Full Stack Development with AI
Lab 12.1 – Creating RESTful API Endpoints with Express.js

Lab Overview

In this lab, you will learn how to create RESTful API endpoints using Express.js. The endpoints are designed to support a client-side rendering web application created using frameworks such as React, Angular and Vue.

Similar to the lab exercises in Module 10, the web application will simulate the administrator panel of a typical B2C ecommerce shopping website. The use cases will allow an administrator to perform basic data manipulation operations on the product catalogue, i.e., CRUD or create, retrieve, update and delete.

You will be using the same synthetic product data as Lab 10.2.

Exercise 1 – Creating a new Express.js Application

1. Use the file explorer or terminal on your computer to create a new folder `app`.

2. Start Visual Studio Code (VS Code) and open the newly created `app` folder.

3. Create a new subfolder `db` within the `app` folder.

4. Start a terminal in VS Code.

5. Run the following commands to install the required node packages using `npm`:

```
npm install express
npm install sqlite3
npm install cors
```

Exercise 2 – Cloning the SQLite Database

For this lab, we will be reusing the “products.db” SQLite database that you have created in Lab 10.2.

Copy the file “products.db” into the `app\db` folder of your new Express.js application.

Exercise 3 – Creating New Endpoints

In this exercise, you will be creating a series of endpoints using the PUT, GET, POST and DELETE HTTP methods to implement the create, retrieve, update and delete use cases for product records.

1. Create a new file "app.js" in the root of the web application or the `app` folder itself.

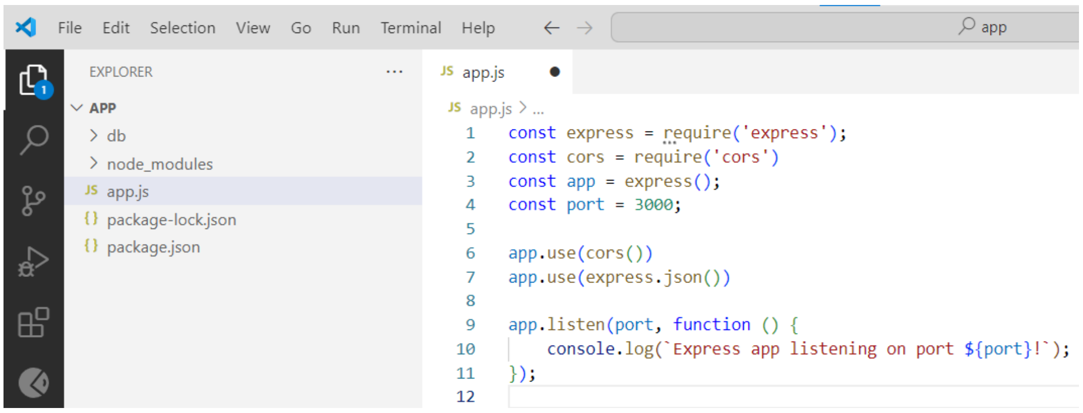
2. Paste the following code fragment representing the basic structure of an Express.js application into "app.js"

```
const express = require('express');
const cors = require('cors')
const app = express();
const port = 3000;

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

app.listen(port, function () {
    console.log(`Express app listening on port ${port}!`);
});
```

3. Your VS Code should resemble the following figure:



```
const express = require('express');
const cors = require('cors')
const app = express();
const port = 3000;

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

app.listen(port, function () {
    console.log(`Express app listening on port ${port}!`);
});
```

4. Between line 7 and 9, add a `GET` endpoint to the path `/product` that performs the following two tasks:

   a. Retrieves all the product records from the `products` table using a `SELECT` query.
   b. Return the product records as an array of objects in JSON format together with the HTTP 200 status code.

5. Add a `GET` endpoint to the path `/product` that performs the following four tasks:

    a. Accepts a path parameter for the product id of the product to be retrieved.
    b. Retrieves the product record from the `products` table using a `SELECT` query with an appropriate `WHERE` clause.
    c. Return the product record as an object in JSON format together with the HTTP 200 status code.
    d. If the required product does not exist, return an appropriate error message together with the HTTP 404 status code.

6. Add a `PUT` endpoint to the path `/product` that performs the following four tasks:

    a. Retrieves the data for a new product record from the body of the request in JSON object format.
    b. Insert the data for the new product record into the `products` table using an `INSERT` statement.
    c. Return an information message together with the HTTP 200 status code.
    d. If an error occurs, return an appropriate error message together with the HTTP 409 status code.

7. Add a `POST` endpoint to the path `/product` that performs the following five tasks:

    a. Accepts a path parameter for the product id of the product to be updated
    b. Retrieves the data for an existing product record from the body of the request in JSON object format.
    c. Update the data for the existing product record into the `products` table using an `UPDATE` statement with an appropriate `WHERE` clause.
    d. Return an information message together with the HTTP 200 status code.
    e. If an error occurs, return an appropriate error message together with the HTTP 400 status code.

