Create the APIs for categories

Relevel
by Unacademy

Class Agenda

- We are going to discuss the category in reference to an eCommerce application
- We will create the schema for the Category resource
- And then we will implement the RESTful endpoints for creating CRUD operation on Category

Educator Introduction

Understanding the use cases around Category

- API to create a new Category
- API to get all the categories
- API to get a category based on the id
- API to update a category
- API to delete the category

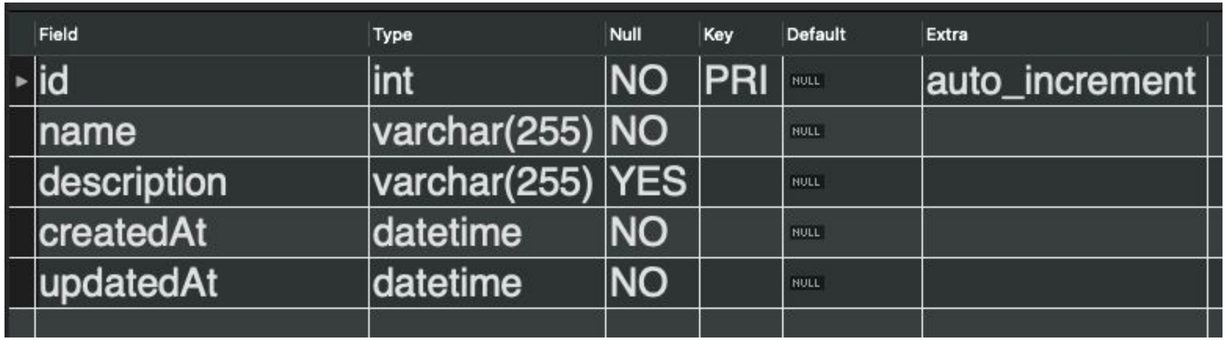
Implementation of the REST APIs

1. Define the Category Resource

Category attributes

- Name
- Description

Category schema/table structure

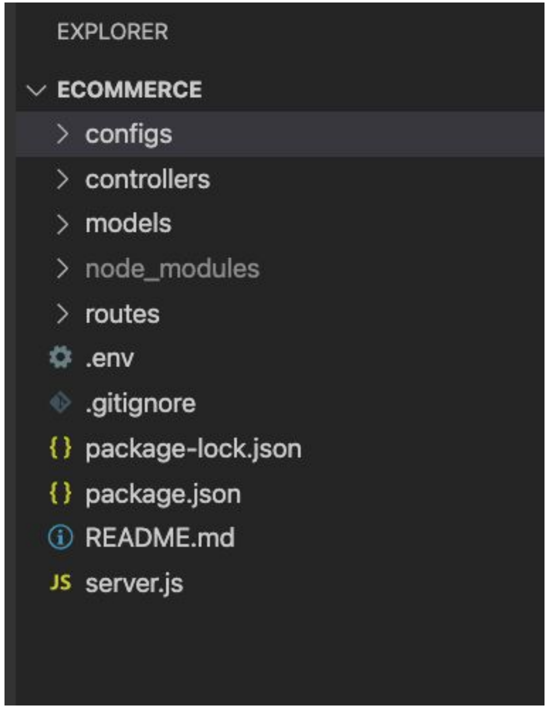


Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
description	varchar(255)	YES		NULL	
createdAt	datetime	NO		NULL	
updatedAt	datetime	NO		NULL	

2. Create the project structure as below

Create the folders :

