**Sahil Gujarathi**

**Employee Code:- 10707**

**JavaScript, NodeJs, MySQL Document**

**GitHub Link:- https://github.com/sahilwinjit/LD**

**JavaScript**

**JavaScript Basics**

* **Variables: Declare variables using `let`, `const`, and `var`.**
* **Variable Rules: Rules for naming and using variables.**
* **DataTypes: Different types of data in JavaScript (String, Number, Boolean, etc.).**
* **Operators: Arithmetic, unary, assignment, comparison, logical operators.**
* **Conditional Statements: `if`, `else if`, `else`, and `switch` statements.**
* **Loops: `for`, `while`, `do...while` loops.**

**DOM Manipulation**

* **Selecting Elements: Using methods like `getElementById`, `querySelector`.**
* **Modifying Elements: Changing the content or attributes of elements.**
* **Creating and Removing Elements: Using `createElement`, `removeChild`.**
* **Manipulating Styles: Changing CSS styles of elements.**
* **Traversing the DOM: Navigating through parent, child, and sibling elements.**

**Event Handling**

* **Event Types: Different types of events (click, submit, etc.).**
* **Event Listeners: Using `addEventListener` to handle events.**

**Callbacks and Promises**

* **Callbacks: Functions passed as arguments to other functions.**
* **Promises: Objects representing the eventual completion or failure of an asynchronous operation.**
* **Async/Await: Syntax for writing asynchronous code.**

**Fetch API**

* **Basic Fetch Request: Using `fetch` to make HTTP requests.**
* **Request Options: Configuring requests with headers, methods, etc.**
* **Response Handling: Processing responses from the server.**
* **Error Handling: Managing errors in fetch requests.**

**Building Applications**

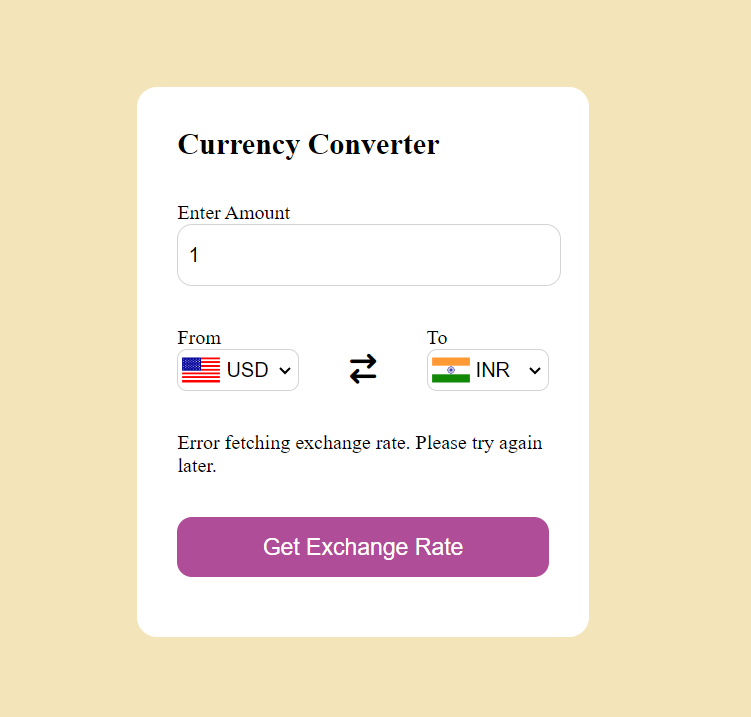
* **Tic-Tac-Toe Game: Using JavaScript concepts to build a game.**



* **Rock-Paper-Scissors Game: Implementing game logic in JavaScript.**



* **Currency Converter: Using fetch API to build a currency converter application.**



**NodeJS**

**Node.js Basics**

* **Introduction to Node.js: Understanding the basics of Node.js.**
* **Modules: Importing and exporting modules in Node.js.**
* **File System: Reading and writing files using the `fs` module.**
* **NPM: Managing packages using Node Package Manager.**

**Express.js Basics**

* **Routing: Defining routes in an Express.js application.**
* **Routing in Express.js involves managing how the application's endpoints (URLs) react to client requests. This process determines the application's response to a client request made to a specific endpoint, identified by a path, and an HTTP request method such as GET, POST, PUT, DELETE, etc.**
* **Basic Route Definition:**

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

**const app = express();**

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

**res.send('Hello World!');**

**});**

**app.listen(3000, () => {**

**console.log('Server is running on port 3000');**

**});**

* **Route Parameters: Route parameters enable the capture of values specified by their position within the URL.**

**app.get('/users/:userId', (req, res) => {**

**res.send(`User ID: ${req.params.userId}`);**

**});**

* **Route Groups: Organize routes that share a common base path.**

**const userRouter = express.Router();**

**userRouter.get('/', (req, res) => {**

**res.send('User Home');**

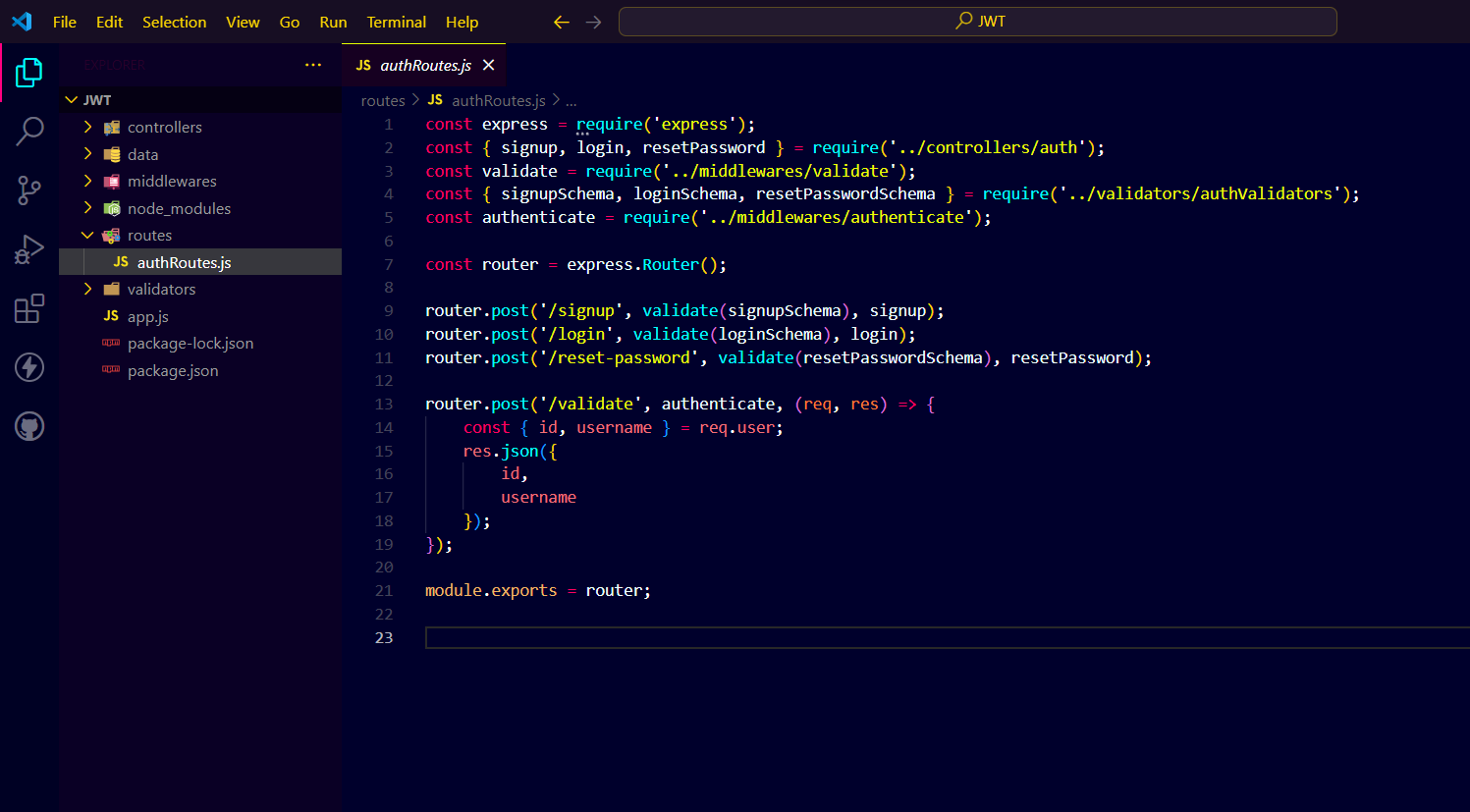
**});**

**userRouter.get('/:userId', (req, res) => {**

**res.send(`User ID: ${req.params.userId}`);**

**});**

**app.use('/users', userRouter);**

****

* **Middleware: Understanding and using middleware.**
* **Middleware functions are functions that can access the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. These functions can carry out various tasks:**
  + **Execute any code.**
  + **Modify the request and response objects.**
  + **Terminate the request-response cycle.**
  + **Invoke the next middleware function in the sequence.**
* **Application-Level Middleware:**

**app.use((req, res, next) => {**

**console.log('Time:', Date.now());**

**next();**

**});**

* **Router-Level Middleware:**

**const userRouter = express.Router();**

**userRouter.use((req, res, next) => {**

**console.log('User Router Time:', Date.now());**

**next();**

**});**

**app.use('/users', userRouter);**

* **Error-Handling Middleware:**

**app.use((err, req, res, next) => {**

**console.error(err.stack);**

**res.status(500).send('Something broke!');**

**});**

* **Built-In Middleware (e.g., for serving static files):**

**app.use(express.static('public'));**

* **Third-Party Middleware (e.g., body-parser for parsing request bodies):**

**const bodyParser = require('body-parser');**

**app.use(bodyParser.json());**

* **HTTP Methods: Handling GET, POST, PUT, DELETE requests.**
* **Express.js offers methods to manage various HTTP requests such as GET, POST, PUT, DELETE, etc. These methods align with the CRUD operations.**
* **GET: Retrieve data from the server.**