8. Add a `DELETE` endpoint to the path `/product` that performs the following four tasks:

    a. Accepts a path parameter for the product id of the product to be deleted
    b. Delete the existing product record from the `products` table using a `DELETE` statement with an appropriate `WHERE` clause.
    c. Return an information message together with the HTTP 200 status code.
    d. If an error occurs, return an appropriate error message together with the HTTP 400 status code.

Exercise 4 – Running and Testing the Endpoints

Follow the instructions below to run the web application:

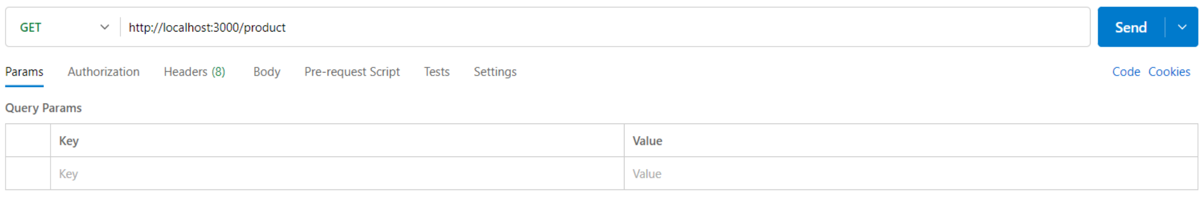
1. Start a new terminal in VS Code.

2. Run the following command to run the web application:

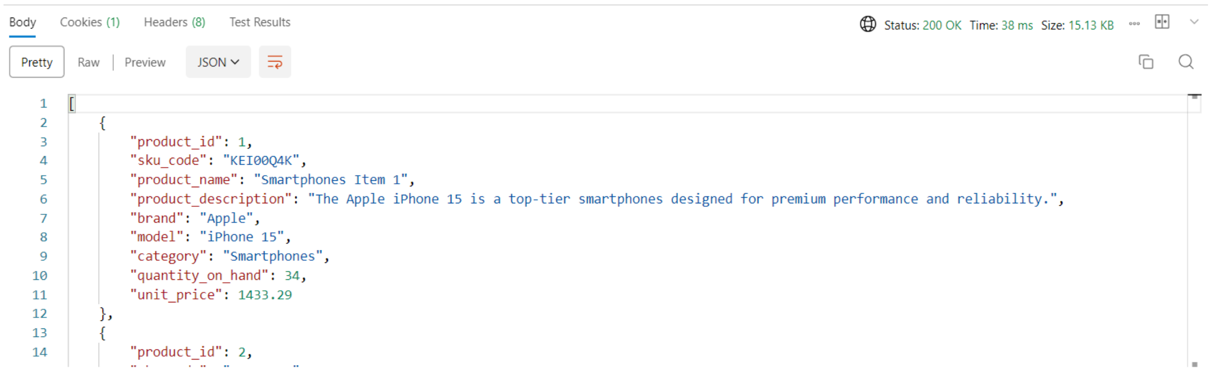
```
node app.js
```

3. Use Postman to test each of the endpoints.

4. `GET` endpoint to retrieve all product records:

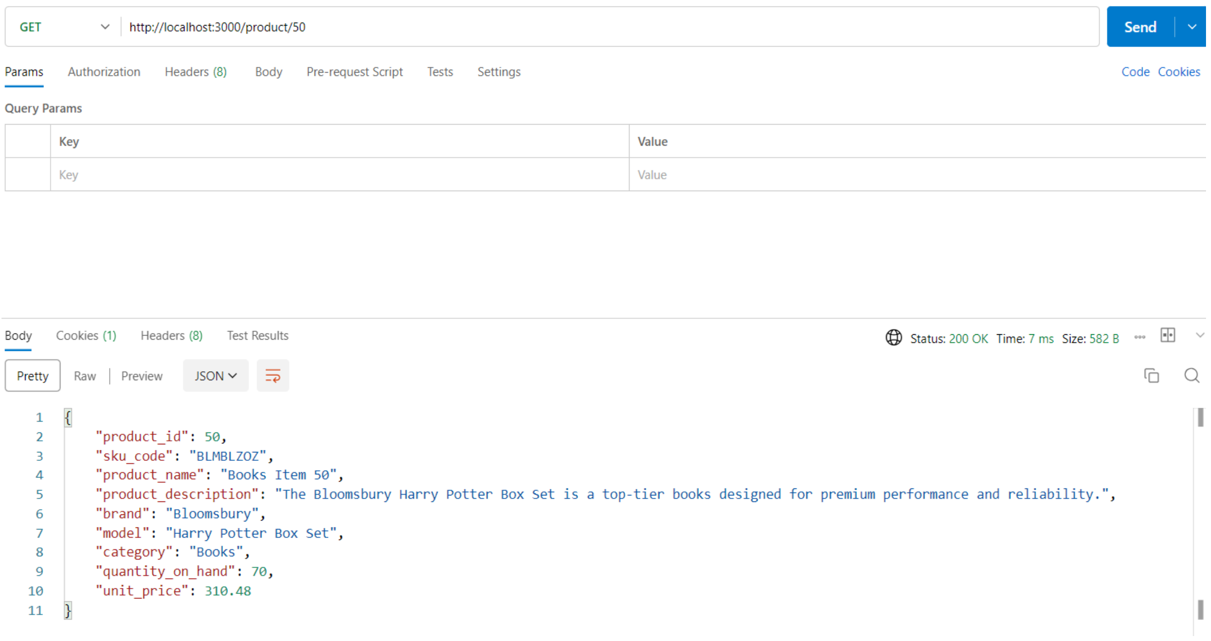


GET http://localhost:3000/product



```
[
    {
        "product_id": 1,
        "sku_code": "KEI00Q4K",
        "product_name": "Smartphones Item 1",
        "product_description": "The Apple iPhone 15 is a top-tier smartphones designed for premium performance and reliability.",
        "brand": "Apple",
        "model": "iPhone 15",
        "category": "Smartphones",
        "quantity_on_hand": 34,
        "unit_price": 1433.29
    },
    {
        "product_id": 2,
```

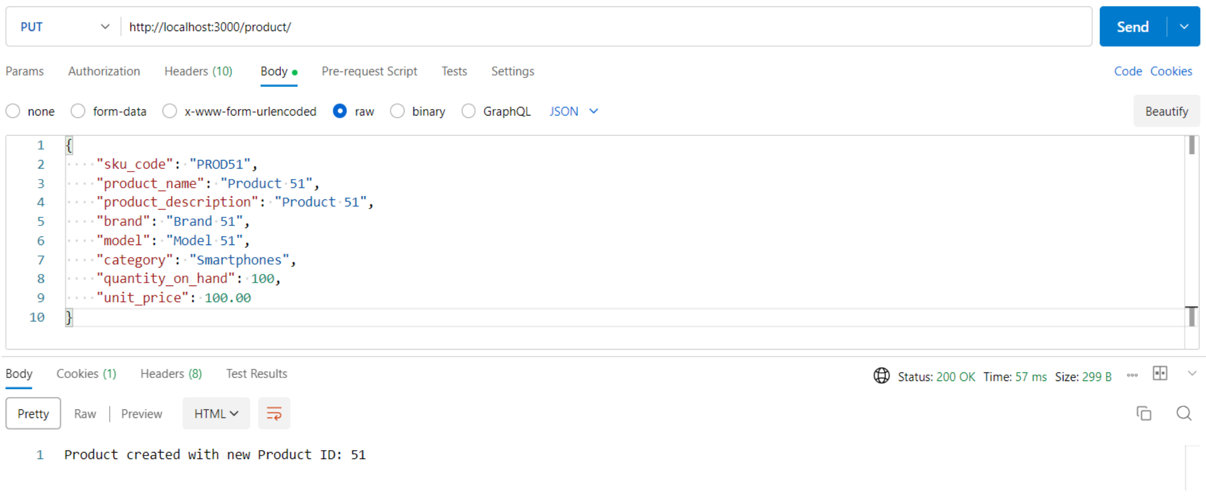
5. `GET` endpoint to retrieve one product record:



```
http://localhost:3000/product/50
```

```
{
    "product_id": 50,
    "sku_code": "BLMBLZOZ",
    "product_name": "Books Item 50",
    "product_description": "The Bloomsbury Harry Potter Box Set is a top-tier books designed for premium performance and reliability.",
    "brand": "Bloomsbury",
    "model": "Harry Potter Box Set",
    "category": "Books",
    "quantity_on_hand": 70,
    "unit_price": 310.48
}
```

6. `PUT` endpoint to create a new product record:



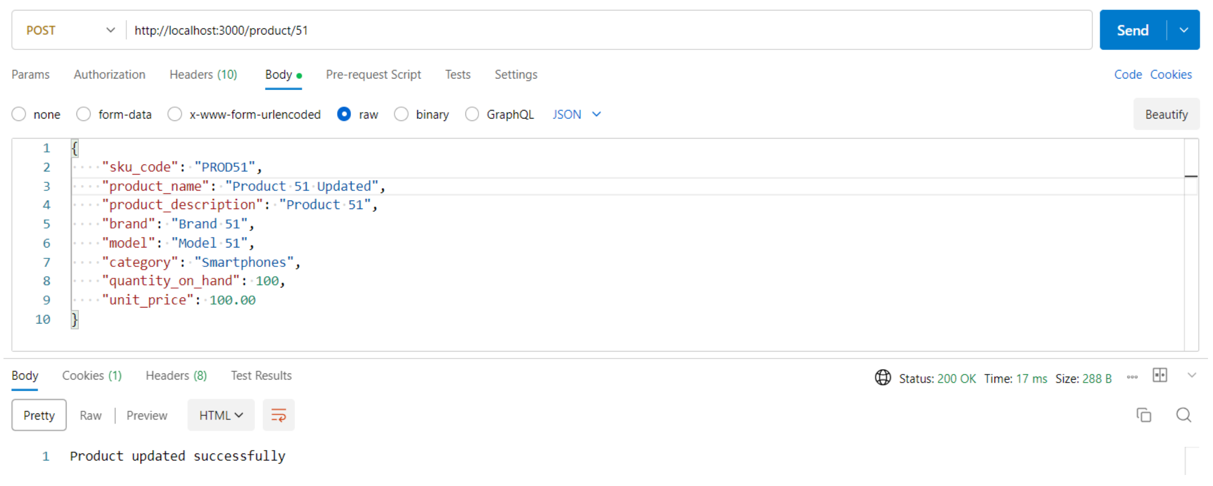
```
http://localhost:3000/product/
```

```
{
    "sku_code": "PROD51",
    "product_name": "Product 51",
    "product_description": "Product 51",
    "brand": "Brand 51",
    "model": "Model 51",
    "category": "Smartphones",
    "quantity_on_hand": 100,
    "unit_price": 100.00
}
```

```
Product created with new Product ID: 51
```

Use the `GET` endpoint to retrieve all products and verify that the new product record has been created successfully.

7. `POST` endpoint to update an existing product record:



```
http://localhost:3000/product/51
```

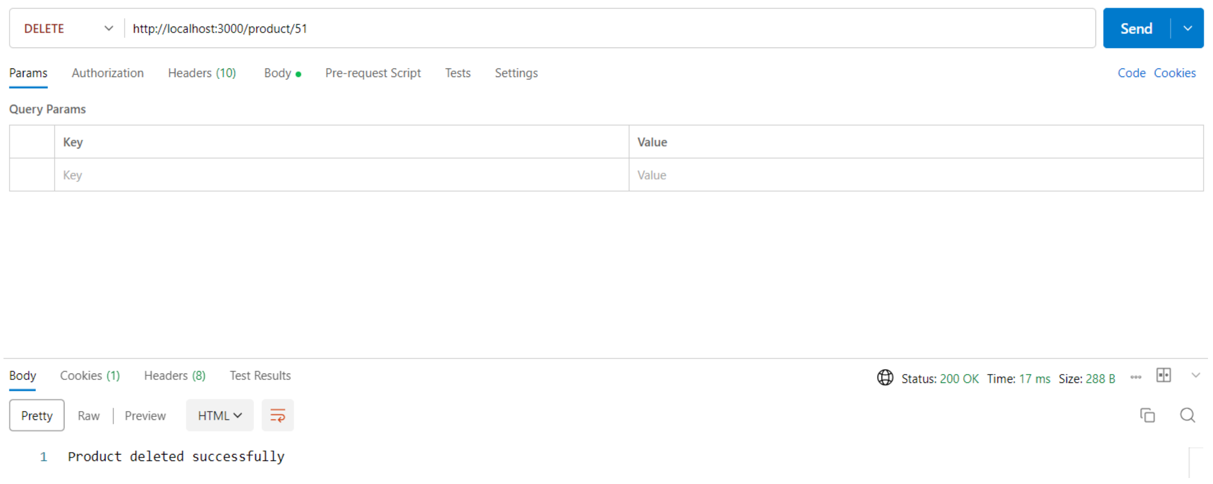
```
{
    "sku_code": "PROD51",
    "product_name": "Product 51 Updated",
    "product_description": "Product 51",
    "brand": "Brand 51",
    "model": "Model 51",
    "category": "Smartphones",
    "quantity_on_hand": 100,
    "unit_price": 100.00
}
```

Status: 200 OK Time: 17 ms Size: 288 B

```
Product updated successfully
```

Use the `GET` endpoint to retrieve all products and verify that the existing product record has been updated successfully.

8. `DELETE` endpoint to update an existing product record:



```
http://localhost:3000/product/51
```

	Key	Value
	Key	Value

Status: 200 OK Time: 17 ms Size: 288 B

```
Product deleted successfully
```

Use the `GET` endpoint to retrieve all products and verify that the existing product record has been deleted successfully.

9. Repeat the testing of the endpoints by introducing some error conditions such as retrieving a product record using a product id that does not exist.

-- End of Lab --