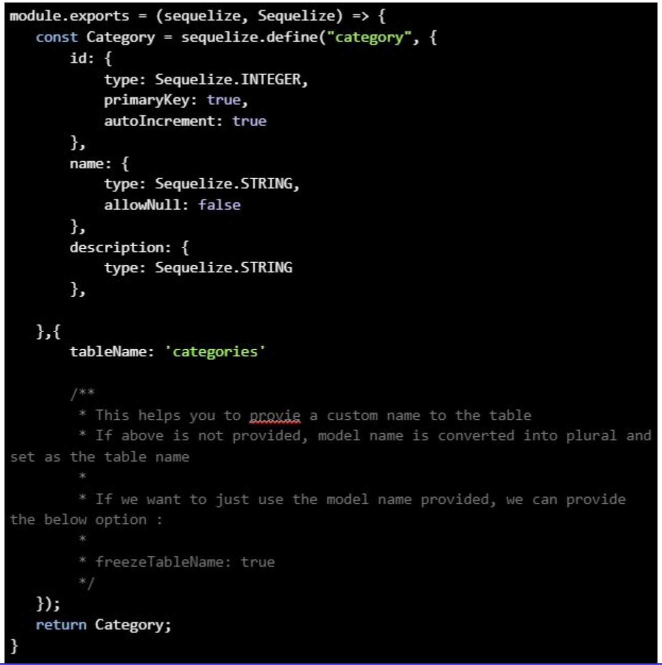
- controllers
- models
- routes



```
EXPLORER
∨ ECOMMERCE
  > configs
  > controllers
  > models
  > node_modules
  > routes
  .env
  .gitignore
  package-lock.json
  package.json
  README.md
  server.js
```

3. Create the model for category

- category.model.js : https://github.com/Vishwa07dev/eCommerce/tree/session2



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

    },{
        tableName: 'categories'

        /**
         * 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 Category;
}
```

4. Define the RESTful endpoints to be created

Ability to create a Category

REST URL :

POST

/ecomm/api/v1/categories/

```
Request body :

{
        "name": "Head Gears",
        "description": "This category will contain all the head gear
products"

}
```

```
{
    "id": 3,
    "name": "Head Gears",
    "description": "This category will contain all the head gear
products",
    "updatedAt": "2022-02-13T09:45:30.851Z",
    "createdAt": "2022-02-13T09:45:30.851Z"
}
Response code : 201
```

4. Define the RESTful endpoints to be created

Ability to get all the categories

REST URL:

GET /ecomm/api/v1/categories/

```
Response body :

[
    {
        "id": 1,
        "name": "Electronics",
        "description": "This category will contain all the electronic
products",
        "createdAt": "2022-02-13T09:45:26.000Z",
        "updatedAt": "2022-02-13T09:45:26.000Z"
    },
    {
        "id": 2,
        "name": "KitchenItems",
        "description": "This category will contain all the Kitchen
related products",
        "createdAt": "2022-02-13T09:45:26.000Z",
        "updatedAt": "2022-02-13T09:45:26.000Z"
    },
    {
        "id": 3,
        "name": "Head Gears",
        "description": "This category will contain all the head gear
products",
```

4. Define the RESTful endpoints to be created

Ability to get all the categories based on id

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

```
Response body :
{
    "id": 1,
    "name": "Electronics",
    "description": "This category will contain all the electronic
products",
    "createdAt": "2022-02-13T09:45:26.000Z",
    "updatedAt": "2022-02-13T09:45:26.000Z"
}
Response code : 200
```

4. Define the RESTful endpoints to be created

Ability to get all the categories based on name

REST URL: GET /ecomm/api/v1/categories?name=Electronics

```
Response body :
{
    "id": 1,
    "name": "Electronics",
    "description": "This category will contain all the electronic
products",
    "createdAt": "2022-02-13T09:45:26.000Z",
    "updatedAt": "2022-02-13T09:45:26.000Z"
}
Response code : 200
```

4. Define the RESTful endpoints to be created

Ability to update the category

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

```
Request body :
{
    "name": "KitchenItems",
    "description": "This category will contain all the Updated v2 Kitchen
related products"