**app.get('/items', (req, res) => {**

**res.send('Get all items');**

**});**

* **POST: Send data to the server to create a new resource.**

**app.post('/items', (req, res) => {**

**res.send('Add an item');**

**});**

* **PUT: Update an existing resource on the server.**

**app.put('/items/:itemId', (req, res) => {**

**res.send(`Update item with ID: ${req.params.itemId}`);**

**});**

* **DELETE: Remove a resource from the server.**

**app.delete('/items/:itemId', (req, res) => {**

**res.send(`Delete item with ID: ${req.params.itemId}`);**

**});**

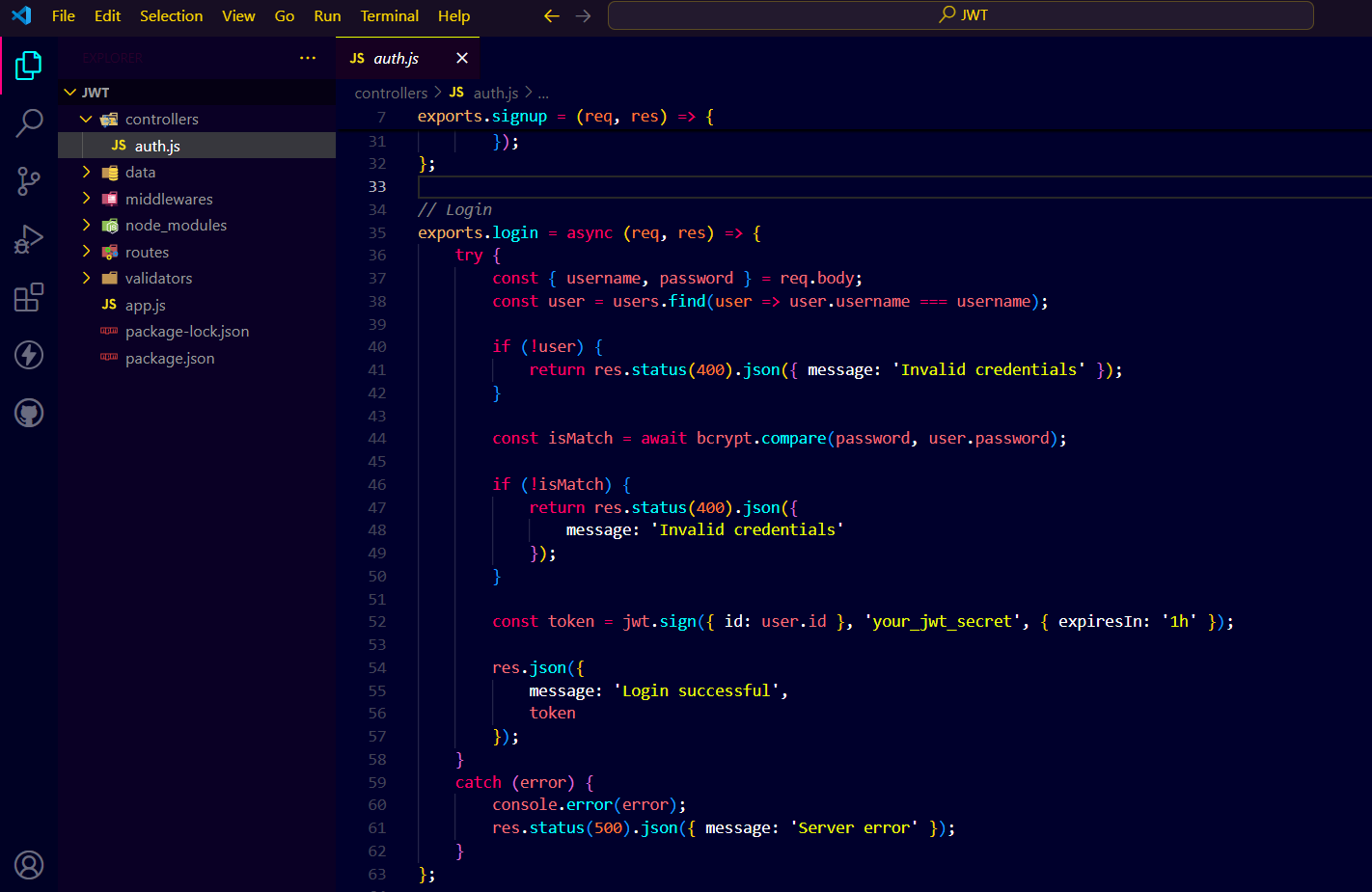
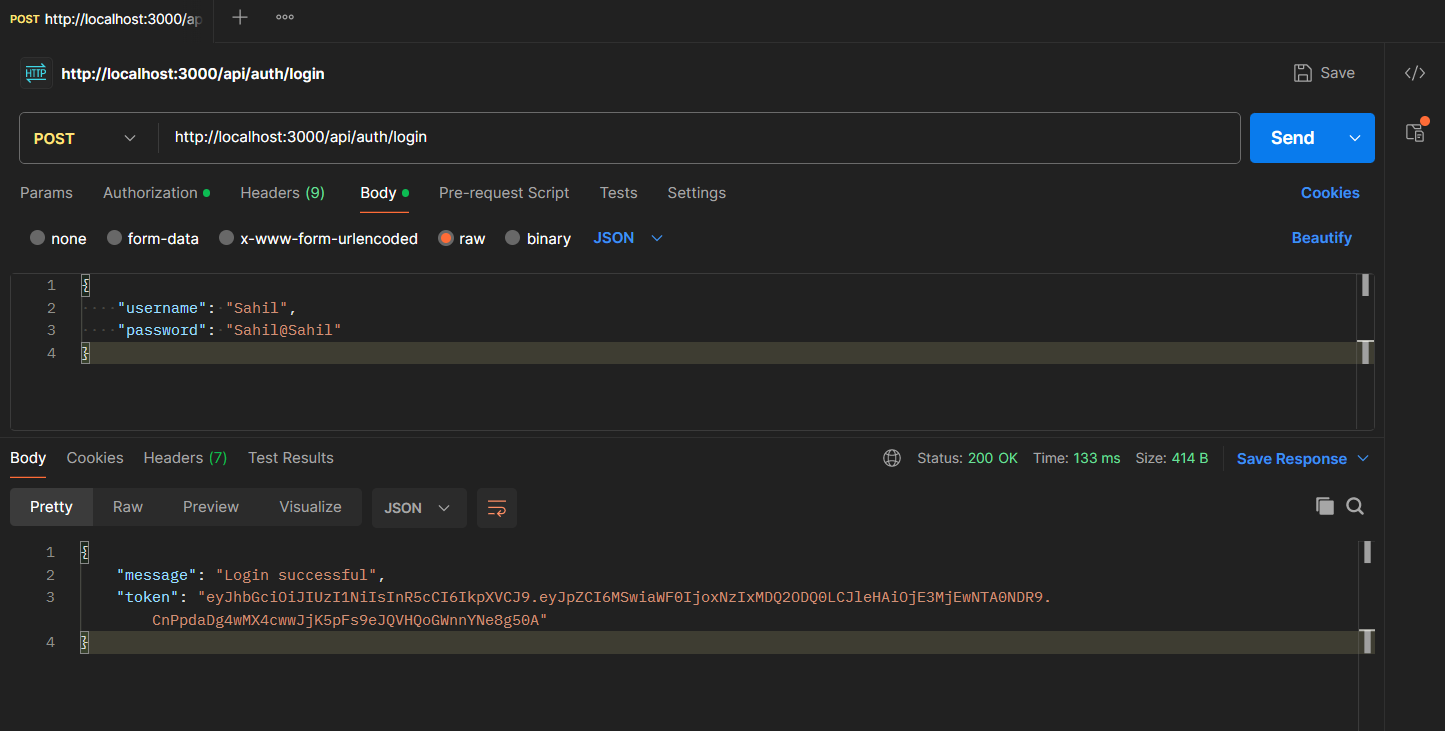
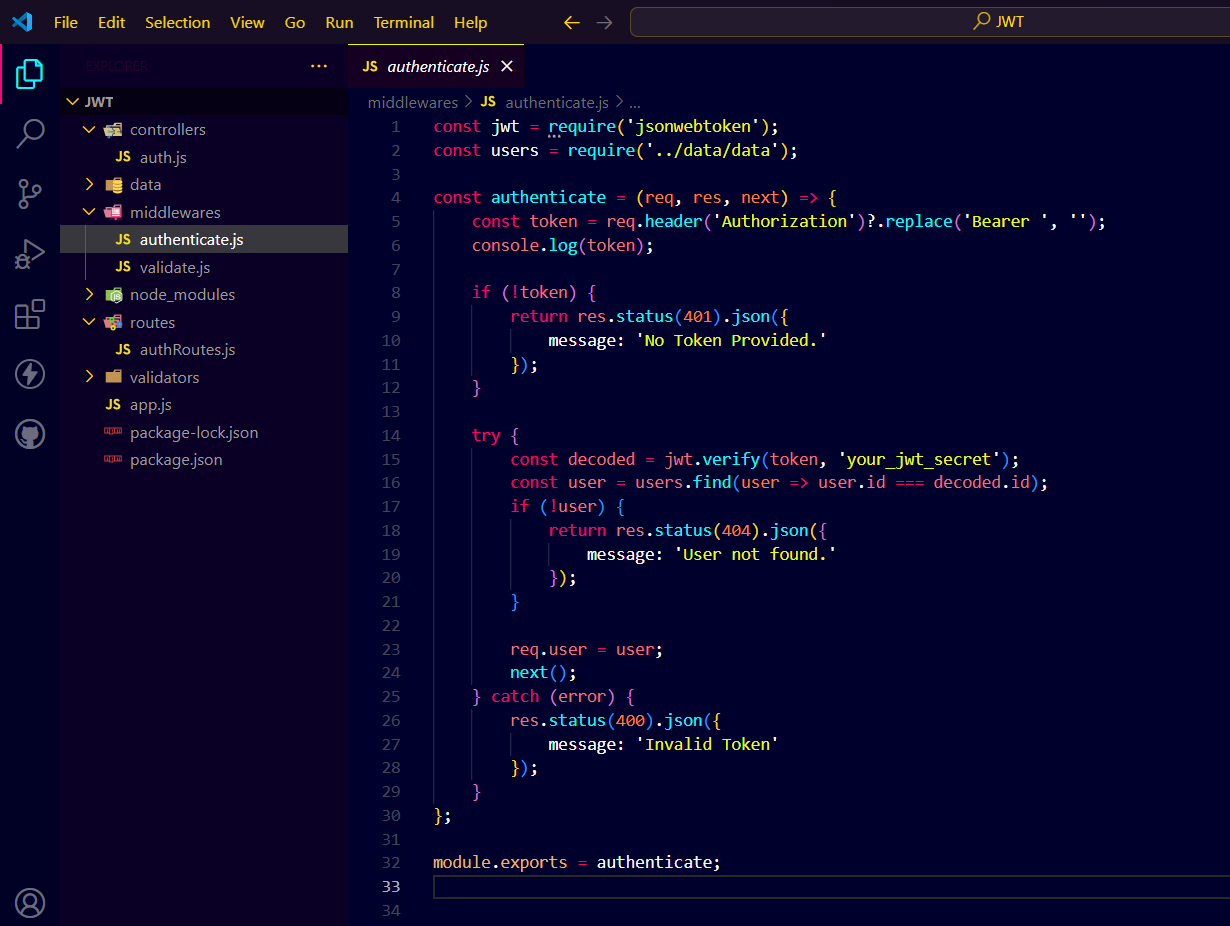
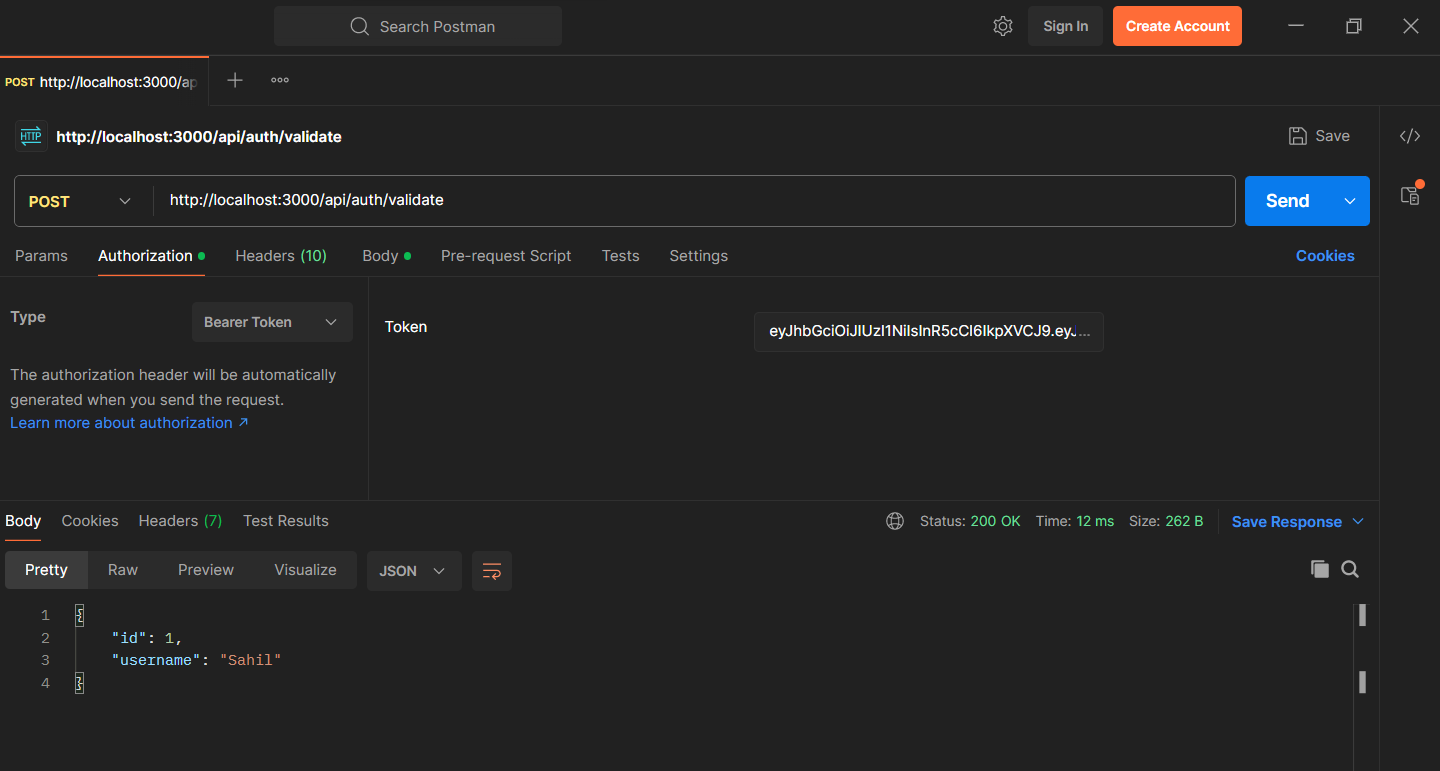
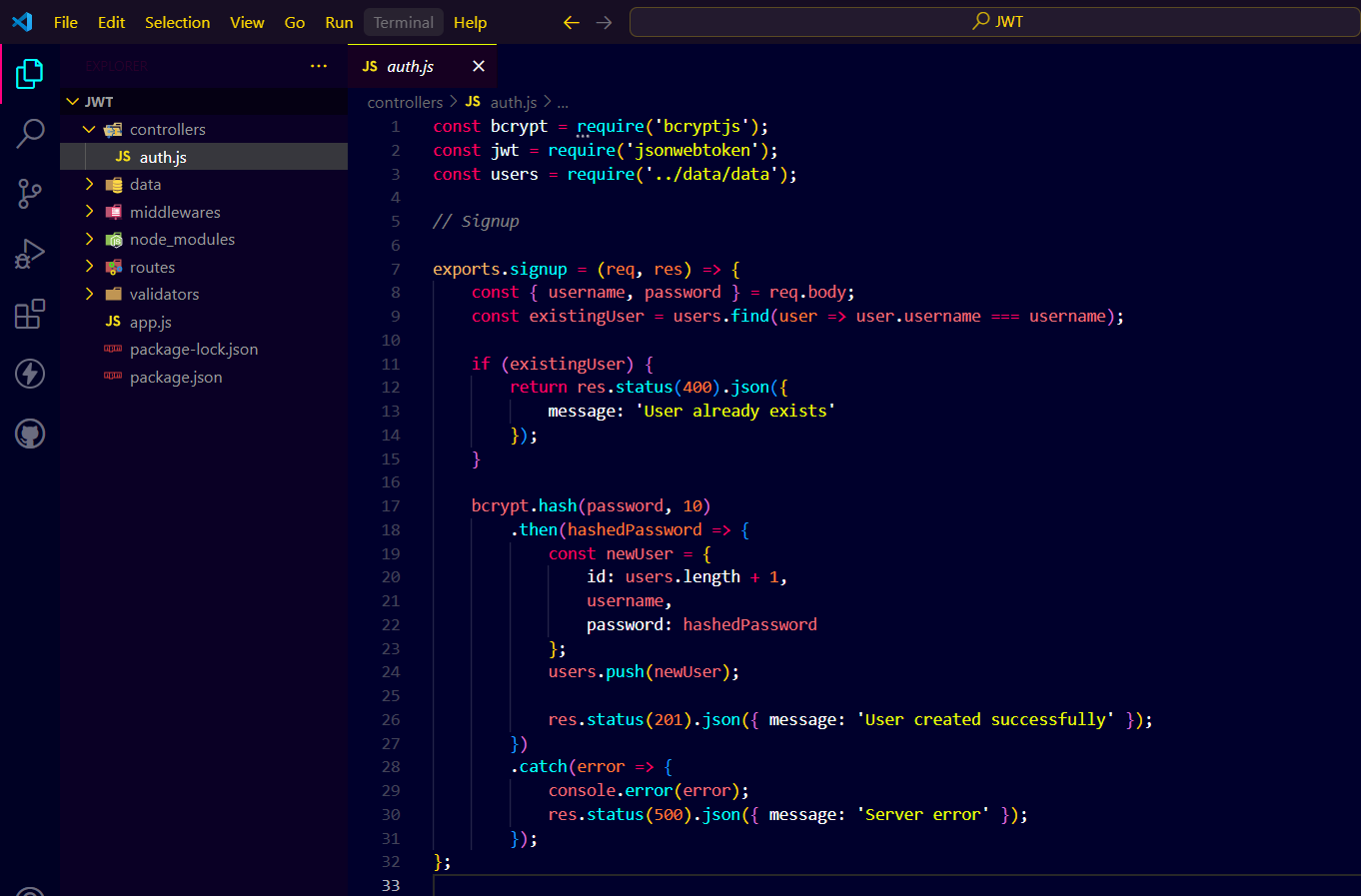
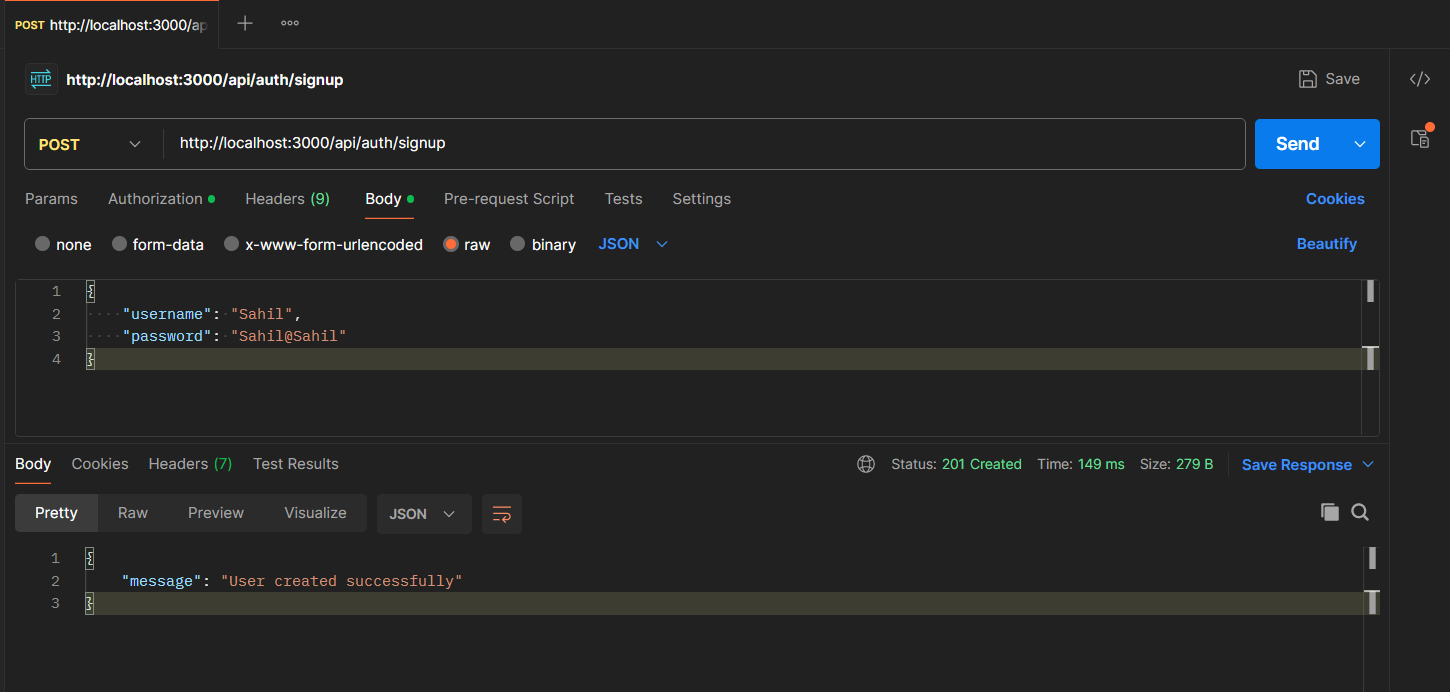
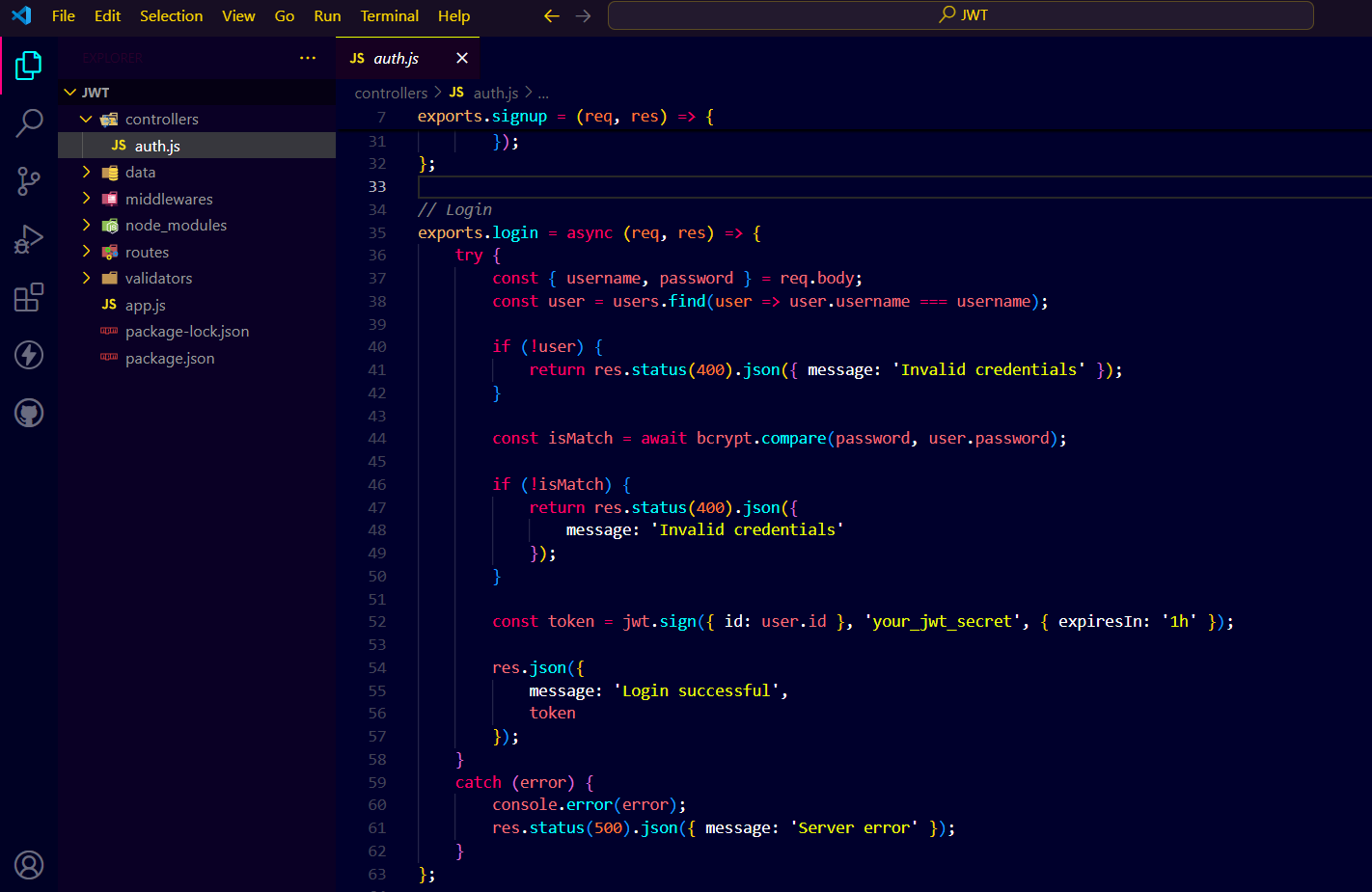
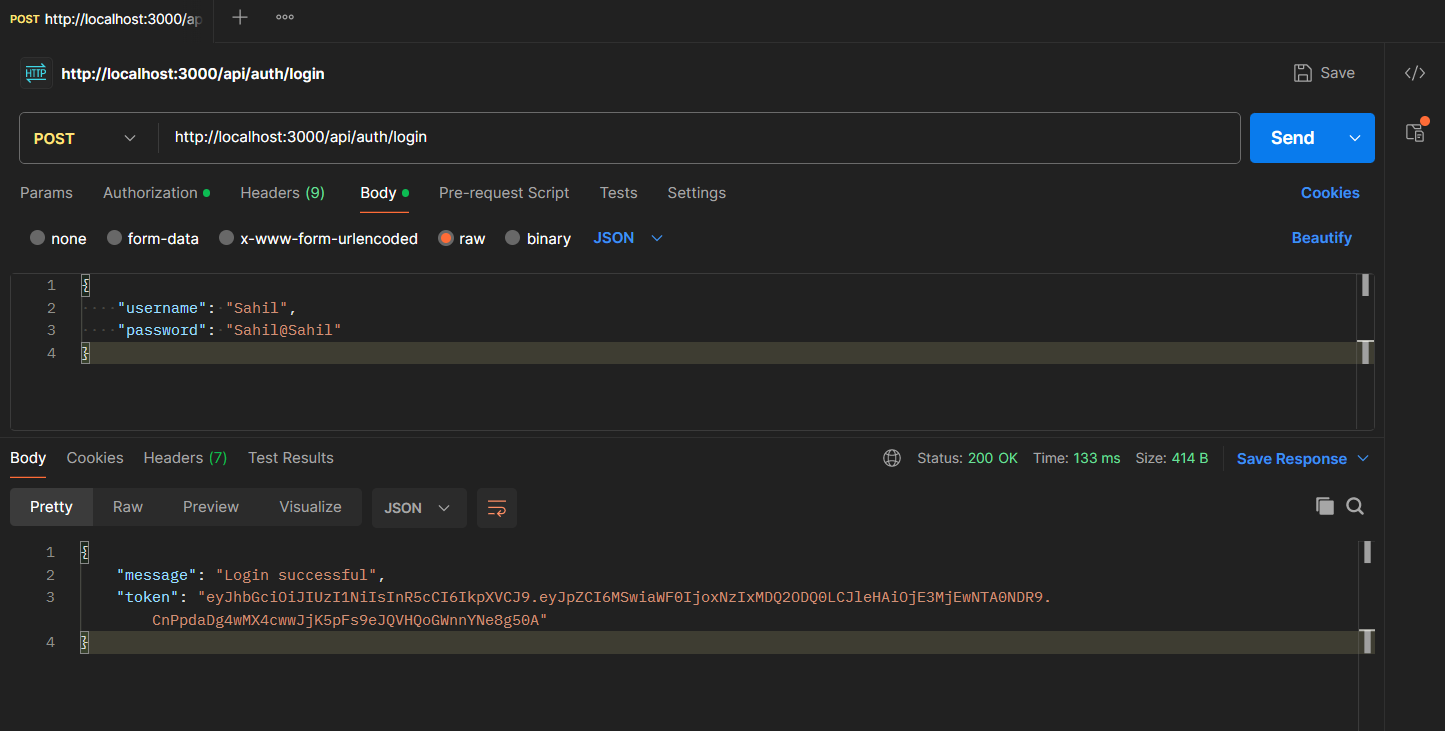
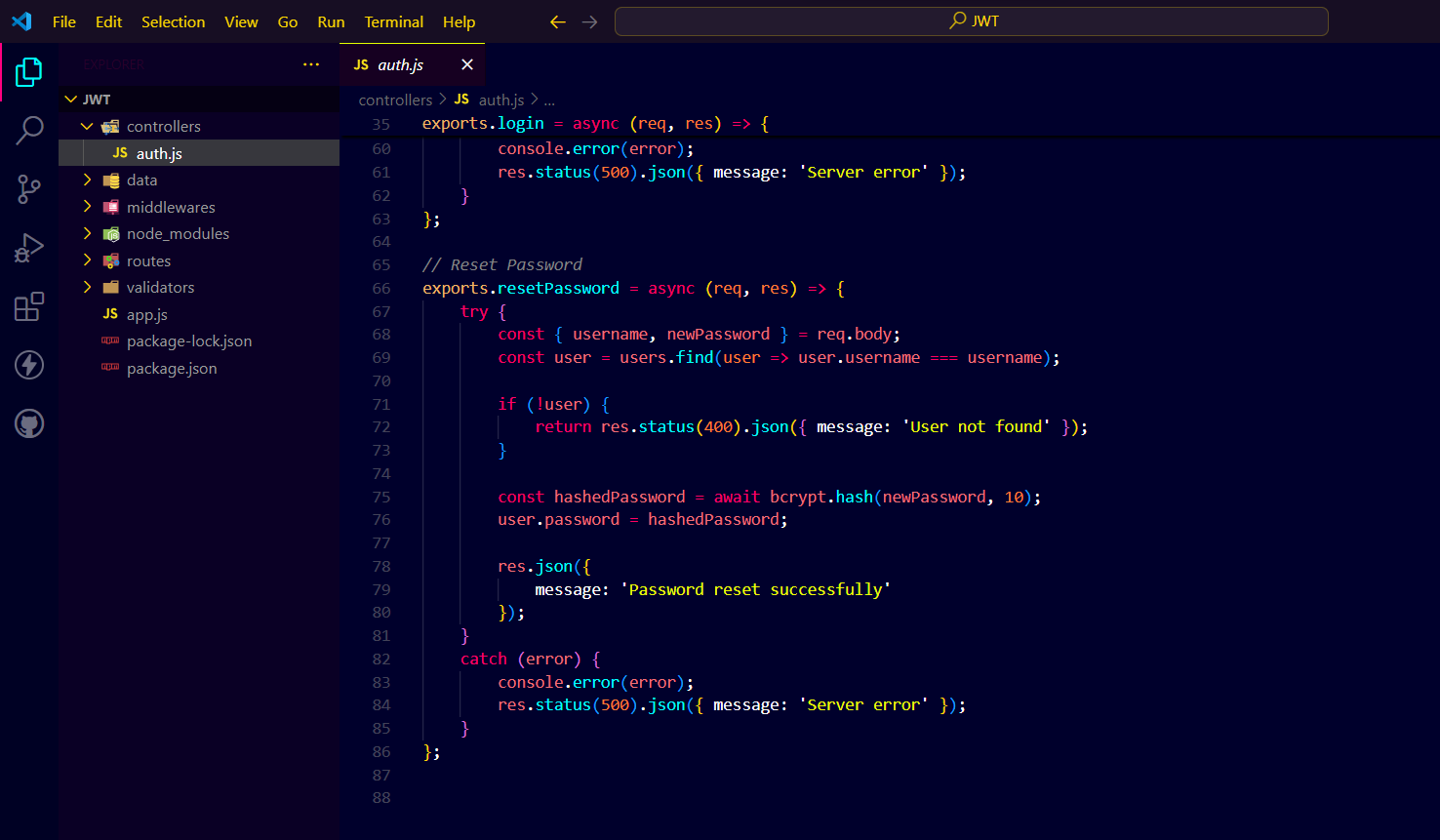
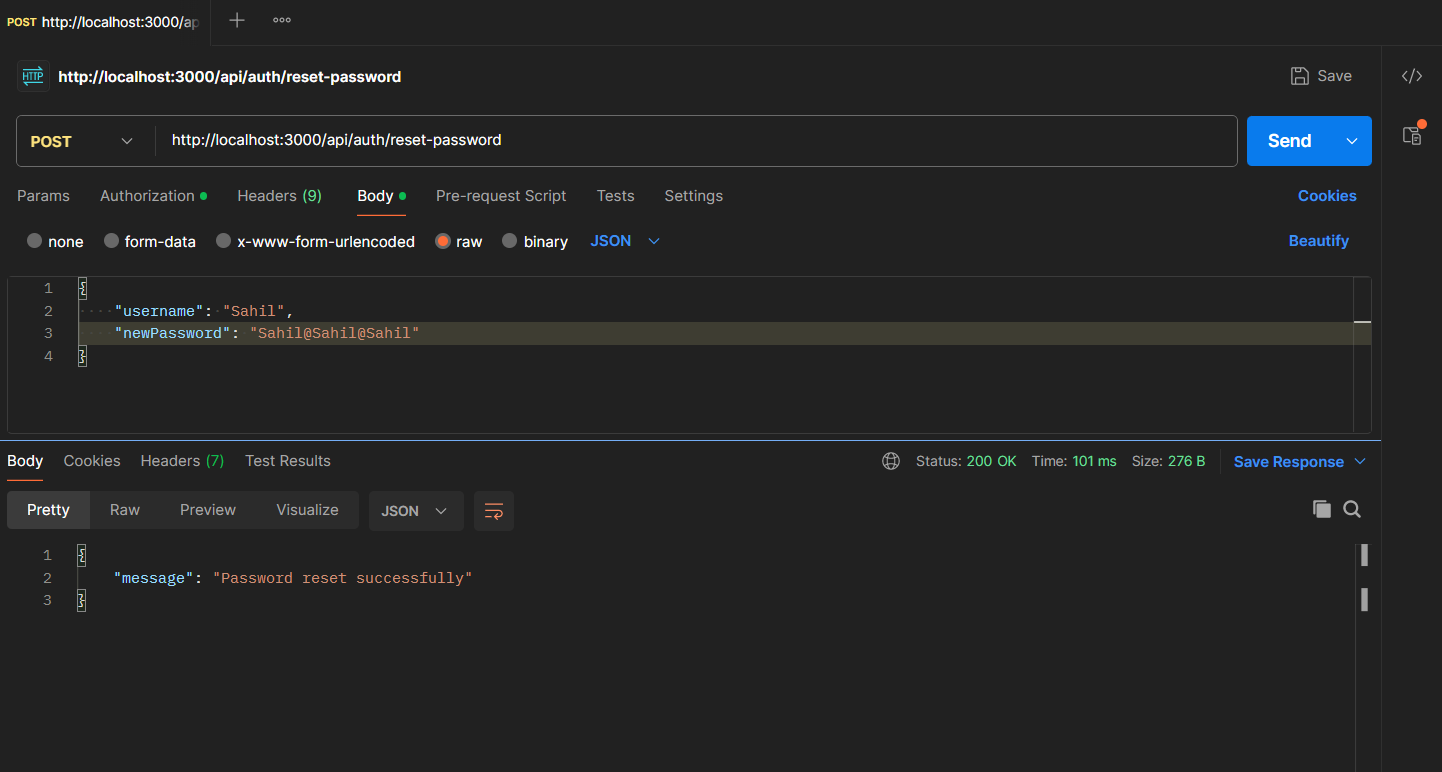
**Joi Schema**

