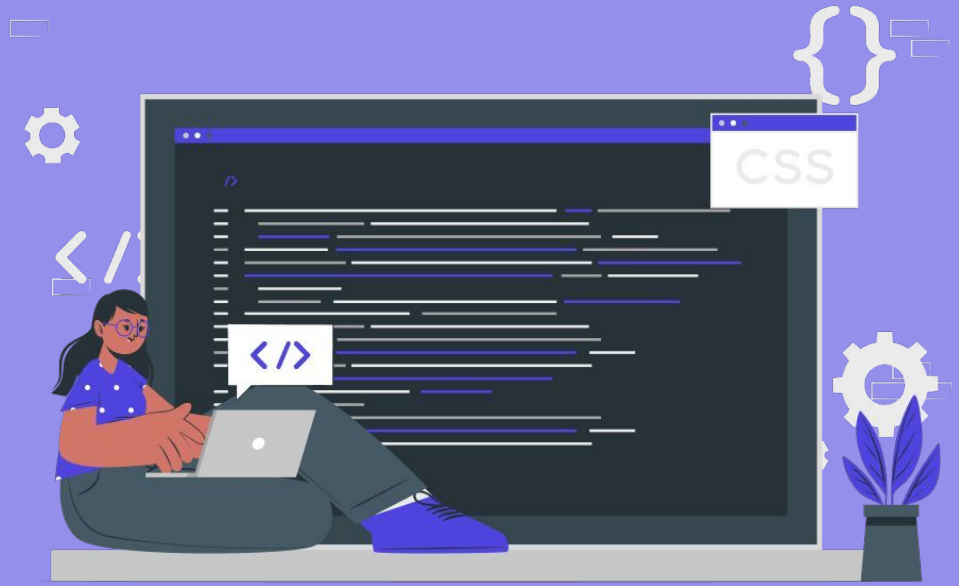
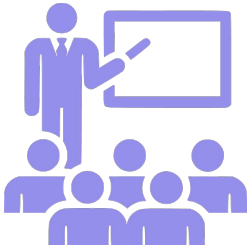
Create the REST APIs for Products



Relevel
by Unacademy

Session Agenda

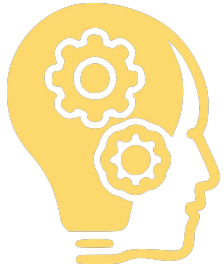
- We will learn about Products in reference to an eCommerce application
- We will create the schema for the Product resource
- Then we will implement the RESTful endpoints for creating CRUD operation on Products



Educator Introduction

Understanding the use cases around Products

- API to create a new Product
- API to get all the products
- API to get a product based on the id
- API to update a product
- API to delete the product