}


Response body :
{
```

```
{
    "id": 2,
    "name": "KitchenItems",
    "description": "This category will contain all the Updated v2 Kitchen
related products",
    "createdAt": "2022-02-13T09:45:26.000Z",
    "updatedAt": "2022-02-13T09:49:44.000Z"
}
Response code : 200
```

4. Define the RESTful endpoints to be created

Ability to delete a category

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

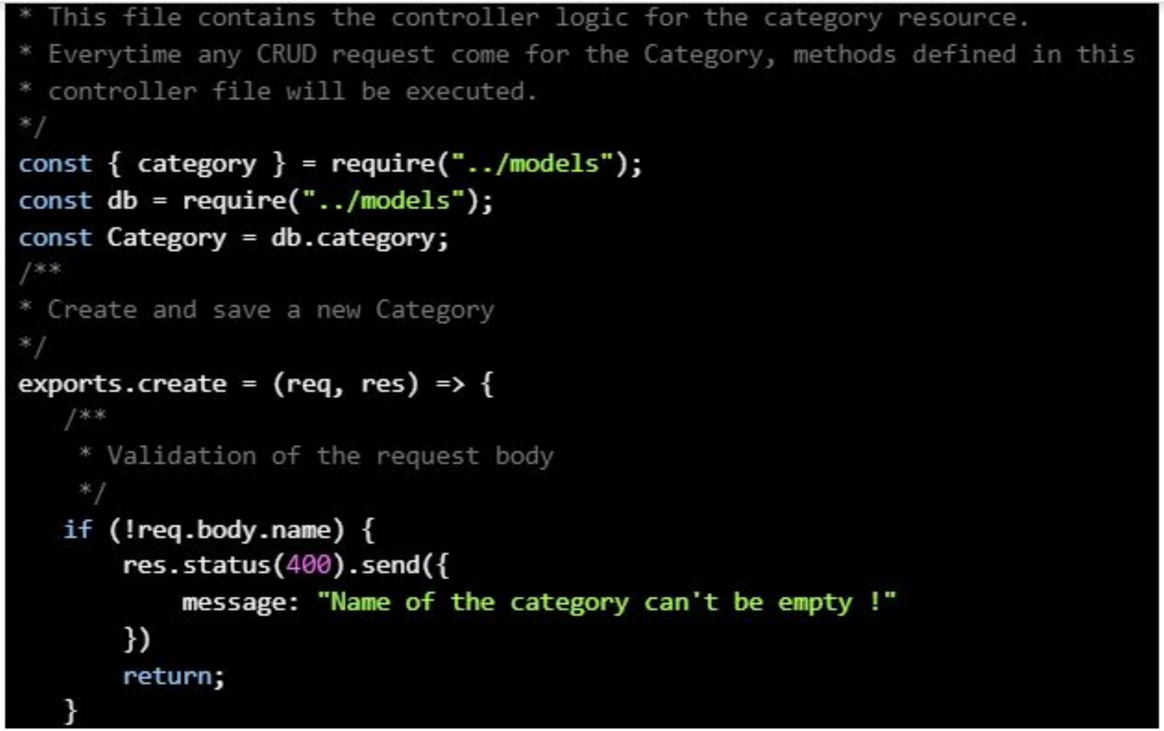
```
Response body :
{
    "message": "Successfully deleted the category"
}
Response code : 200
```

5. Create the controller file for the Category resource

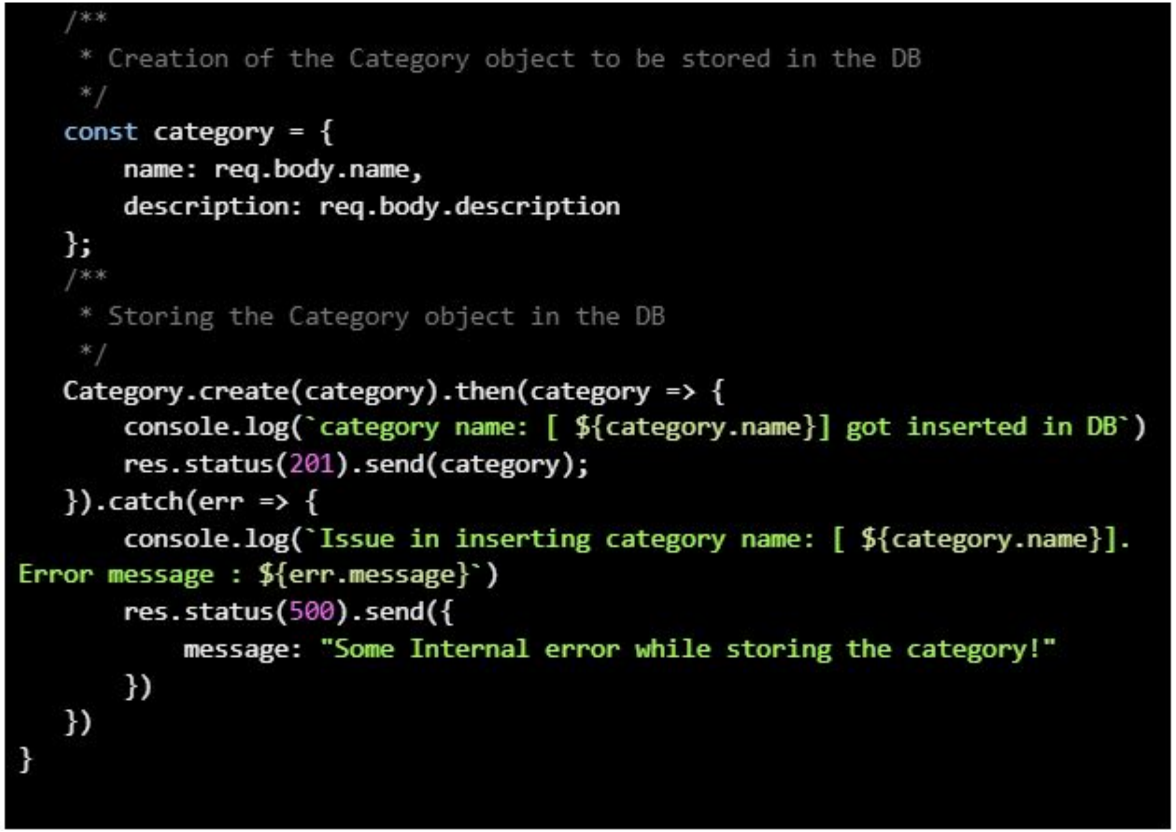
category.controller.js :

