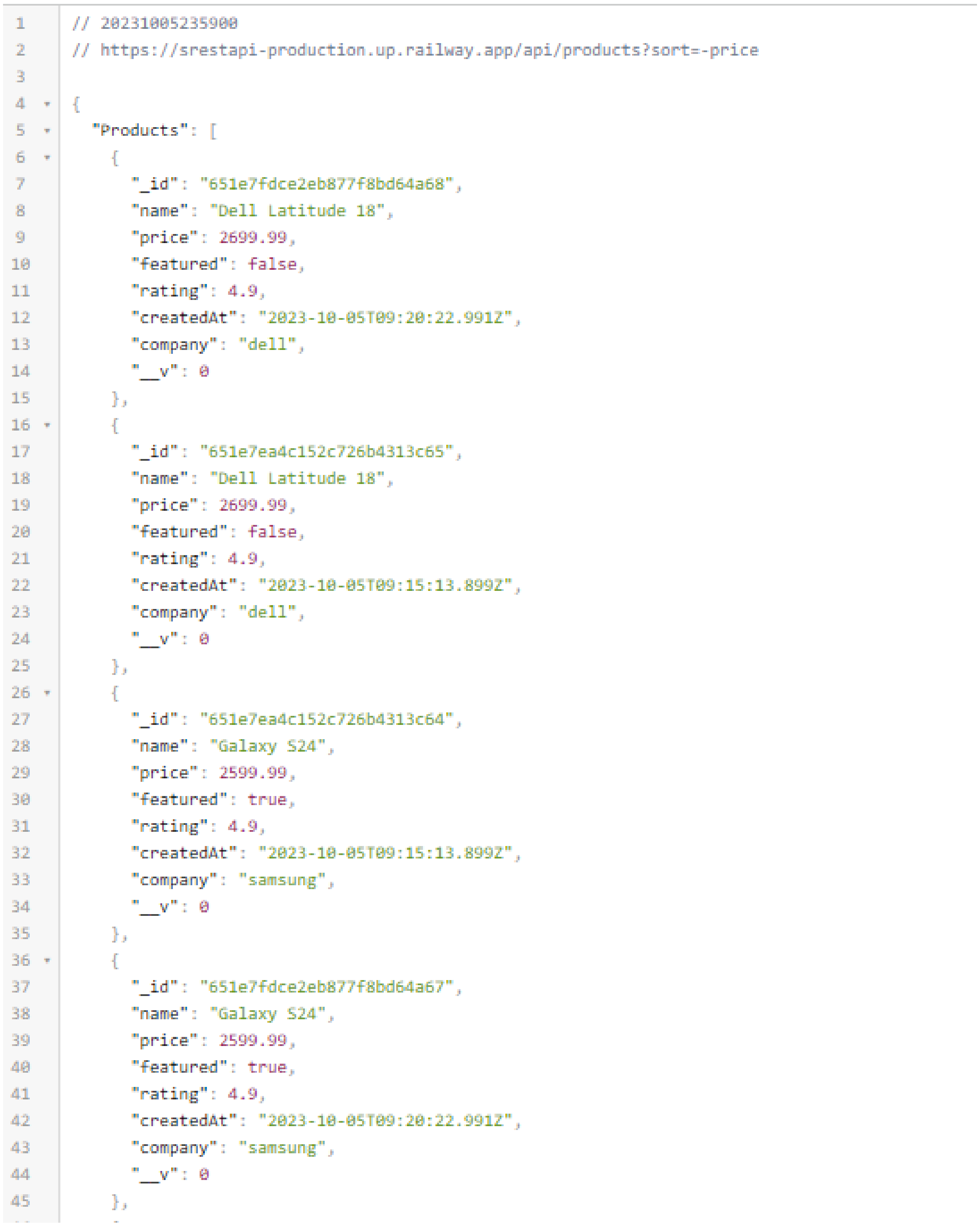
5.2.4 Result for url

https://srestapi-production.up.railway.app/api/products?sort=price



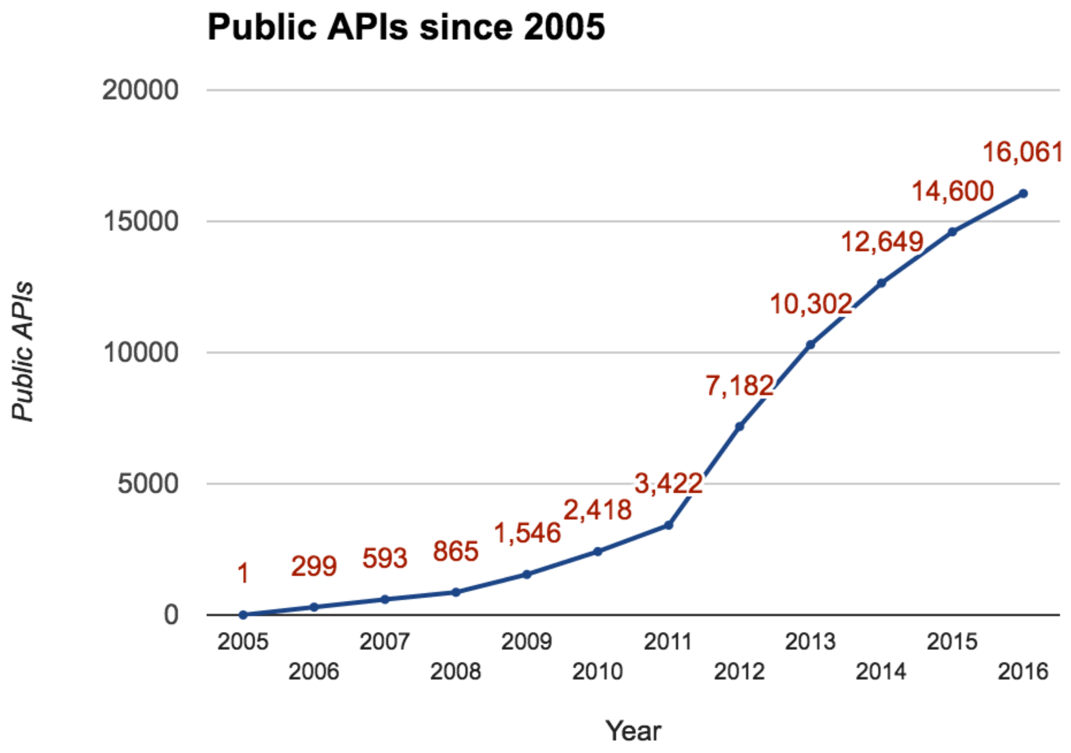
```
// 20231005235835
// https://srestapi-production.up.railway.app/api/products?sort=price

{
  "Products": [
    {
      "_id": "651e7fdce2eb877f8bd64a4b",
      "name": "Samsung Galaxy A52",
      "price": 349.99,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:20:22.991Z",
      "company": "samsung",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c48",
      "name": "Samsung Galaxy A52",
      "price": 349.99,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "samsung",
      "__v": 0
    },
    {
      "_id": "651e7fdce2eb877f8bd64a4d",
      "name": "OnePlus Nord 2",
      "price": 399,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:20:22.991Z",
      "company": "oneplus",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c4a",
      "name": "OnePlus Nord 2",
      "price": 399,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "oneplus",
      "__v": 0
    },
    {
      "_id": "651e7fdce2eb877f8bd64a4a",
      "name": "iPhone SE (2022)",
      "price": 499.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:20:22.991Z",