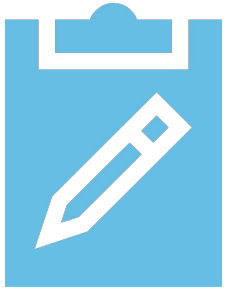


Implementation of the REST APIs

1. Define the Product Resource

Product attributes

- Name
- Description
- Cost



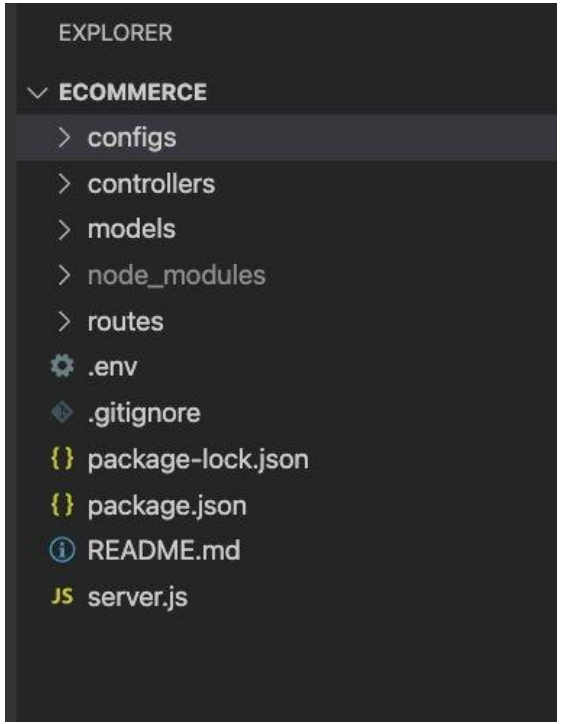
Category schema/table structure

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
description	varchar(255)	YES		NULL	
cost	int	NO		NULL	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

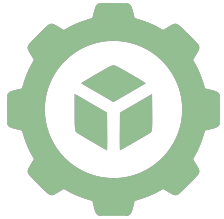
2. Create the project structure as below

Create the folders :

- controllers
- models
- routes



3. Create the model for Product



- product.model.js :

  https://github.com/Vishwa07dev/eCommerce/blob/session3/models/product.m odel.js

```
module.exports = (sequelize, Sequelize) => {
    const Product = sequelize.define("product", {
        id: {
            type: Sequelize.INTEGER,
            primaryKey: true,
            autoIncrement: true
        },
        name: {
            type: Sequelize.STRING,
            allowNull: false
        },
        description: {
            type: Sequelize.STRING
        },
        cost:{
            type: Sequelize.INTEGER,
            allowNull : false
        }

    },{
        tableName: 'products'
```

```
        /**
         * This helps you to provie a custom name to the table
         * If above is not provided, model name is converted into plural and
set as the table name
         *
         * If we want to just use the model name provided, we can provide
the below option :
         *
         * freezeTableName: true
         */
    });
    return Product;
}
```

4. Define the RESTful endpoints to be created

Ability to create a Product

REST URL : POST /ecomm/api/v1/products/

Request body :

```
{
        "name": "s17",
        "description": "Another samsung product", "cost":62344

}
```

Response body :

```
{
        "id": 1,
        "name": "s17",
        "description": "Another samsung product",
```

```
        "cost": 62344,
        "updatedAt": "2022-02-16T05:03:53.291Z",
        "createdAt": "2022-02-16T05:03:53.291Z"
}

Response code : 201
```

Ability to get all the products

REST URL: GET /ecomm/api/v1/products/

Response body :

```
[
    {
        "id": 1,
        "name": "s17",
        "description": "Another samsung product", "cost":
        62344,
        "createdAt": "2022-02-16T05:03:53.000Z",
        "updatedAt": "2022-02-16T05:03:53.000Z"
    },
    {
        "id": 2,
        "name": "i12",
```

```
            "createdAt": "2022-02-16T05:04:49.000Z",
            "updatedAt": "2022-02-16T05:04:49.000Z"
        }
]
```

Response code : 200

Ability to get all the products

REST URL: GET /ecomm/api/v1/products/

Response body :


```
[
    {
        "id": 1,
        "name": "s17",
        "description": "Another samsung product", "cost":
        62344,
        "createdAt": "2022-02-16T05:03:53.000Z",
        "updatedAt": "2022-02-16T05:03:53.000Z"
    },
    {
        "id": 2,
        "name": "i12",
        "description": "App product", "cost":
        92344,
```

```
            "createdAt": "2022-02-16T05:04:49.000Z",
            "updatedAt": "2022-02-16T05:04:49.000Z"
        }
]
```

Response code : 200

Ability to get the product based on id

REST URL: GET /ecomm/api/v1/products/1

Response body :


```
{
        "id": 1,
        "name": "s17",
        "description": "Another samsung product", "cost":
        62344,
        "createdAt": "2022-02-16T05:03:53.000Z",
        "updatedAt": "2022-02-16T05:03:53.000Z"
}
```