* **Joi Schema is a JavaScript validation library. It helps to define and validate structures. With Joi, we can specify the expected format, types, and rules for data, such as mandatory fields, default values, patterns, and custom validations. In Express.js applications, Joi is used for verifying request bodies, query parameters, and input data. This ensures that only valid data is handled by the application.**
* ****

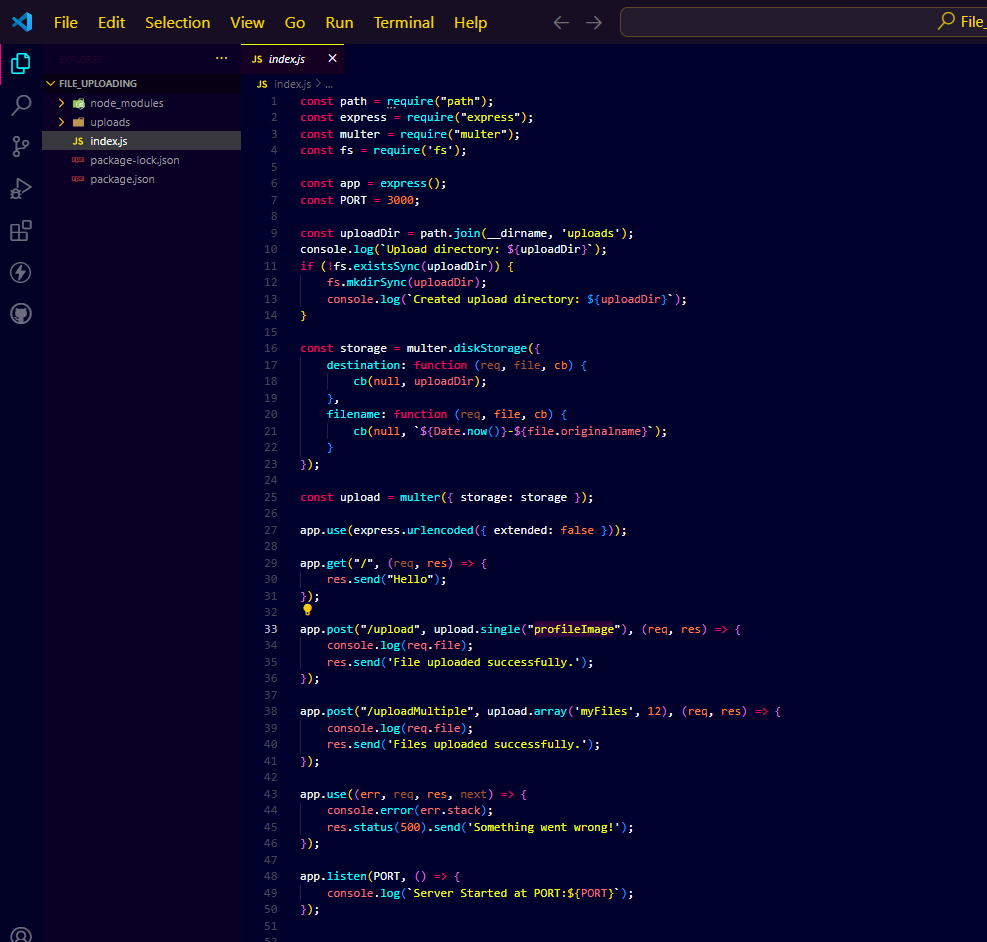
**Bycrypt Library**

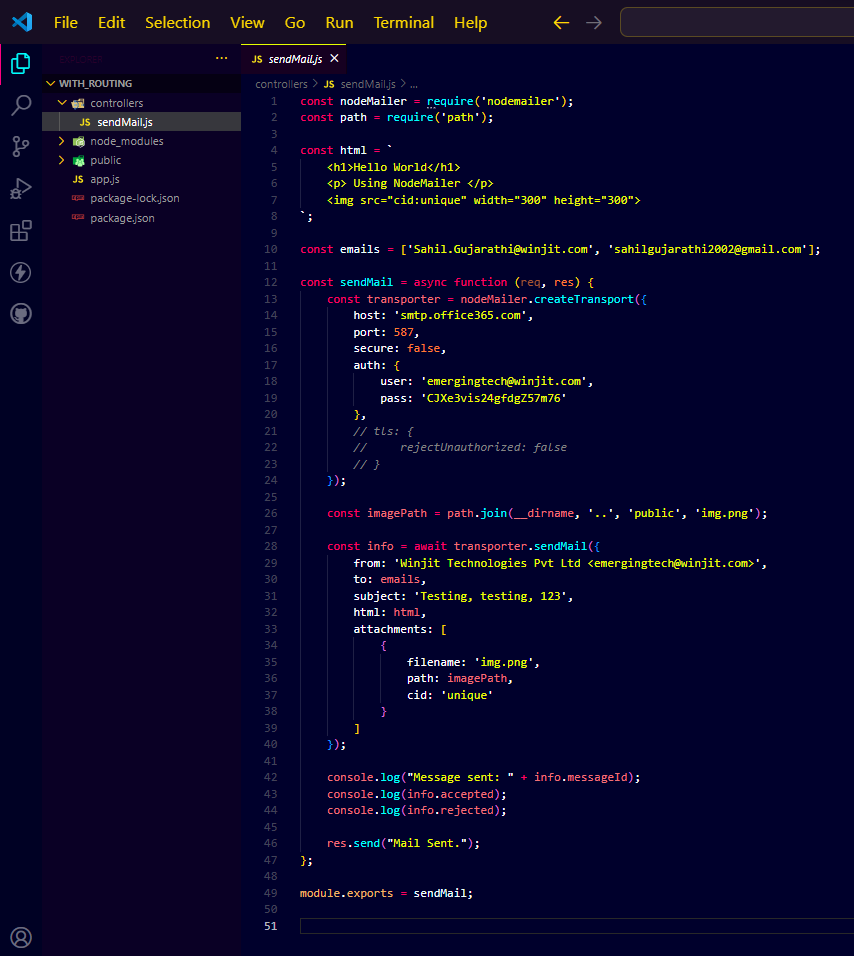
**• Bcrypt is a library used in Node.js apps to hash passwords. It applies a hashing method. This makes sure that even if a database is hacked, original passwords are hard to recover. Bcrypt is used for securing user authentication. It defends brute force attacks and password cracking. we use bcrypt to hash passwords during signup and check them during login.**