      "company": "apple",
      "__v": 0
    },
```

5.2.5 Result for url https://srestapi-production.up.railway.app/api/products?sort=-price

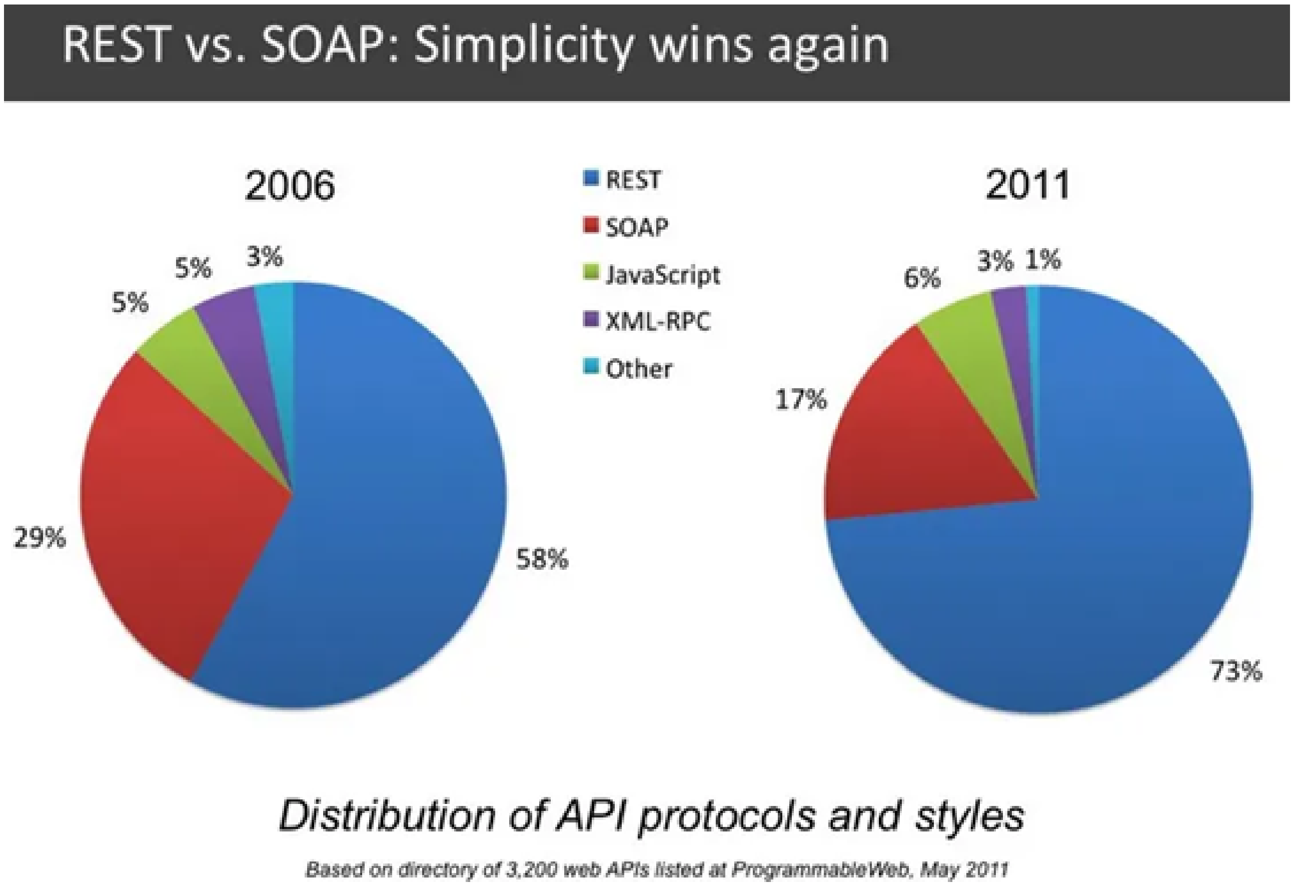


```
// 20231005235900
// https://srestapi-production.up.railway.app/api/products?sort=-price

{
  "Products": [
    {
      "_id": "651e7fdce2eb877f8bd64a68",
      "name": "Dell Latitude 18",
      "price": 2699.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:20:22.991Z",
      "company": "dell",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c65",
      "name": "Dell Latitude 18",
      "price": 2699.99,
      "featured": false,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "dell",
      "__v": 0
    },
    {
      "_id": "651e7ea4c152c726b4313c64",
      "name": "Galaxy S24",
      "price": 2599.99,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:15:13.899Z",
      "company": "samsung",
      "__v": 0
    },
    {
      "_id": "651e7fdce2eb877f8bd64a67",
      "name": "Galaxy S24",
      "price": 2599.99,
      "featured": true,
      "rating": 4.9,
      "createdAt": "2023-10-05T09:20:22.991Z",
      "company": "samsung",
      "__v": 0
    },
```