Ability to get all the products based on name

REST URL: GET /ecomm/api/v1/products?name=s17

Response body :


```
[
    {
        "id": 1,
        "name": "s17",
        "description": "Another samsung product", "cost":
        62344,
        "createdAt": "2022-02-16T05:03:53.000Z",
        "updatedAt": "2022-02-16T05:03:53.000Z"
    }
]
```


Response code : 200

Ability to update the product

REST URL: PUT /ecomm/api/v1/products/1

Request body :

```
{
        "name": "s17",
        "description": "Updated another samsung product",
        "cost": 82344

}
```

Response body :


```
{
        "id": 1,
        "name": "s17",
        "description": "Updated another samsung product", "cost": 62344,
        "createdAt": "2022-02-16T05:03:53.000Z",
        "updatedAt": "2022-02-16T05:07:38.000Z"
}
```


Response code : 200

Ability to delete a category

REST URL : DELETE /ecomm/api/v1/products/1

Response body :

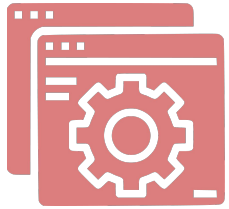
```
{
        "message": "Successfully deleted the product"
}
```

Response code : 200

5. Create the controller file for the Product resource

product.controller.js :

https://github.com/Vishwa07dev/eCommerce/blob/session3/controllers/product.controller.js



```
* This file contains the controller logic for the product resource.
* Everytime any CRUD request come for the Product, methods defined in this
* controller file will be executed.
*/

const { product } = require("../models");
const db = require("../models");
const Product = db.product;

/**
* Create and save a new Product
*/
exports.create = (req, res) => {
   /**
    * Validation of the request body
    */

   if (!req.body.name) {
       res.status(400).send({
           message: "Name of the product can't be empty !"
       })
       return;
   }
```

```
/**
 * Creation of the Product object to be stored in the DB
 */
const product = {
    name: req.body.name,
    description: req.body.description,
    cost : req.body.cost
};

/**
 * Storing the Product object in the DB
 */
Product.create(product).then(product => {
    console.log(`product name: [ ${product.name}] got inserted in DB`)
    res.status(201).send(product);
}).catch(err => {
    console.log(`Issue in inserting product name: [ ${product.name}].
```

```
Error message : ${err.message}`)
        res.status(500).send({
            message: "Some Internal error while storing the product!"
        })
    })
}

/**
* Get a list of all the products
*/
exports.findAll = (req, res) => {

    //Supporting the query param
    let productName = req.query.name;
    let promise ;
    if(productName){
        promise = Product.findAll({
            where : {
                name : productName
            }
        });
    }else{
        promise =  Product.findAll();
    }
```

```
    promise.then(products => {
        res.status(200).send(products);
    }).catch(err => {
        res.status(500).send({
            message: "Some Internal error while fetching all the products"
        })
    })
}

/**
* Get a product based on the product id
*/
exports.findOne = (req, res) => {
    const productId = req.params.id;

    Product.findByPk(productId).then(product => {
        res.status(200).send(product);
    }).catch(err => {
        res.status(500).send({
```

```
            message: "Some Internal error while fetching the product based
on the id"
        })
    })
}


/**
* Update an existing product
*/
exports.update = (req, res) => {

    /**
     * Validation of the request body
     */

    if (!req.body.name) {
        res.status(400).send({
            message: "Name of the product can't be empty !"
        })
        return;
    }
```

```
/**
 * Creation of the Product object to be stored in the DB
 */
const product = {
    name: req.body.name,
    description: req.body.description
};
const productId = req.params.id;

Product.update(product, {
    returning: true,
    where: { id: productId }
}).then(updatedProduct => {

    Product.findByPk(productId).then(product => {
        res.status(200).send(product);
    }).catch(err => {
        res.status(500).send({
```