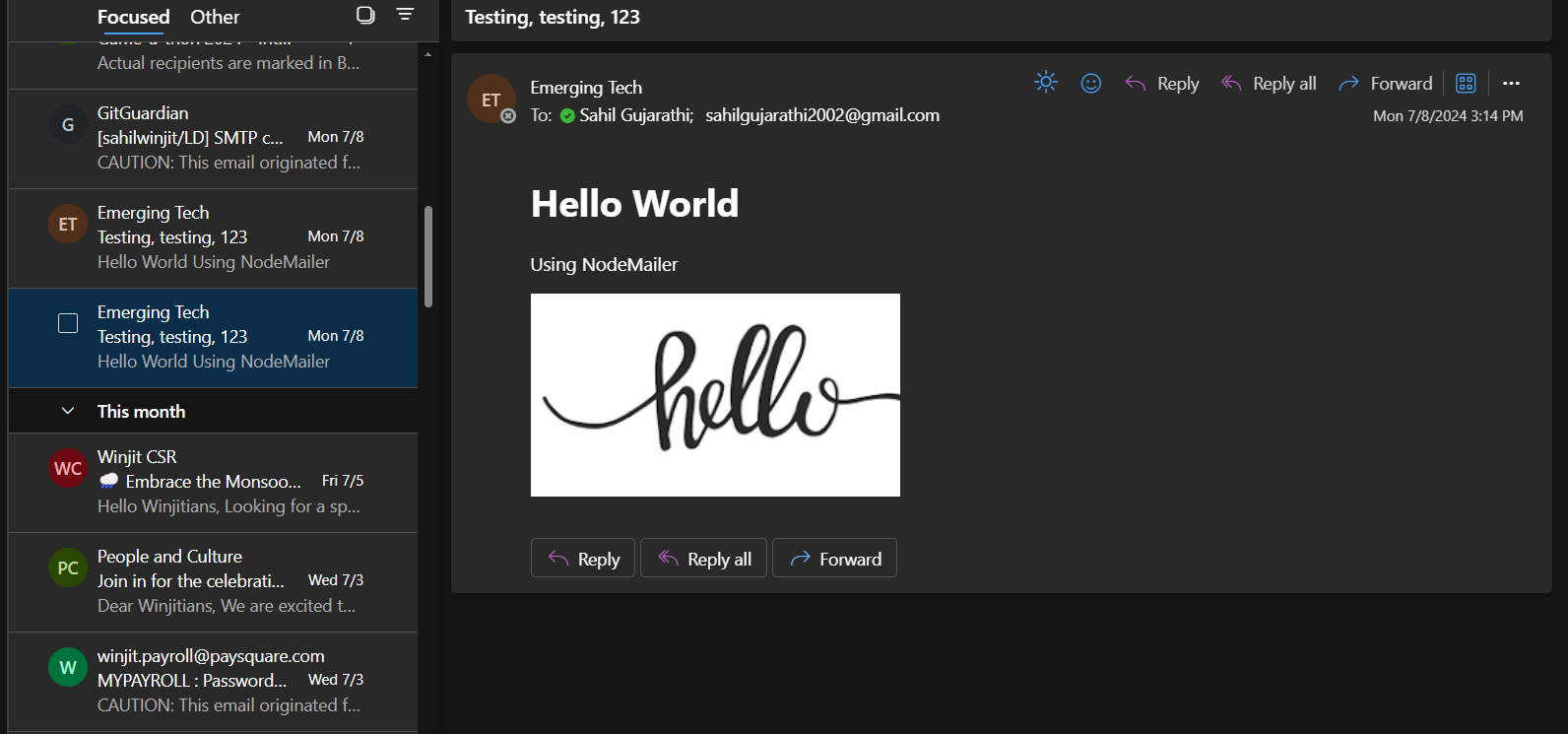
**JWT Token and Authentication**

* **JWT Token: Creating and verifying JWT tokens.**
* **JWT token stands for JSON web token where a token is generated for that particular user at the time of login, that token generated can be used to retreive the data of user. Using the token the user can be validated.**
* **Creating a JWT Token:**
  + **To create a JWT, we need a secret key and a payload. The payload is the encoded data within the token, which includes user information.**
  + ****
  + ****
* **Verifying a JWT Token:**
  + **To verify a JWT, we decode it using the same secret key that was used to create it.**
  + ****
  + ****
* **Authentication: Implementing signup, login, and reset password functionalities.**
  + **Authentication means confirming who a user is. In web apps, it often involves checking user credentials (like username and password) and giving tokens for future use.**
  + **Signup: The process of creating a new user account. The user's details are validated, hashed passwords are stored.**
  + ****
  + ****
  + **Login: The process where a user submits their credentials to gain access to web app. The credentials are validated, and if they match, a JWT token is issued.**
  + ****
  + ****
  + **Reset Password: The process where a user requests to reset their password. This typically involves verifying the user's identity and then allowing the user to set a new password.**
  + ****
  + ****

**File Handling in Node.js**

* **Multer Package: Using Multer for file uploads.**
  + **Multer is a middleware used in Node.js to manage form-data, mainly for uploading files. It simplifies the process of file uploading in web applications.**
* **NodeMailer: Sending emails using NodeMailer.**
  + **NodeMailer is a module for Node.js applications that simplifies the process of sending emails.**





**MySQL**

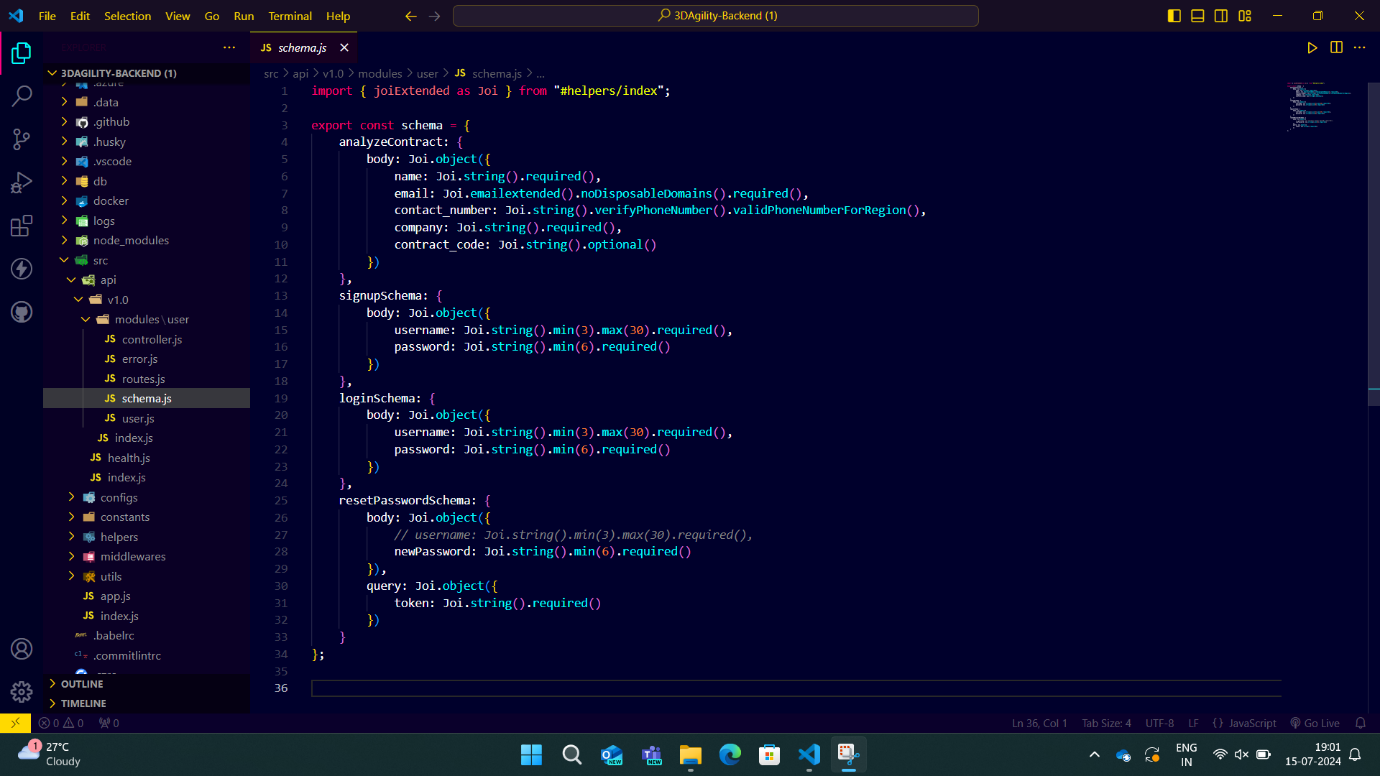
**MySQL Basics**

* **Creating Databases and Tables: Setting up databases and tables in MySQL.**
* **SELECT Clause: Retrieves data from a database.**
* **WHERE Clause: Filters records based on specified conditions.**
* **AND, OR, NOT: Logical operators used to combine conditions.**
* **IN: Checks if a value matches any value in a list.**
* **BETWEEN: Checks if a value is within a range.**
* **LIKE: Searches for a pattern in a column.**
* **REGEXP: Searches for a pattern using a regular expression.**
* **IS NULL: Checks for NULL values.**
* **ORDER BY: Sorts the result set in ascending or descending order.**
* **LIMIT Operator: Limits the number of rows returned in a result set.**
* **Inner Joins: Retrieves records that have matching values in both tables.**
* **Outer Joins: Retrieves all records from one table and the matched records from another.**
* **Outer Join Between Multiple Tables: Extends outer join to more than two tables.**
* **Unions: Combines the result sets of two or more SELECT statements.**
* **Column Attributes: Properties or characteristics of a column.**
* **Inserting Rows: Adds new records to a table.**
* **Creating copy of table: Duplicates the structure and data of a table.**
* **Updating a single row, multiple rows: Modifies existing records in a table.**
* **Using subqueries in updates: Performs updates based on the result of a nested SELECT statement.**
* **Deleting rows: Removes records from a table based on specified conditions.**
* **CRUD Operations: Performing Create, Read, Update, Delete operations.**
* **Stored Procedures: Predefined SQL queries stored in the database server for reuse. They can accept parameters and execute multiple SQL statements.**
* **SQL Rollback and Commit: Transaction control statements used to manage the changes made by SQL statements. COMMIT saves changes permanently, while ROLLBACK reverts changes if an error occurs or if explicitly commanded.**

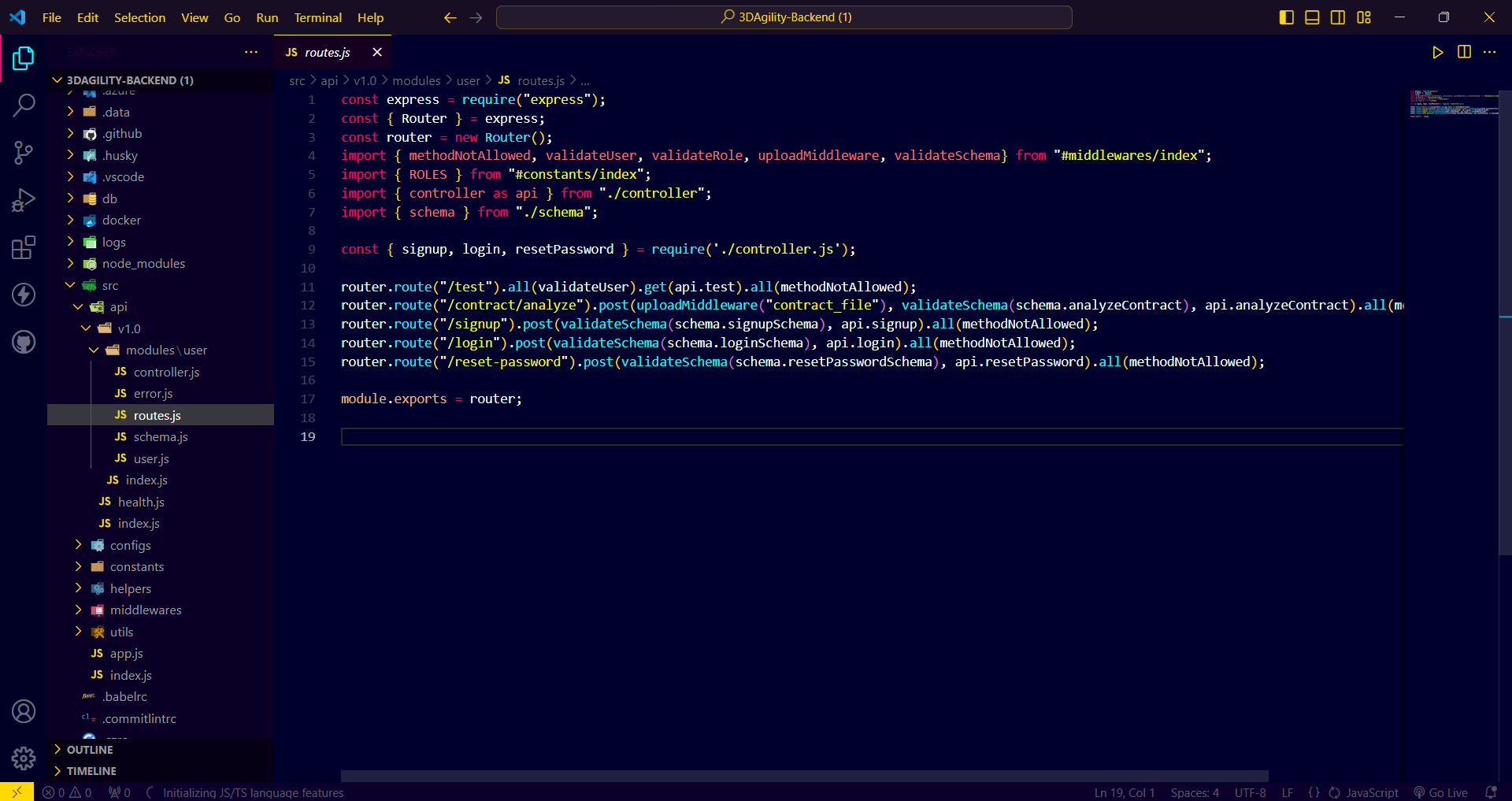
**Task 1**

**In my organized Node.js project, I created APIs for user authentication: Signup, Login, and Reset Password. Each API was designed with custom error handling for secure user interactions. I integrated JWT token management for authentication, using custom functions for generating and verifying tokens. Implemented Bcrypt functions to securely hash passwords. I also applied Joi schema validation to enforce data integrity and validate user inputs efficiently. The project was structured into separate components like controllers, schemas, and routes, ensuring scalability and maintainability of the codebase.**