5.3 PUBLIC RestAPI



5.4 RestAPI Vs SOAP



6.CONCLUSION

In the domain of digital commerce and online interactions, the development of a robust REST API stands as the cornerstone of modern web applications. Throughout this project, we embarked on a transformative journey, unraveling the intricacies of API development using Node.js, Express.js, and MongoDB. This endeavor was not merely an exploration of technologies; it was a profound immersion into the art and science of creating seamless user experiences. Starting from the inception of a solid server foundation, we meticulously designed routes and controllers, orchestrating a symphony of data flow and functionality. The integration with MongoDB, facilitated by the Mongoose ODM, empowered our API with the ability to store, retrieve, and manipulate data efficiently. The emphasis on data operations – from precise filtering to dynamic pagination – ensured that our API not only met but exceeded user expectations in terms of speed, accuracy, and user-friendliness. The deployment phase marked a pivotal moment, elevating our API from a local development environment to a global digital asset. Railway.app, with its seamless hosting capabilities, provided the perfect launchpad for our creation, making it accessible to users worldwide.

7. FUTURE SCOPE

- Optimized Server Performance: Focus on server optimization techniques to ensure lightning-fast response times and efficient resource utilization.

- Advanced Controllers: Develop specialized controllers for complex business logic, enabling intricate operations and personalized user experiences.

- Database Enhancements: Implement database indexing and caching for faster data retrieval, enhancing overall system responsiveness.

- Data Enrichment: Integrate external APIs to enrich existing data, providing users with comprehensive product information and boosting engagement.

- Real-Time Updates: Implement real-time data synchronization for instant product updates, ensuring users always access the most current information.

- Machine Learning Integration: Explore machine learning algorithms for predictive analytics, enabling dynamic product recommendations and personalized experiences.

- Mobile App Integration: Create native mobile applications consuming the API, optimizing user interactions on smartphones and tablets.

- Enhanced Security Measures: Implement robust security protocols, including encryption and threat detection, safeguarding user data and ensuring system integrity.

7.REFERENCES

[1] Imam Ahmad, Emi Suwarni, Rohmat Indra Borman, Asmawati, Farli Rossi, Yessi Jusman, “Implementation of RESTful API Web Services Architecture in Takeaway Application Development” in Proc. IEEE 4th Inf. Technol, December 2021, pp. 1895–1899

[2] Sujan Y M, Dr. Shashidhara, and Dr. Rohini Nagapadma, " Framework of REST APIs" in Proc. International Research Journal of Engineering and Technology (IRJET), : 07 Issue: 06 | June 2020 .

[3] Carlos Rodriguez, Marcos Baez, Florian Daniel, “A Large-Scale Analysis of Compliance with Principles and Best Practices” in Proc. Research Gate, December 2021, pp. 1895–1899