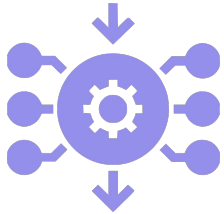
```
message: "Some Internal error while fetching the product based on the id"
            })
        })
    }).catch(err => {
        res.status(500).send({
            message: "Some Internal error while fetching the product based
on the id"
        })
    })
}

/**
* Delete an existing product based on the product name
*/
exports.delete = (req, res) => {
    const productId = req.params.id;

    Product.destroy({
        where: {
            id: productId
        }
```

```
    }).then(result => {
        res.status(200).send(
            {
            message: "Successfully deleted the product"
        }
        );
    }).catch(err => {
        res.status(500).send({
            message: "Some Internal error while deleting the product based
on the id"
        })
    })
}
```

6. Define the routes for the product

product.routes.js :

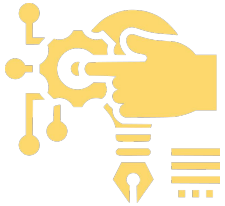
https://github.com/Vishwa07dev/eCommerce/blob/session3/routes/product.routes.js



```
/**
* This file will contain the routes logic for the Product resource
* and will export it.
*/

const productController = require("../controllers/product.controller")

module.exports = function(app){

    //Route for the POST request to create the product
    app.post("/ecomm/api/v1/products", productController.create);
    //Route for the GET request to fetch all the products
    app.get("/ecomm/api/v1/products", productController.findAll);
    //Route for the GET request to fetch a product based on the id
    app.get("/ecomm/api/v1/products/:id", productController.findOne);
    //Route for the PUT request to update a product based on the id
    app.put("/ecomm/api/v1/products/:id", productController.update);
    //Route for the DELETE request to delete a product based on the id
    app.delete("/ecomm/api/v1/products/:id", productController.delete);
}
```

7. Update the server.js file in the root folder to stitch all the modules

server.js :
https://github.com/Vishwa07dev/eCommerce/blob/session3/server.js



```
const express = require('express');
const serverConfig = require('./configs/server.config');
const bodyParser = require('body-parser');

// initiaizing express
const app = express();


/**
* Using the body-parser middleware
*
* Using for parsing the request.
* Parsing the request of the type json and convert that to object
*
* */
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
```

```
/**
* Initializing the database
*/
const db = require("./models");
const Category = db.category;

console.log(Category);
db.sequelize.sync({ force: true }).then(() => {
   console.log('tables dropped and recreated');
   init();
})

function init() {

   //Initializing few Categories
```

```
var categories = [
        {
            name: "Electronics",
            description: "This category will contain all the electronic
products"
        },
        {
            name: "KitchenItems",
            description: "This category will contain all the Kitchen related
products"
        }
    ];

    Category.bulkCreate(categories).then(() => {
        console.log("Categories table is initialized");
    }).catch(err => {
        console.log("Error while initializing ategories table");
    })

}
```