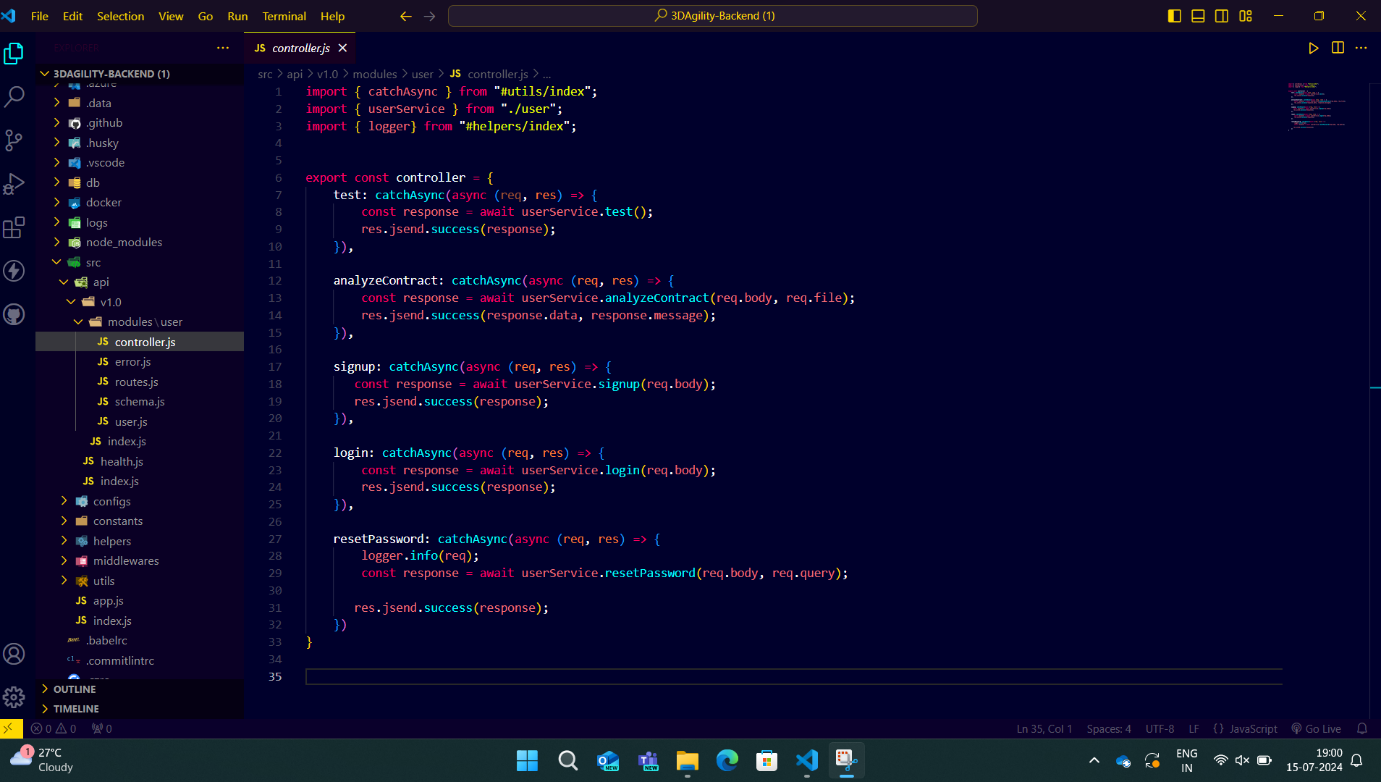
**Joi Schema:-**

****

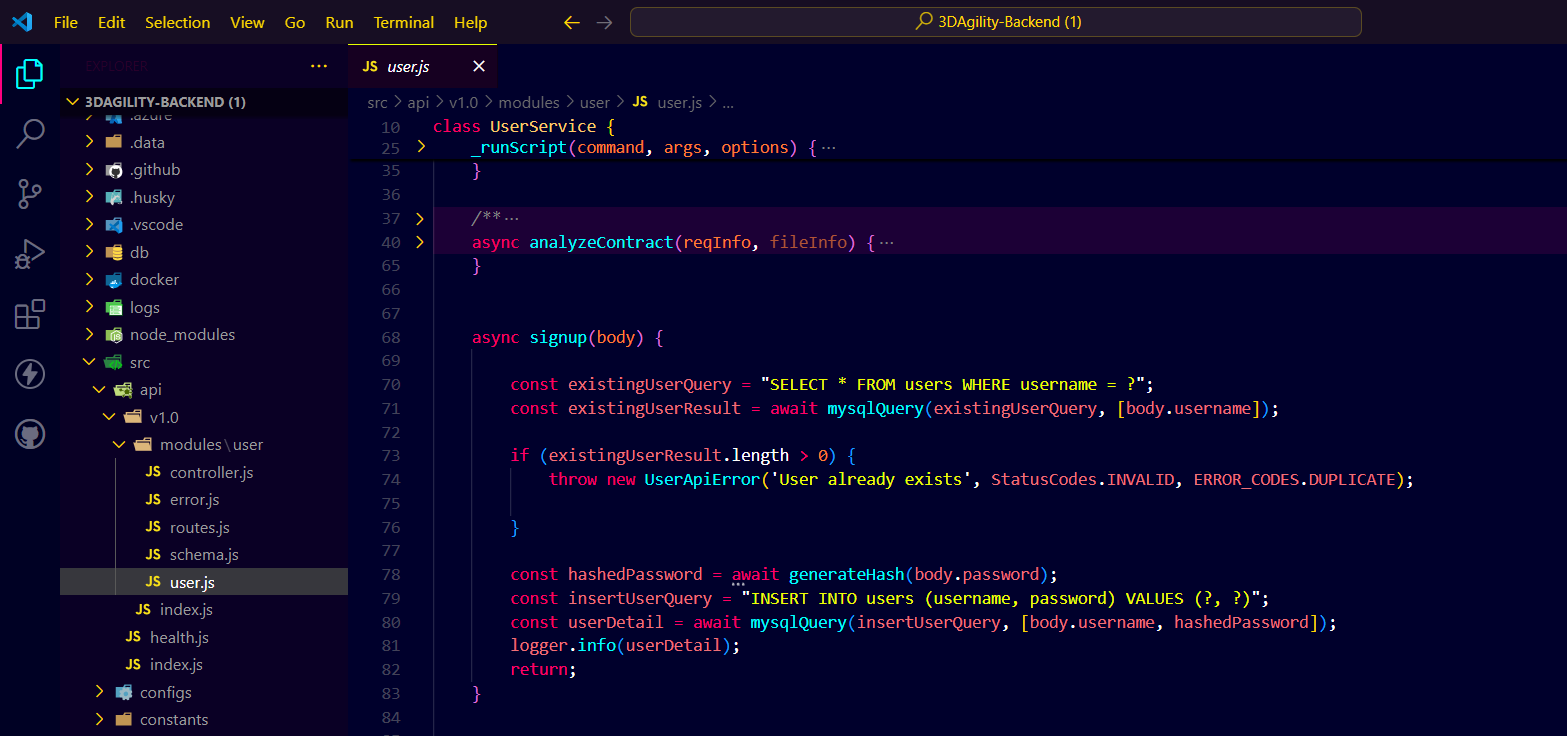
**Routing:-**

****

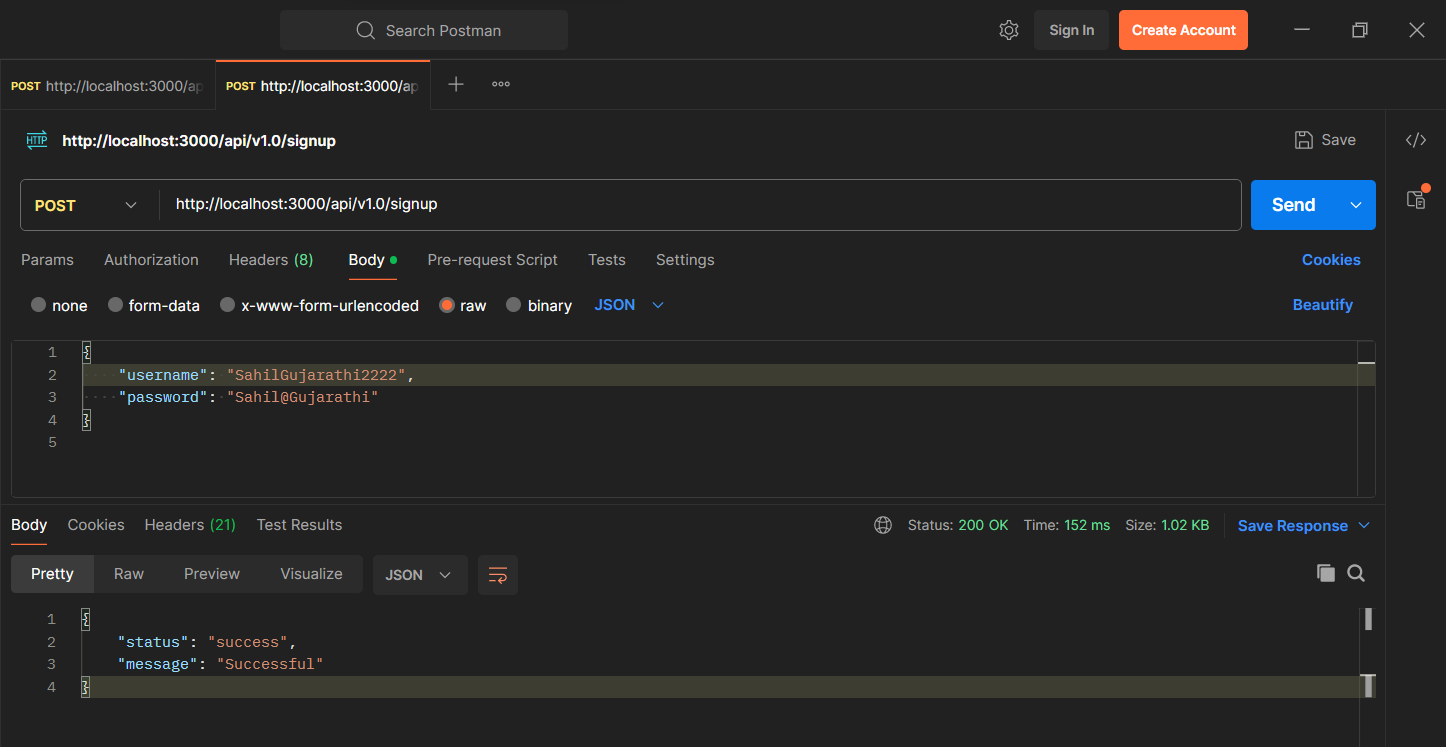
**Controller File:-**

****

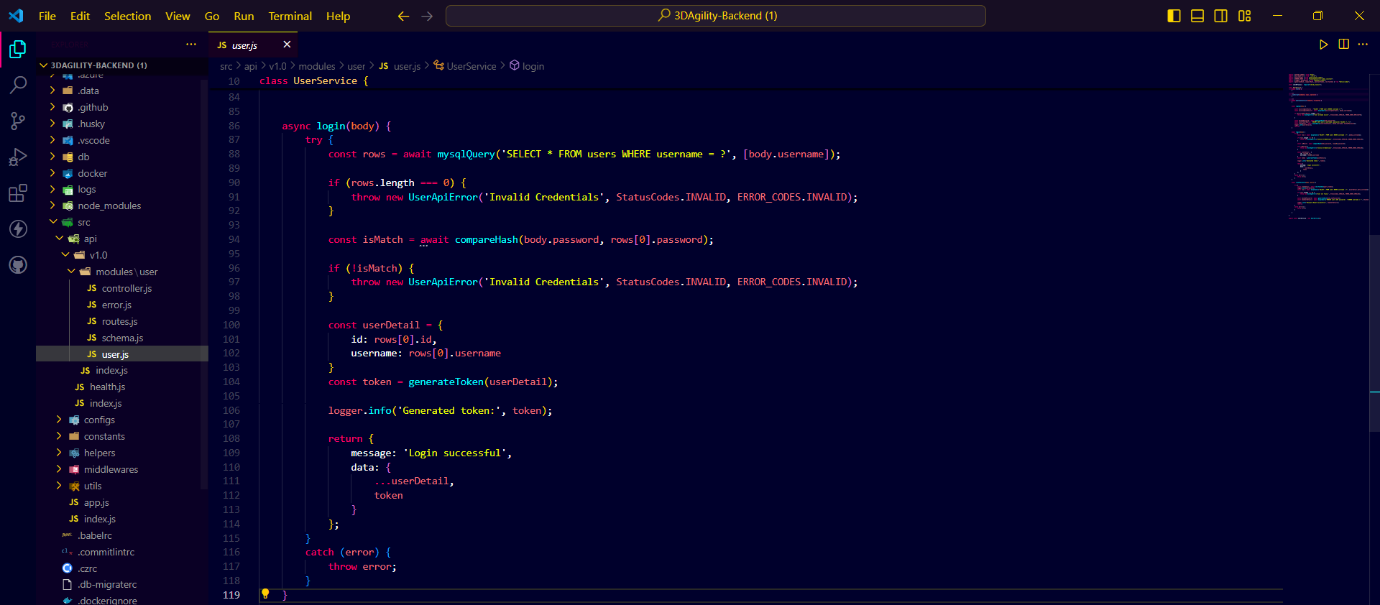
**Signup Code:-**

****

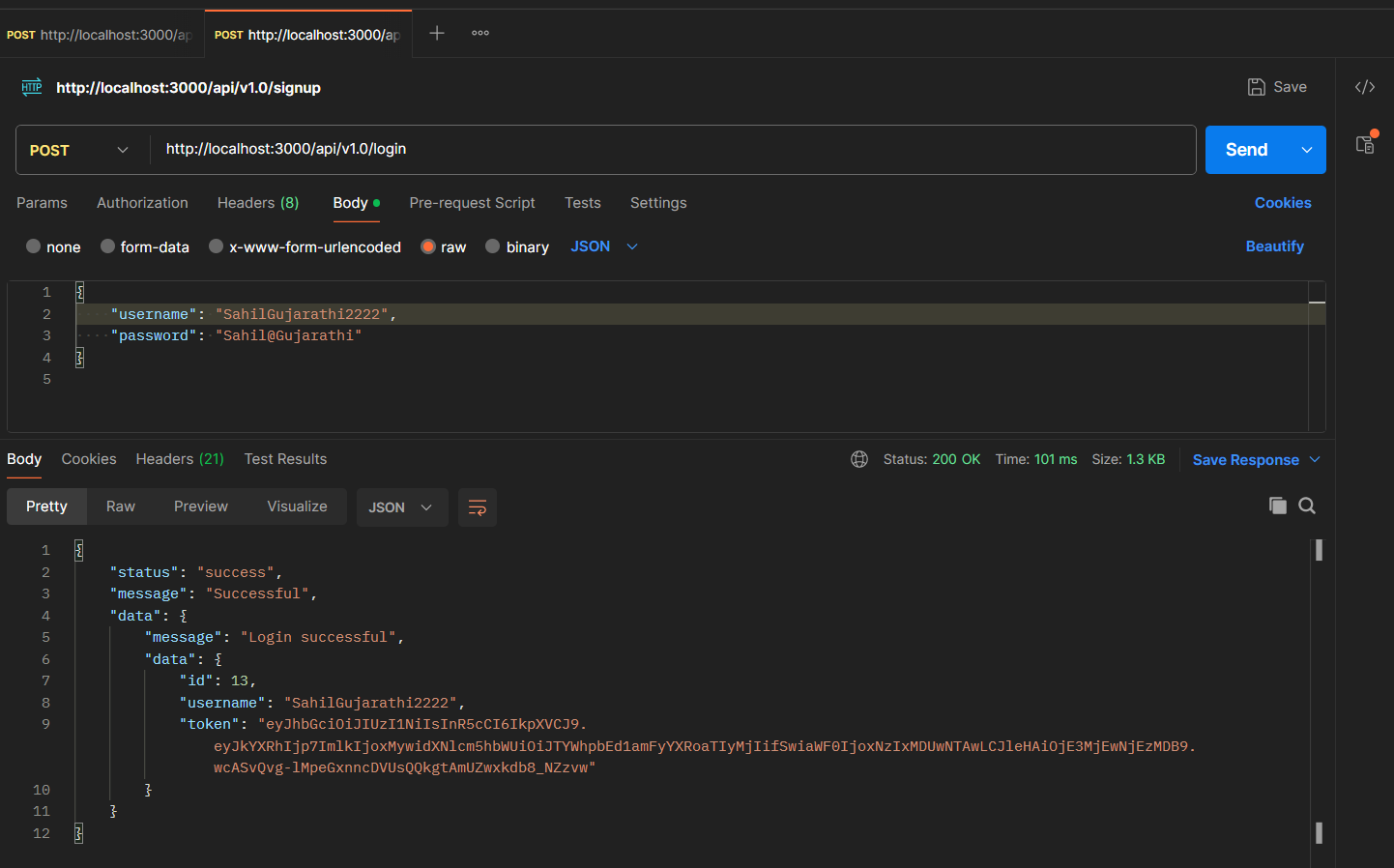
**Signup API on postman:-**

****

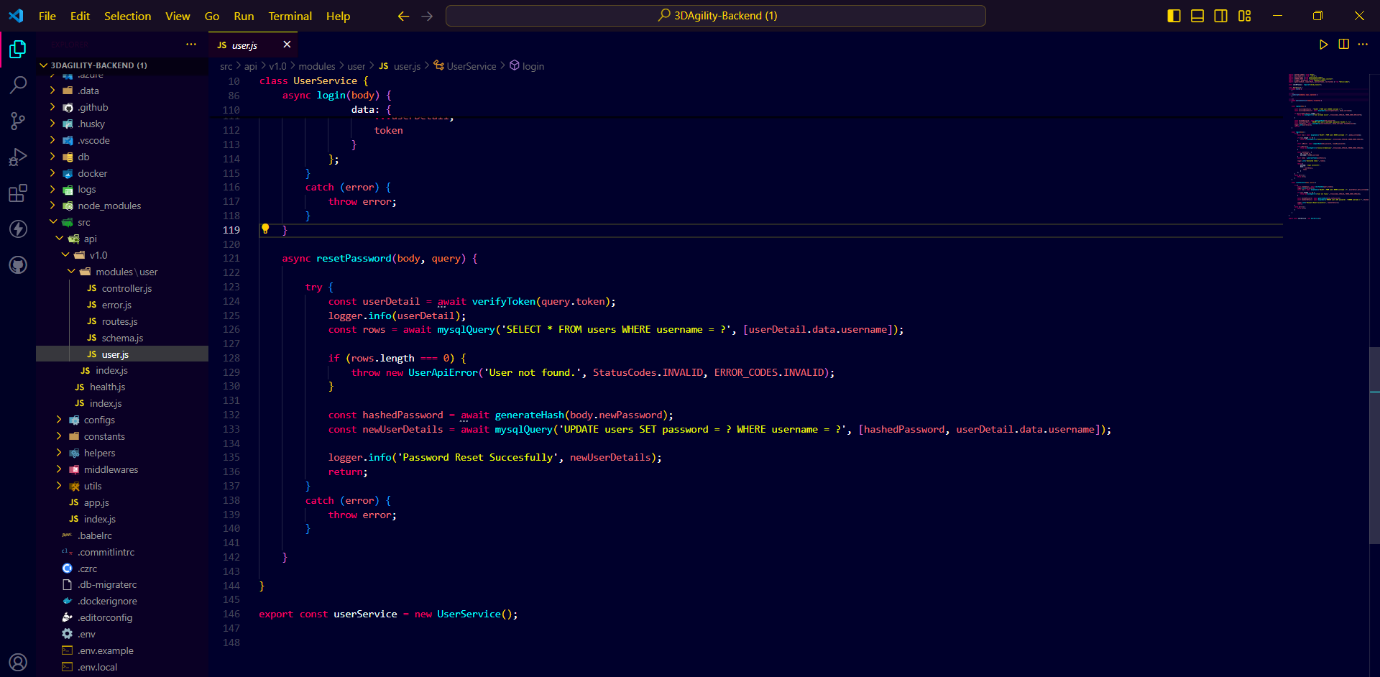
**Login Code:-**

****

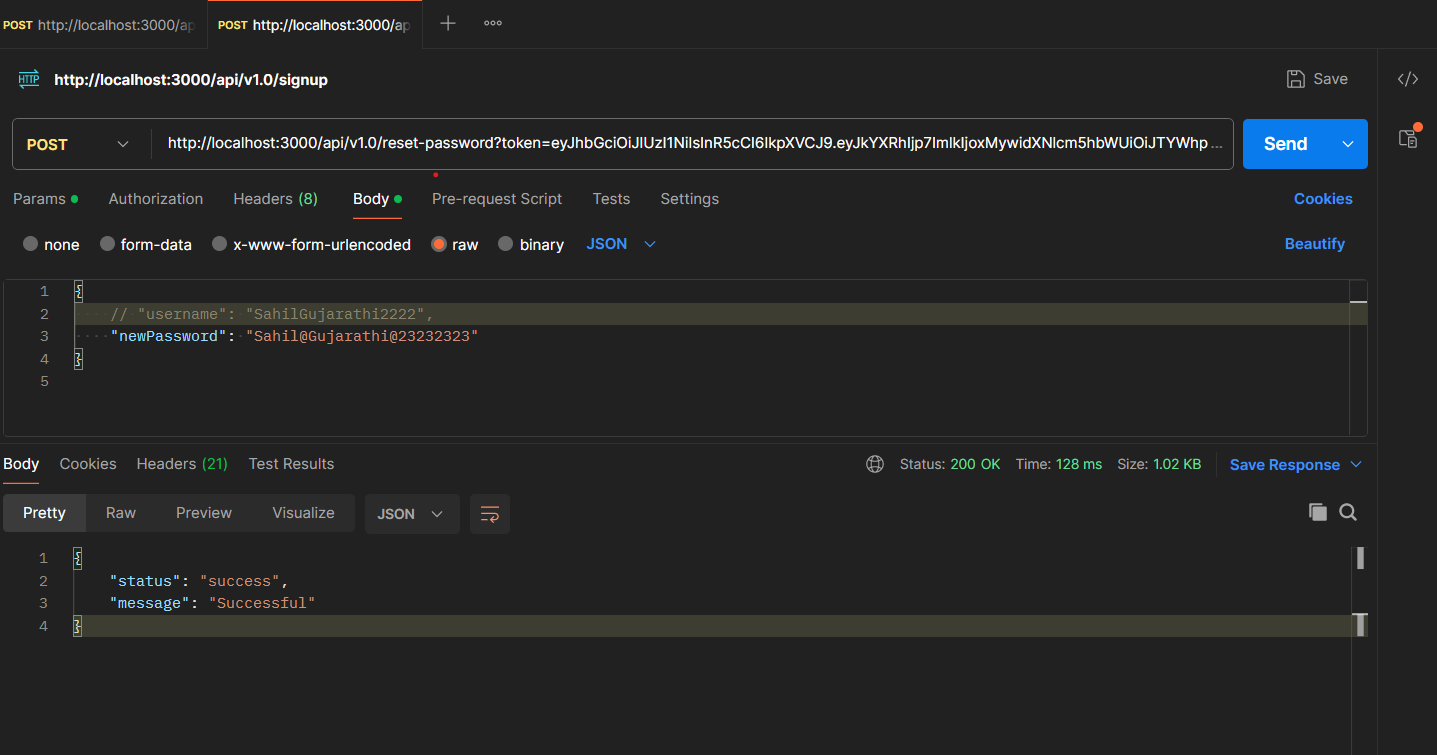
**Login API on postman:-**

****

**Reset Password Code:-**

****

**Reset Password API on postman:-**

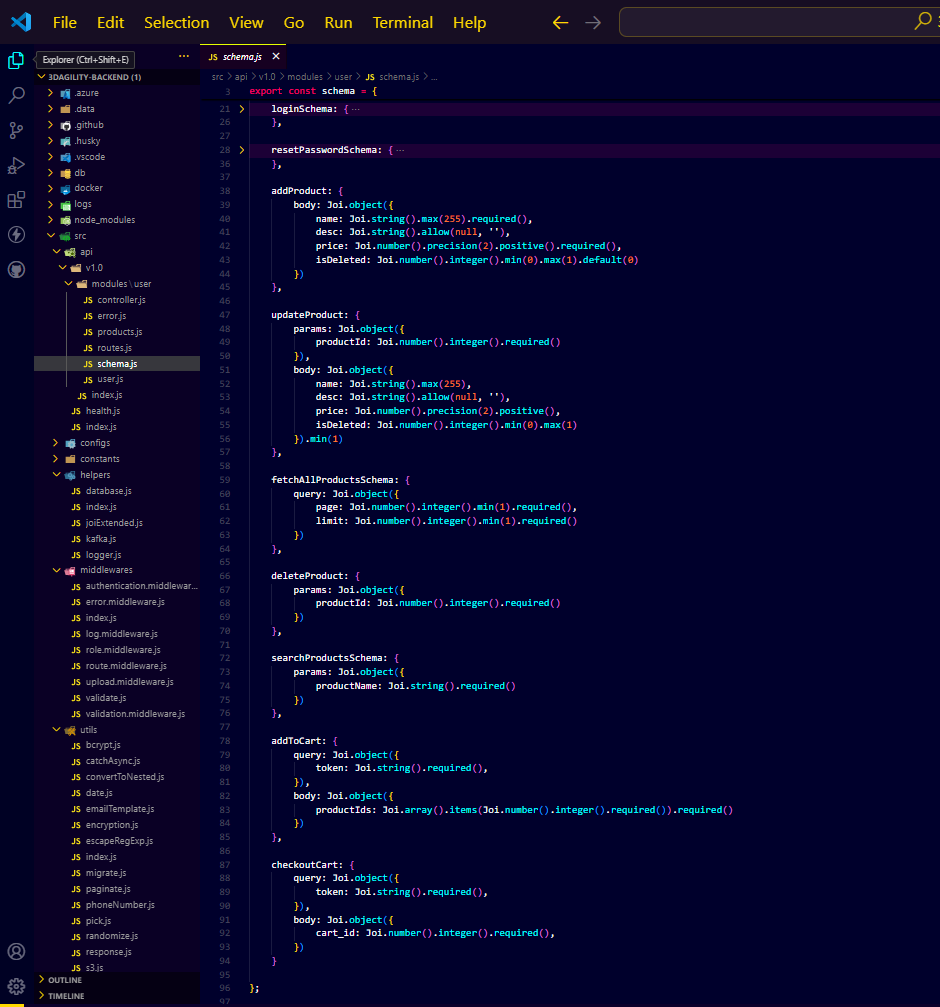
****

**Task 2**

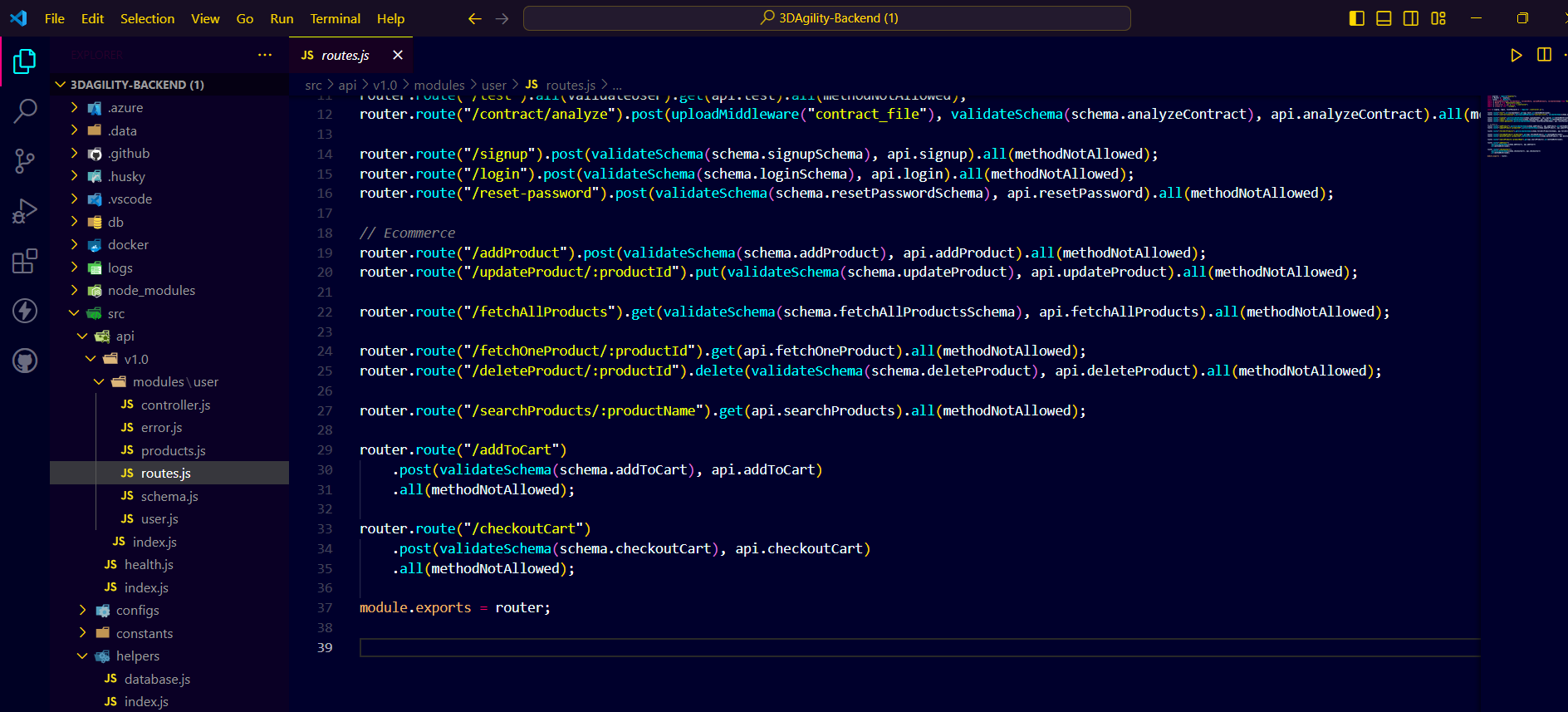
**In my organized Node.js project, I created e-commerce APIs for adding, updating, fetching, deleting, and searching products, as well as for managing the cart and checkout process. Each API includes custom error handling to ensure secure user interactions. I integrated JWT token management for authentication, using custom functions to generate and verify tokens. Bcrypt functions were implemented to securely hash passwords. Additionally, I applied Joi schema validation to enforce data integrity and validate user inputs efficiently.**

**The project is structured into separate components such as controllers, schemas, and routes, ensuring scalability and maintainability of the codebase. Users can add products to the database, update product details, fetch product information, delete products, and search for products. They can also add products to their cart, with the system verifying the user before adding items. During checkout, the user receives the total bill, and the products are then removed from the cart.**

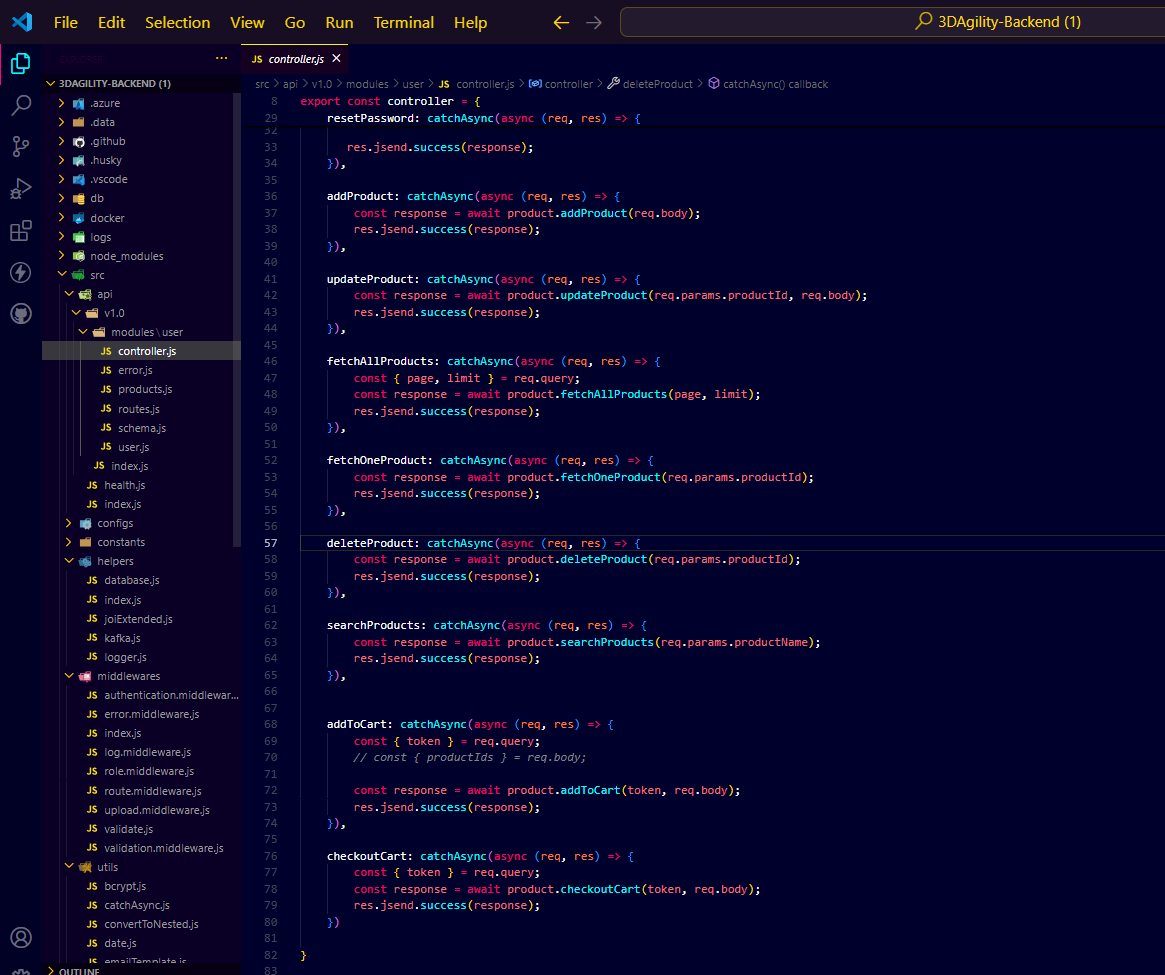
**Schema.js**

****

**Routes.js**

****

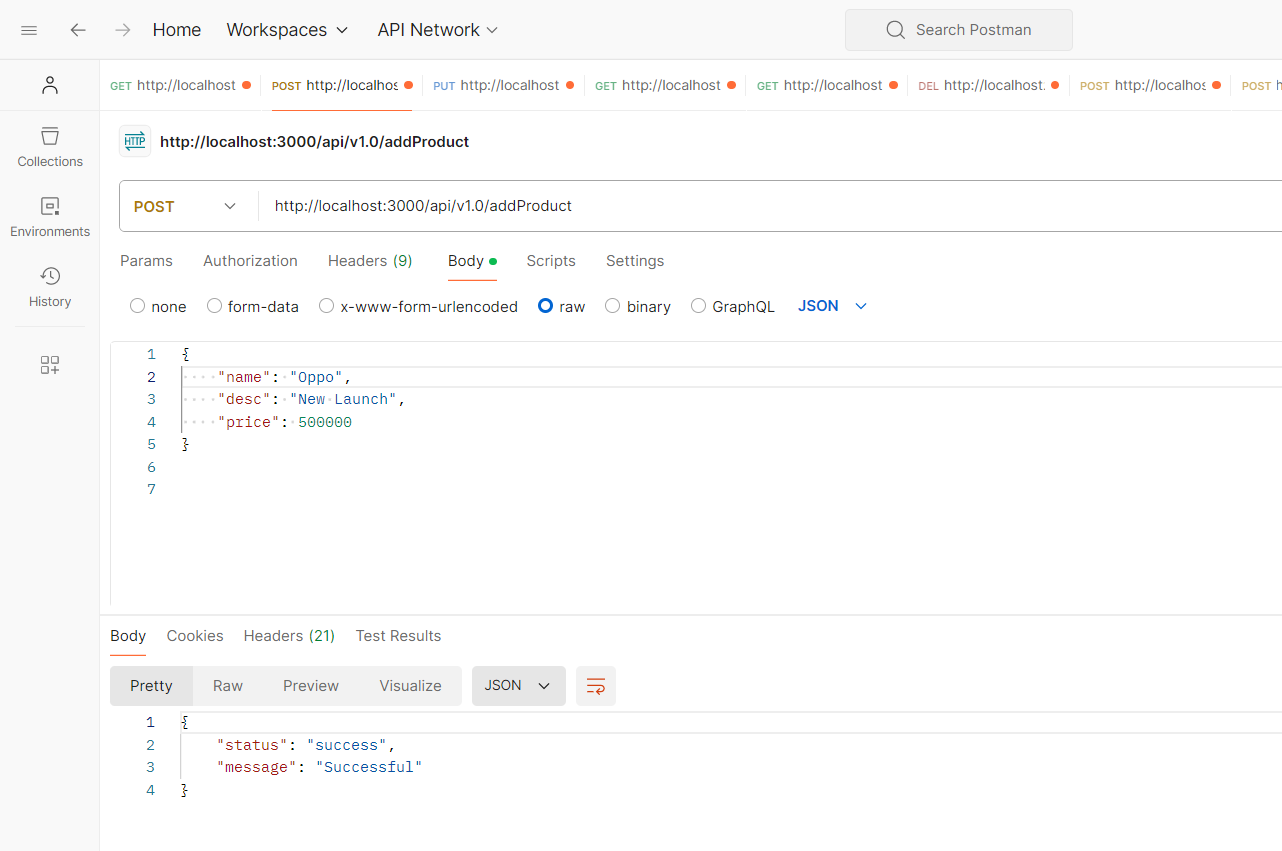
**Controller.js**

****

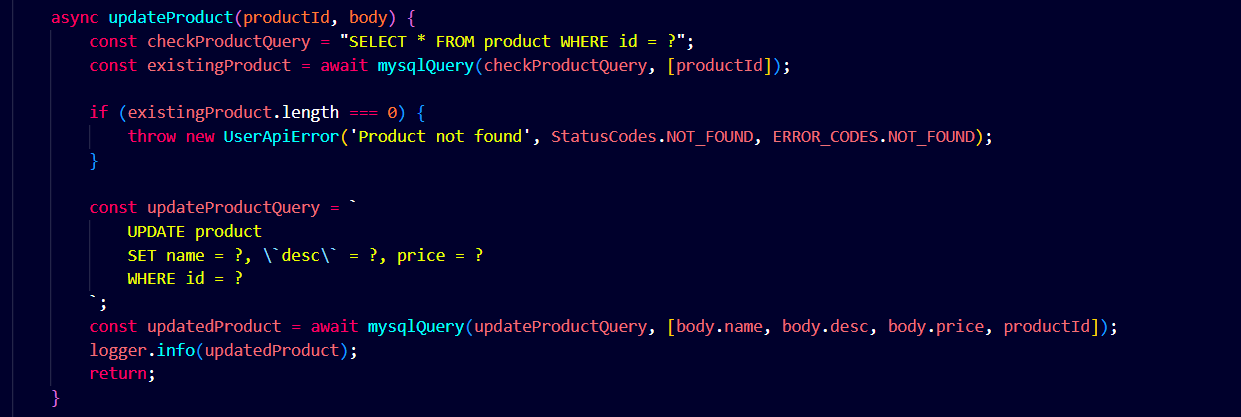
**Add Product Code:-**

****

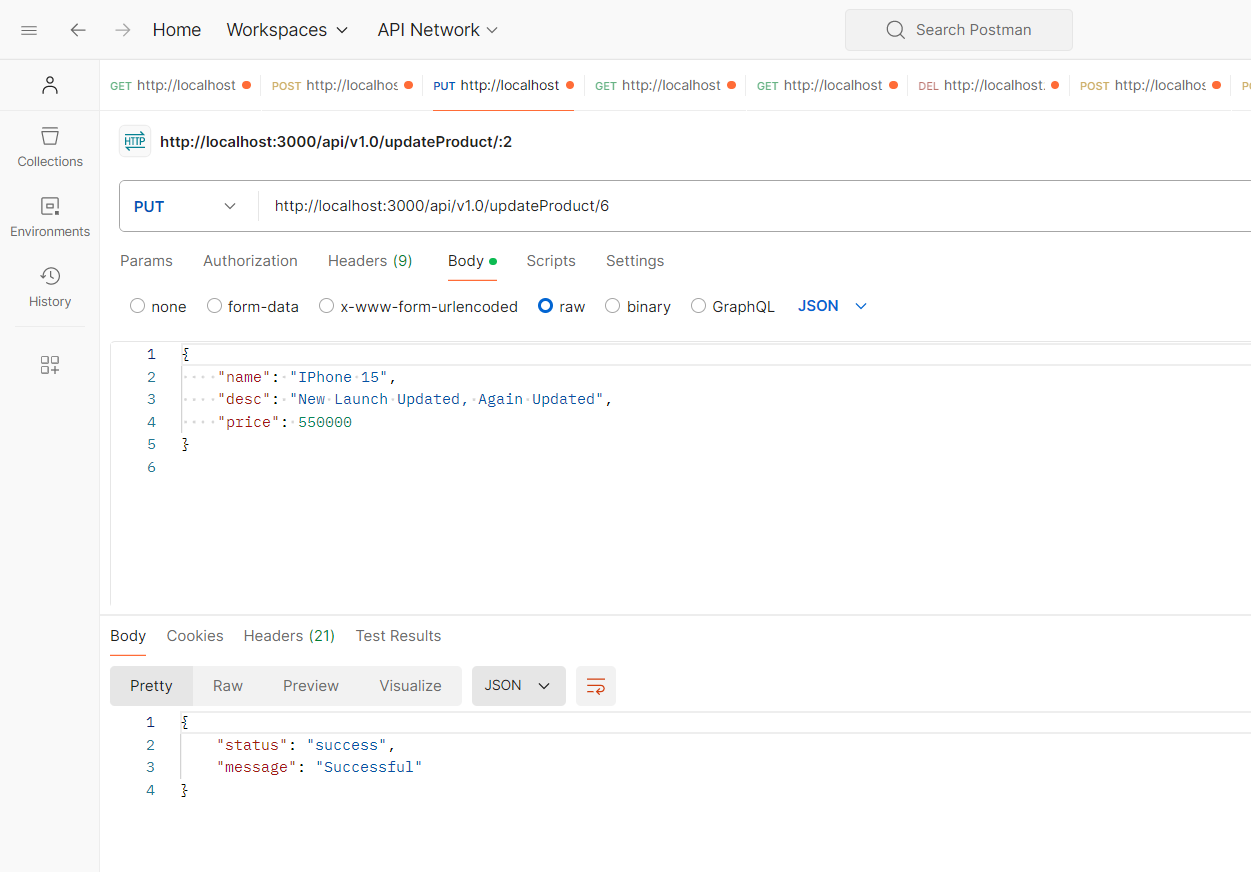
**Add Product API on Postman:-**

****

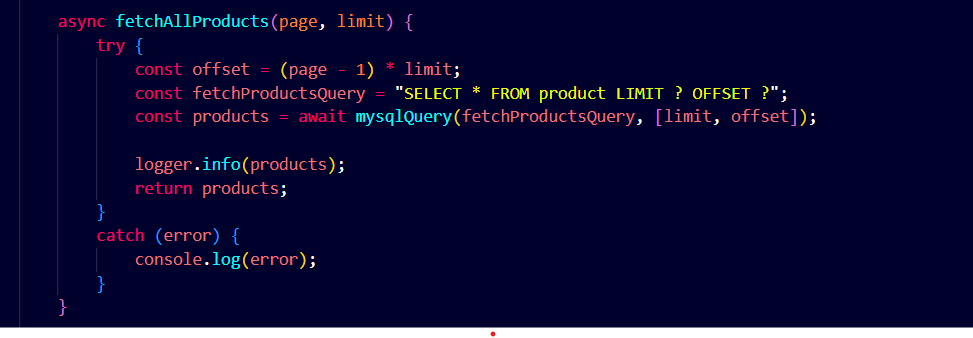
**Update Product Code:-**

****

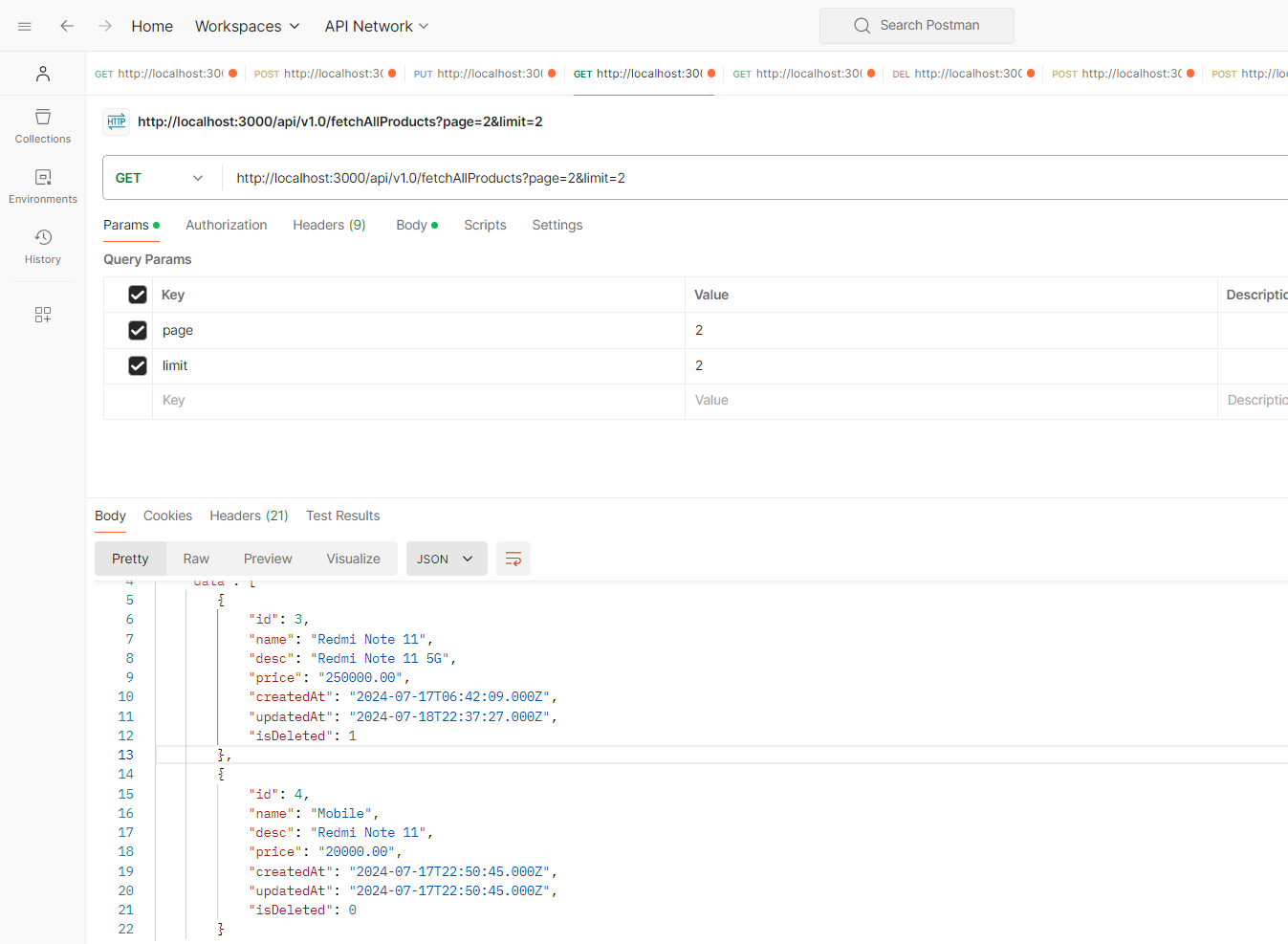
**Update Product API on Postman:-**

****

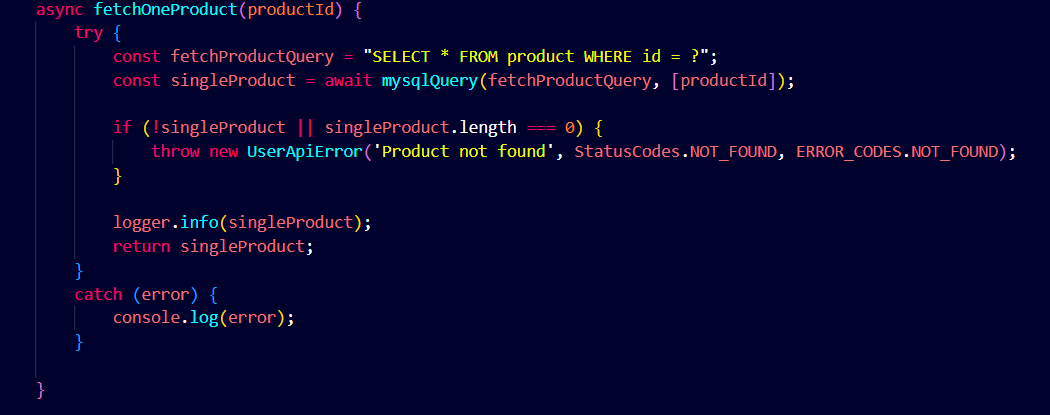
**Fetching all Products Code:-**

****

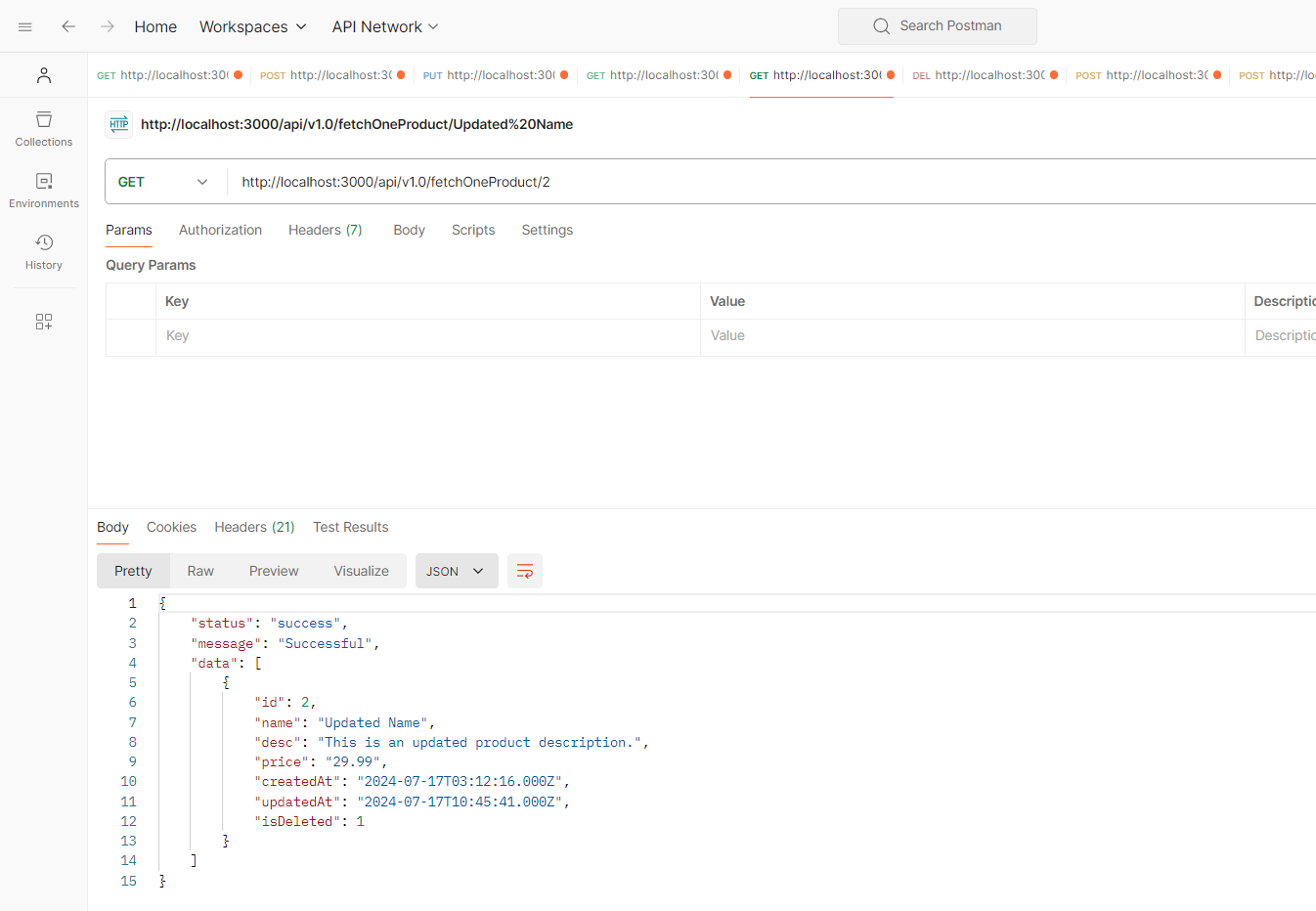
**Fetching all Products API on postman:-**