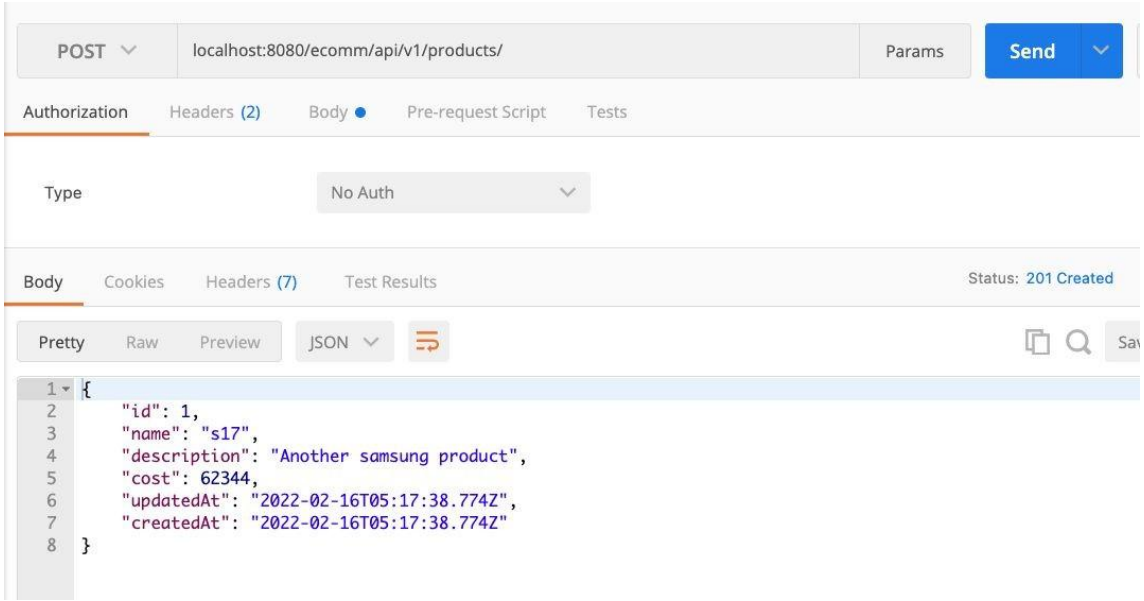
```
/**
* Importing the routes and using it
*/
require('./routes/category.routes')(app);
require('./routes/product.routes')(app);


//Starting the server
app.listen(serverConfig.PORT, () => {
    console.log(`Application started on the port no :
${serverConfig.PORT}`);
})
```

8. Start the node.js server using

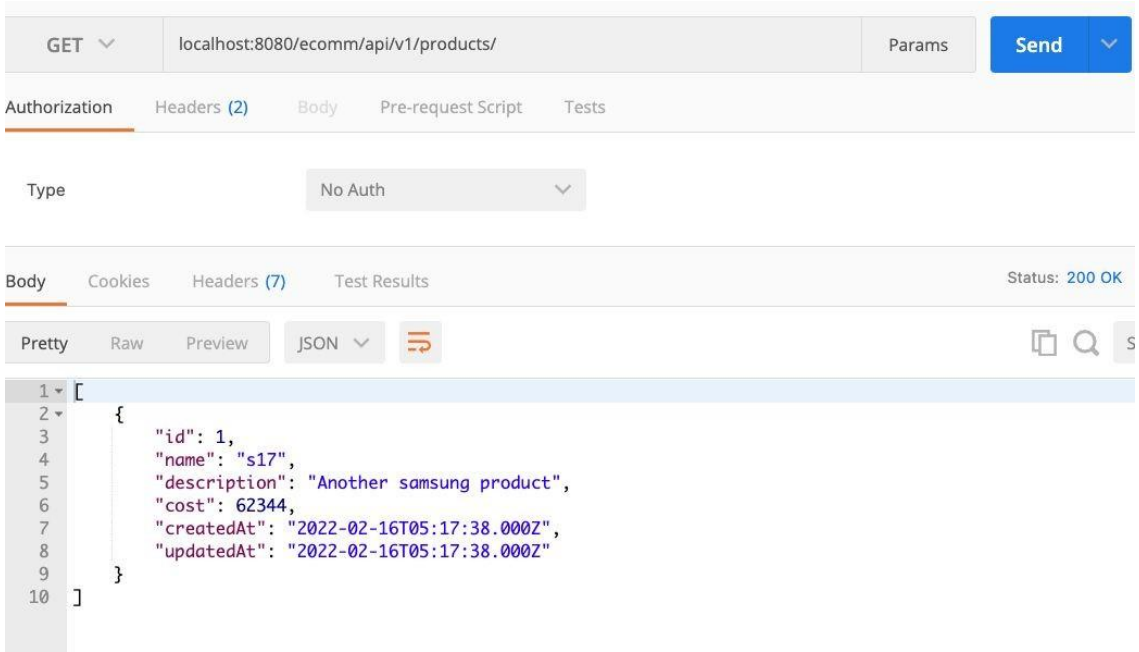
```
node server.js
```

9. Testing the APIs using Postman

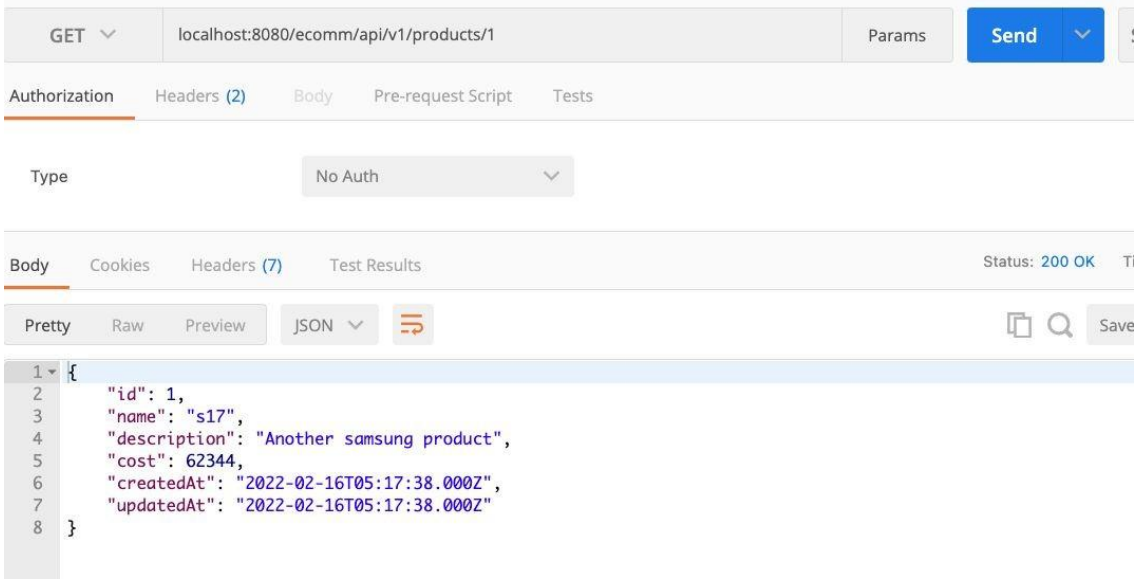
POST API



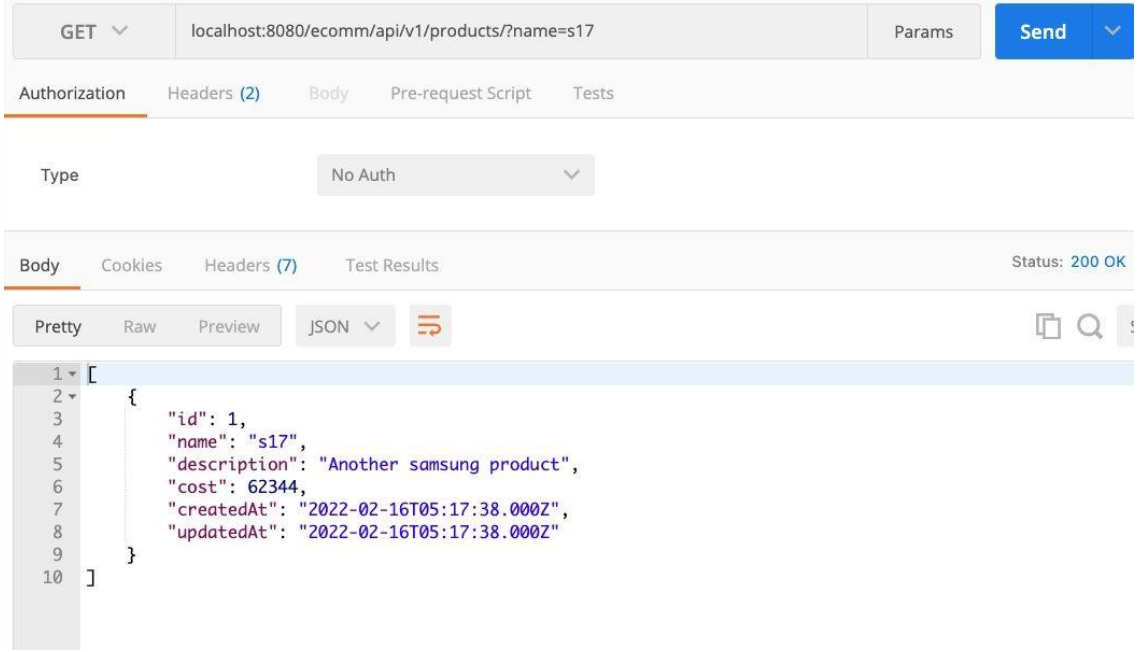
GET ALL API


GET
localhost:8080/ecomm/api/v1/products/
Params
Send
Authorization
Headers (2)
Body
Pre-request Script
Tests
Type
No Auth
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Pretty
Raw
Preview
JSON
[
{
"id": 1,
"name": "s17",
"description": "Another samsung product",
"cost": 62344,
"createdAt": "2022-02-16T05:17:38.000Z",
"updatedAt": "2022-02-16T05:17:38.000Z"
}
]