****

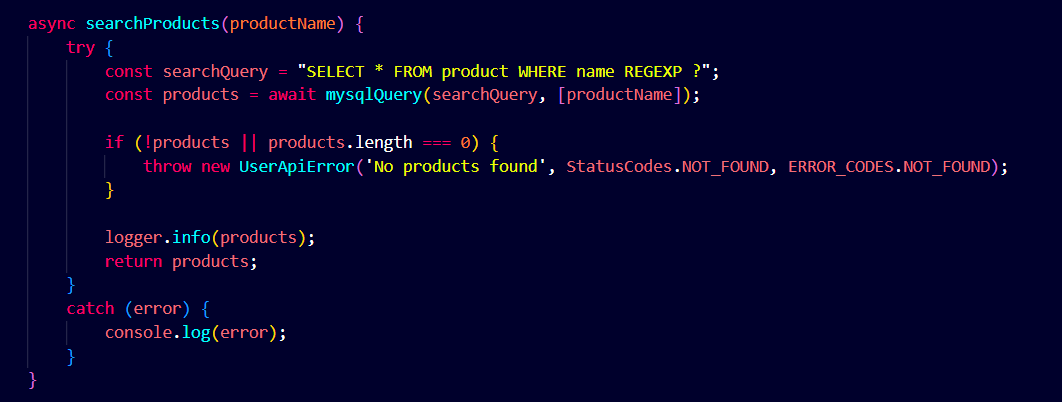
**Fetching Single Product:-**

****

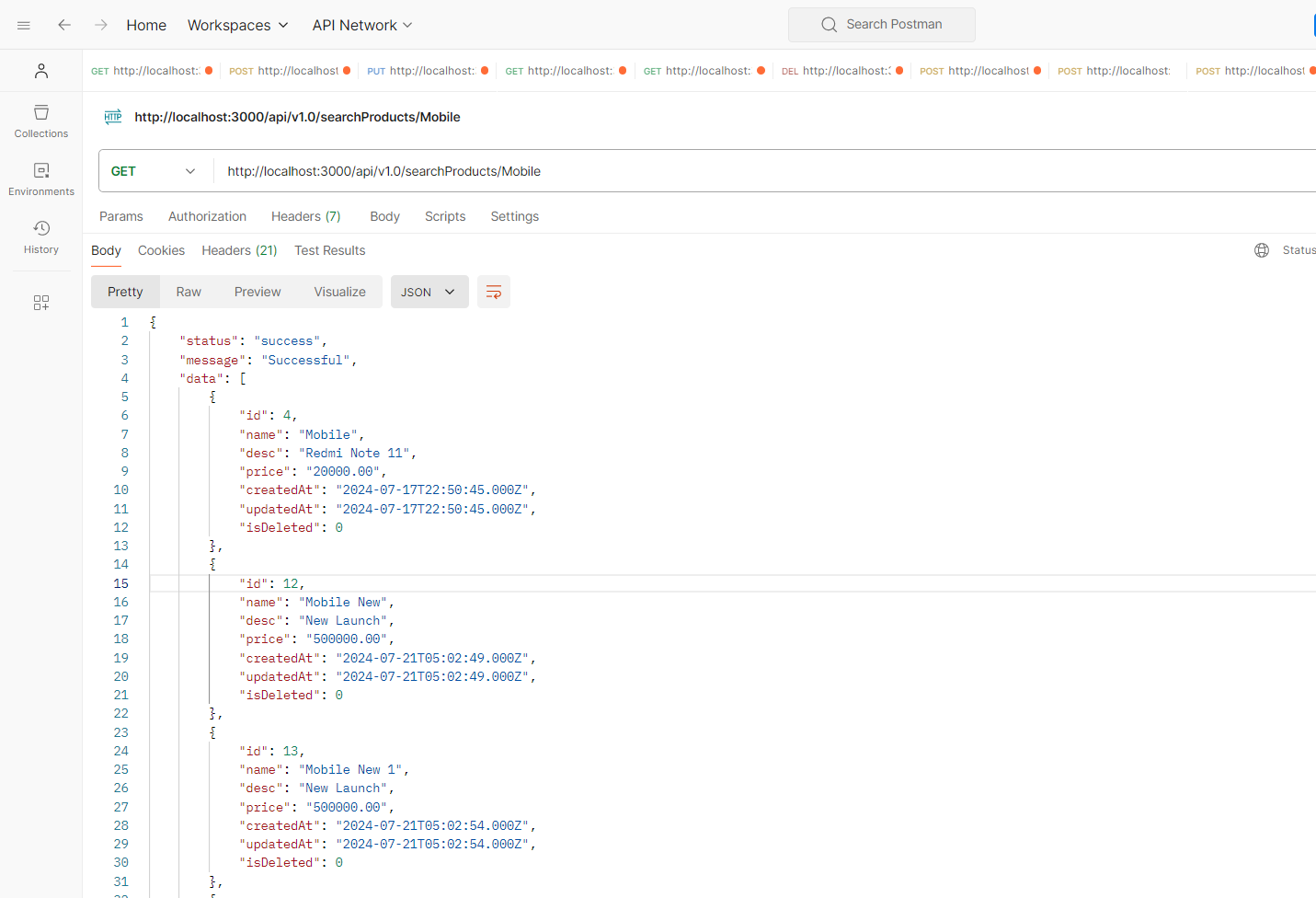
**Fetching Single Product API on postman:-**

****

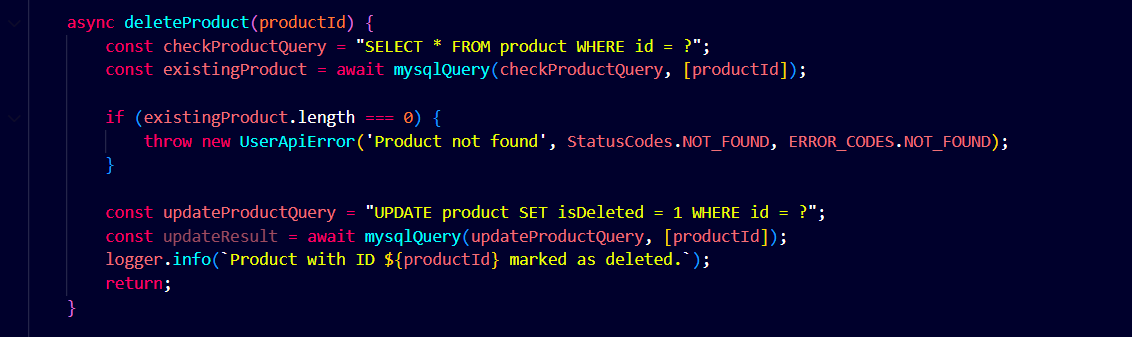
**Searching Products Code:-**

****

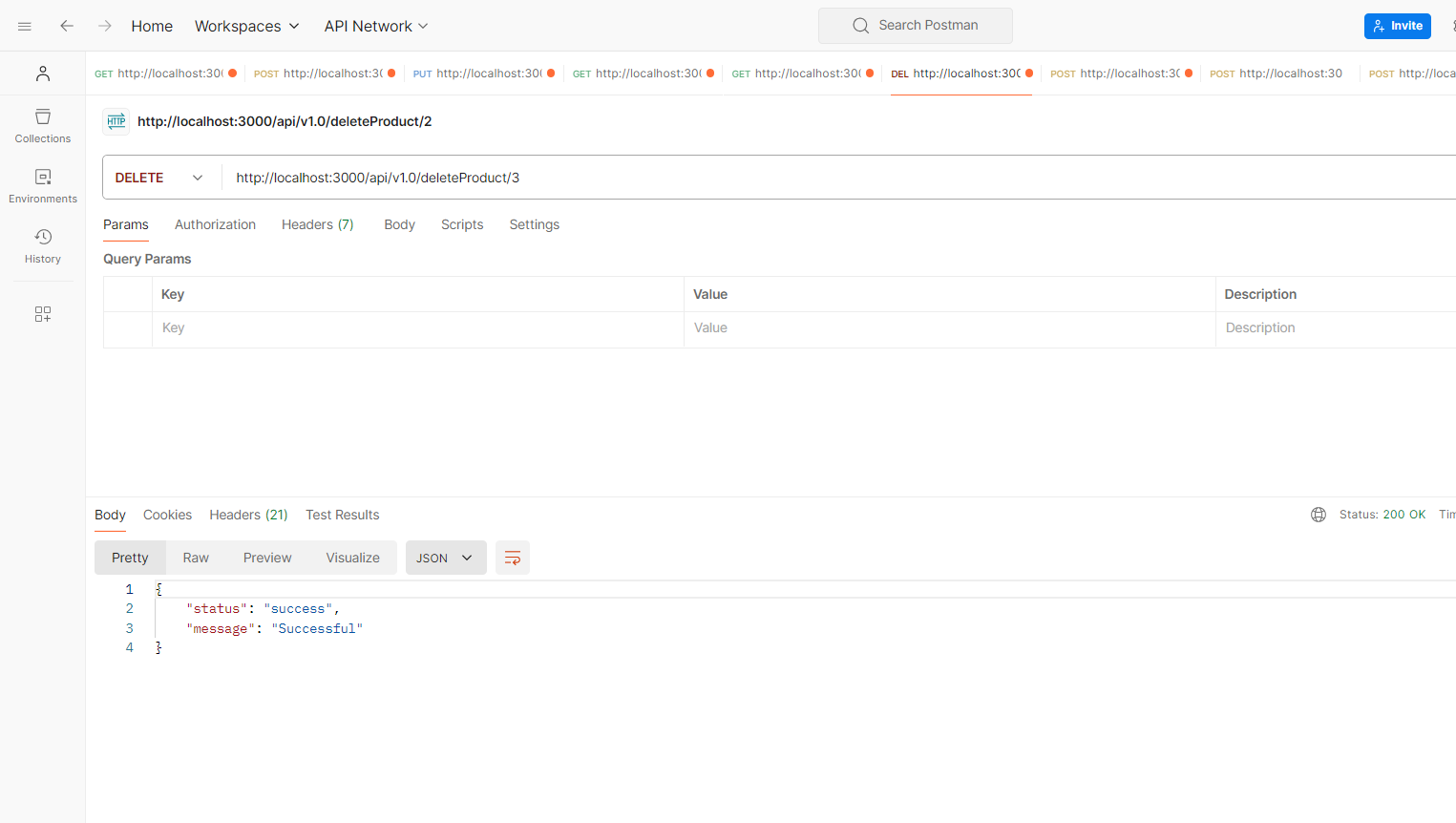
**Searching Products API on Postman:-**

****

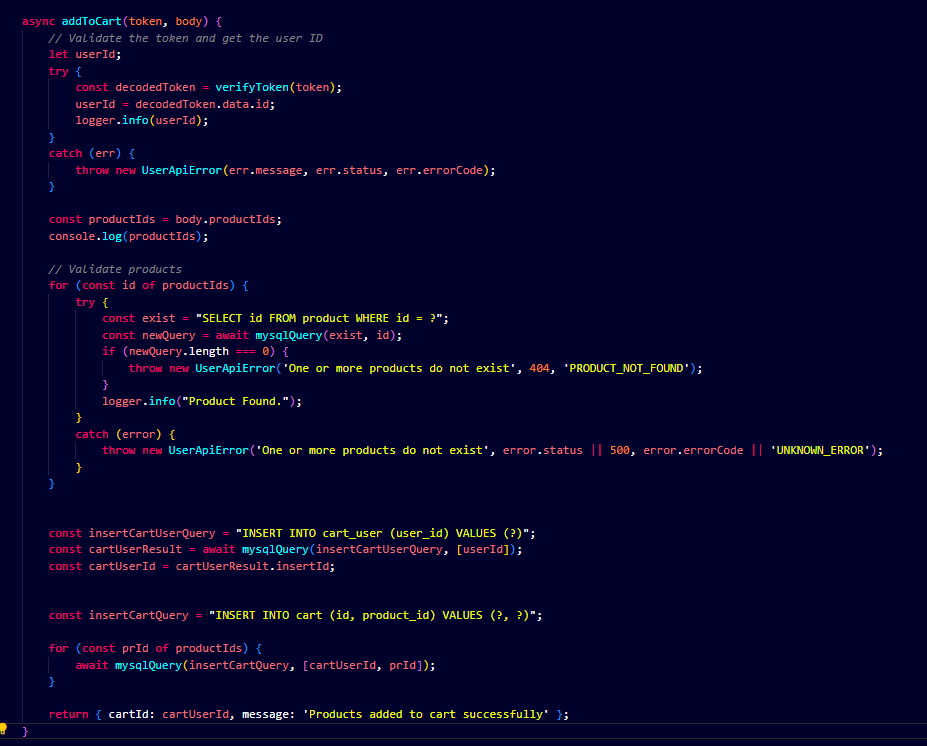
**Deleting Product Code:-**

****

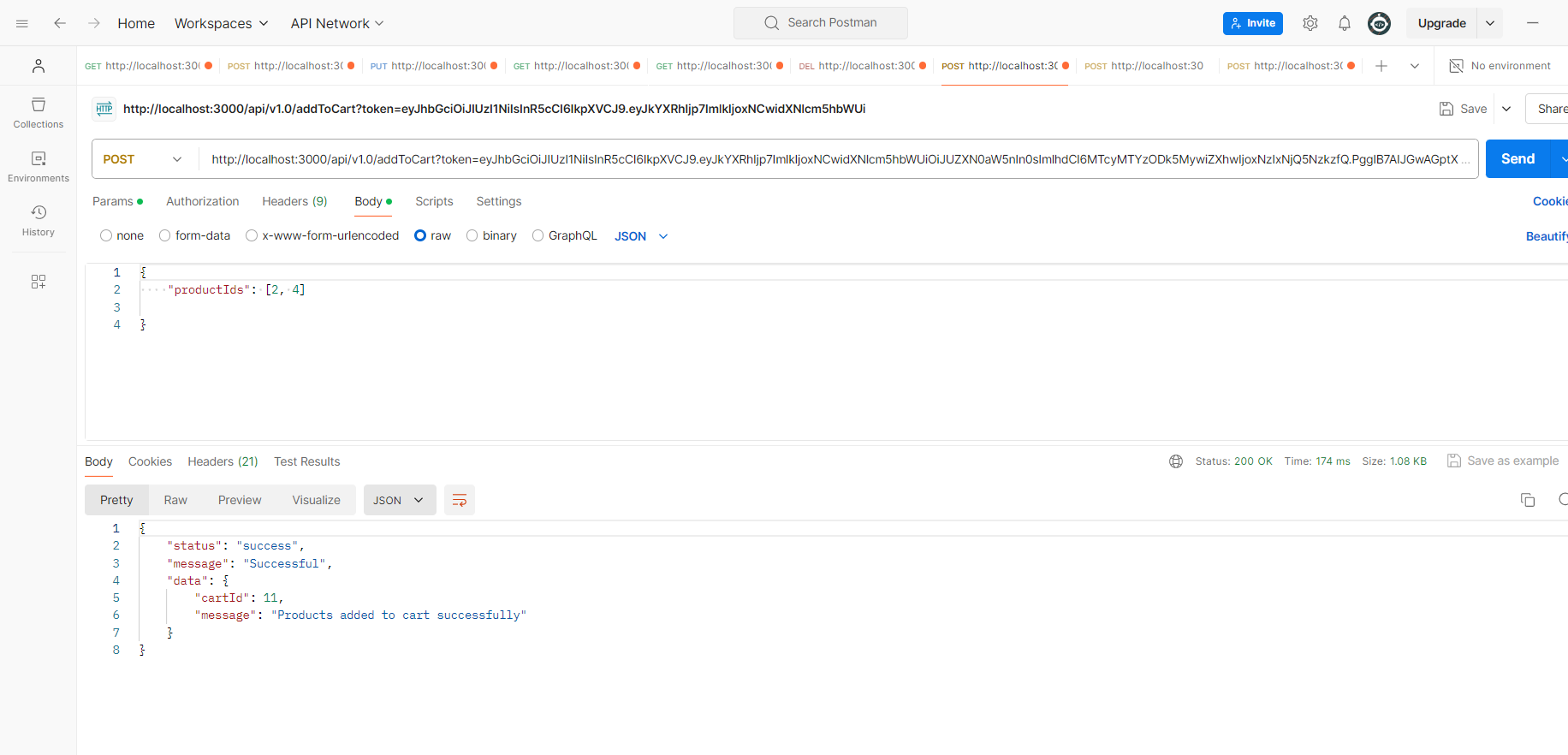
**Deleting Product API on Postman:-**

****

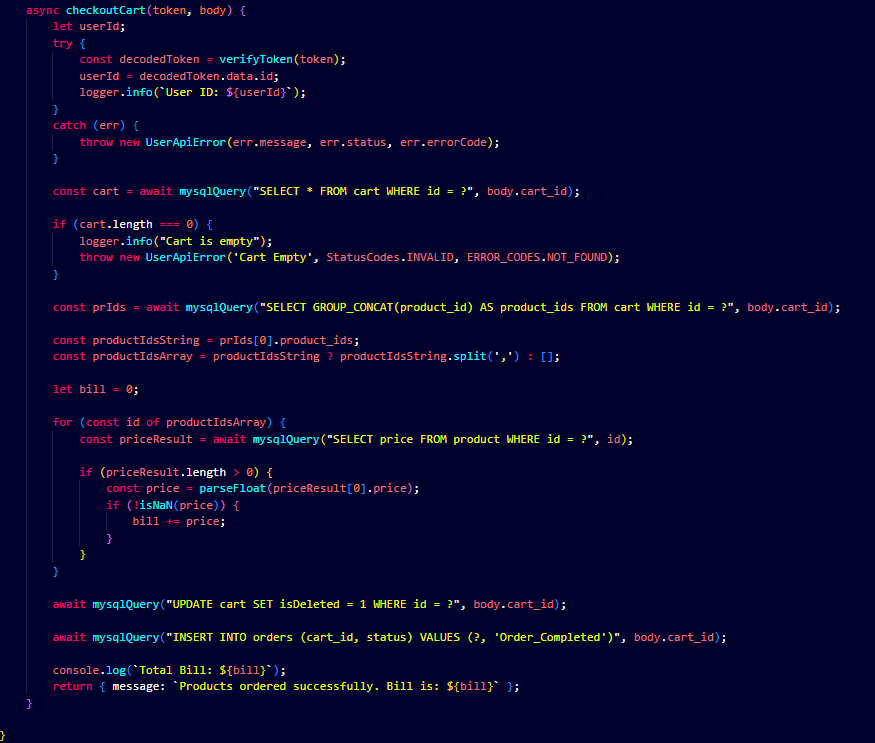
**Add to Cart Code:-**

****

**Add to Cart API on Postman:-**

****

**Checkout Code:-**

****

**Checkout API on Postman:-**

****

**Conclusion**

**The documentation covers JavaScript, Node.js, and MySQL thoroughly, from basics to advanced topics. It helps to write efficient code, build scalable server applications, manage databases effectively, and streamline project workflows. Topics include mastering JavaScript syntax, handling asynchronous operations, setting up servers, querying databases, and managing transactions. Clear instructions and best practices are provided for robust application development and efficient data management.**