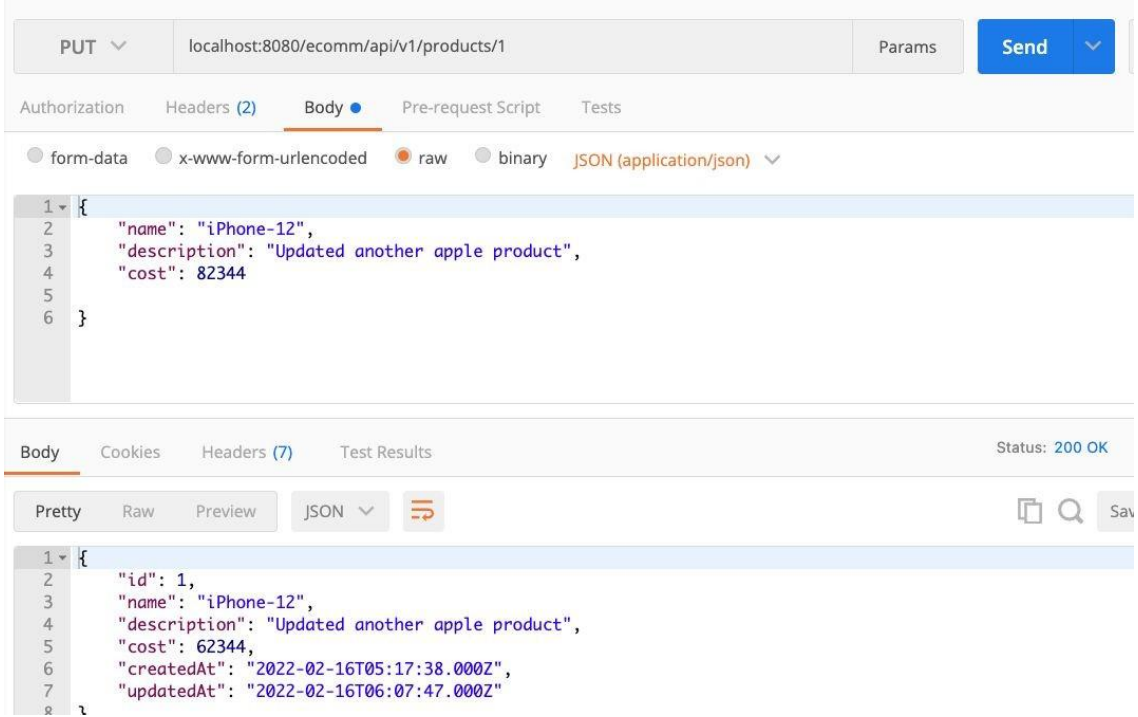
GET Product based on Id



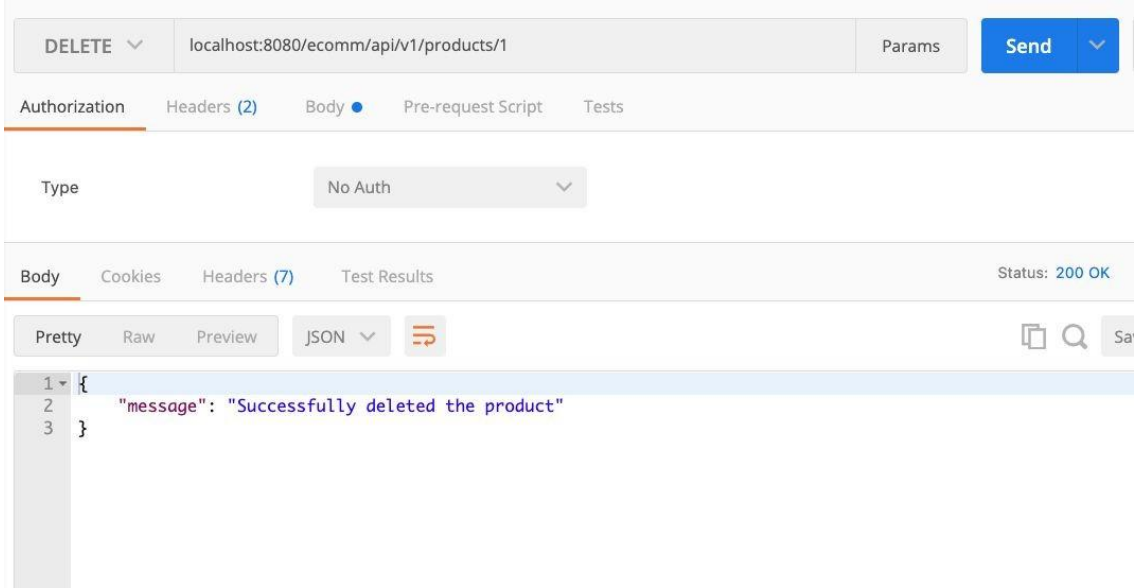
GET product based on name



Update Product



Delete Product based on the id



MCQs

1. Which of the following code prints memory usage?

A. console.log(process.memoryUsage());

B. console.log('Current version: ' + process.memory());

C. console.log('Current version: ' + process.getMemory());

D. None of the above.

2. What is Callback?

A. Callback is an asynchronous equivalent for a function.

B. Callback is a technique in which a method call back the caller method.

C. Both of the above.

D. None of the above.

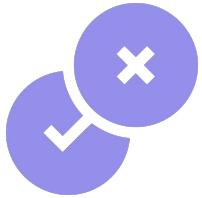
3. **Which extension is used to save NodeJs files?**

A. .js

B. .txt

C. .node

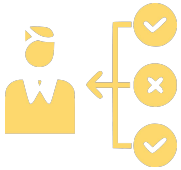
D. .java



4. **Is node js multithreaded?**

A. Yes

B. No



5. Node.js application can access which of the following databases?

A. NoSQL databases

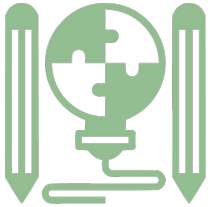
B. Relational databases

C. All of the above

D. None of the above

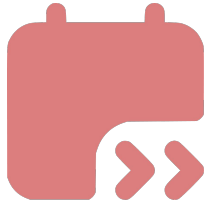
Practice Problem:

- In our last mini project on books api, let us now use mysql to fetch data from it.



Next session

- We will get to know relation between Category and Product
- Validation of the request body using custom middlewares
- Each product should definitely have the category id associated with it
- Get All the products in a Category : GET /eCom/v1/api/categories/:categoryId/products
- Get All the products whose value is greater than X
- Get All the products whose value is less than X
- Get All the products whose value is more than X



THANK YOU