https://github.com/Vishwa07dev/eCommerce/blob/session2/controllers/category.controller.js

Function to create and save a new Category

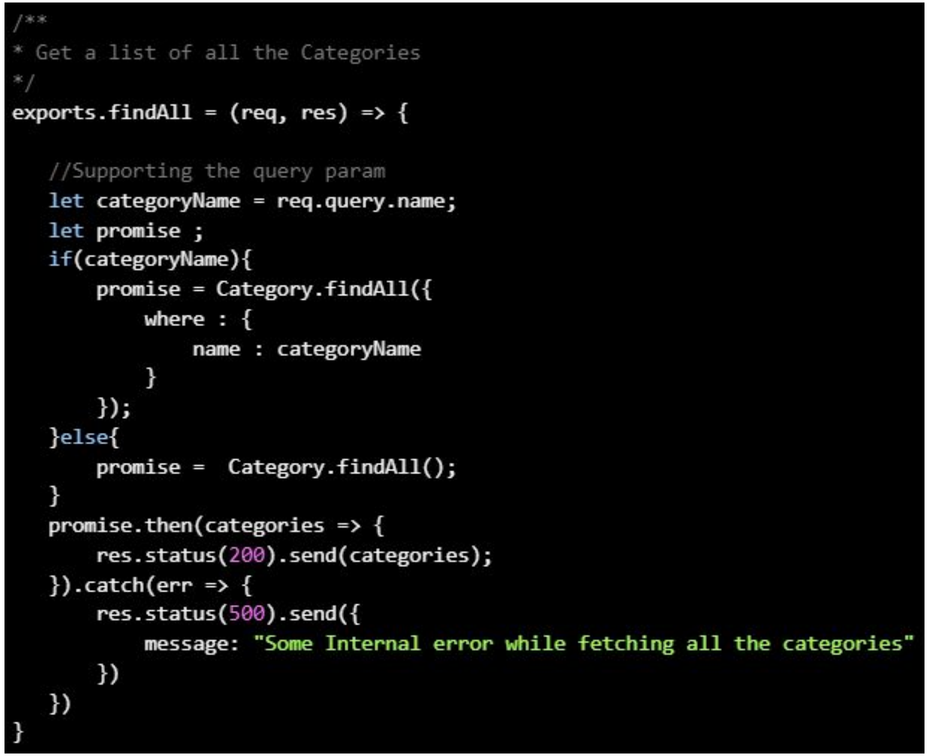


```
* This file contains the controller logic for the category resource.
* Everytime any CRUD request come for the Category, methods defined in this
* controller file will be executed.
*/
const { category } = require("../models");
const db = require("../models");
const Category = db.category;
/**
* Create and save a new Category
*/
exports.create = (req, res) => {
   /**
    * Validation of the request body
    */
   if (!req.body.name) {
       res.status(400).send({
           message: "Name of the category can't be empty !"
       })
       return;
   }
```



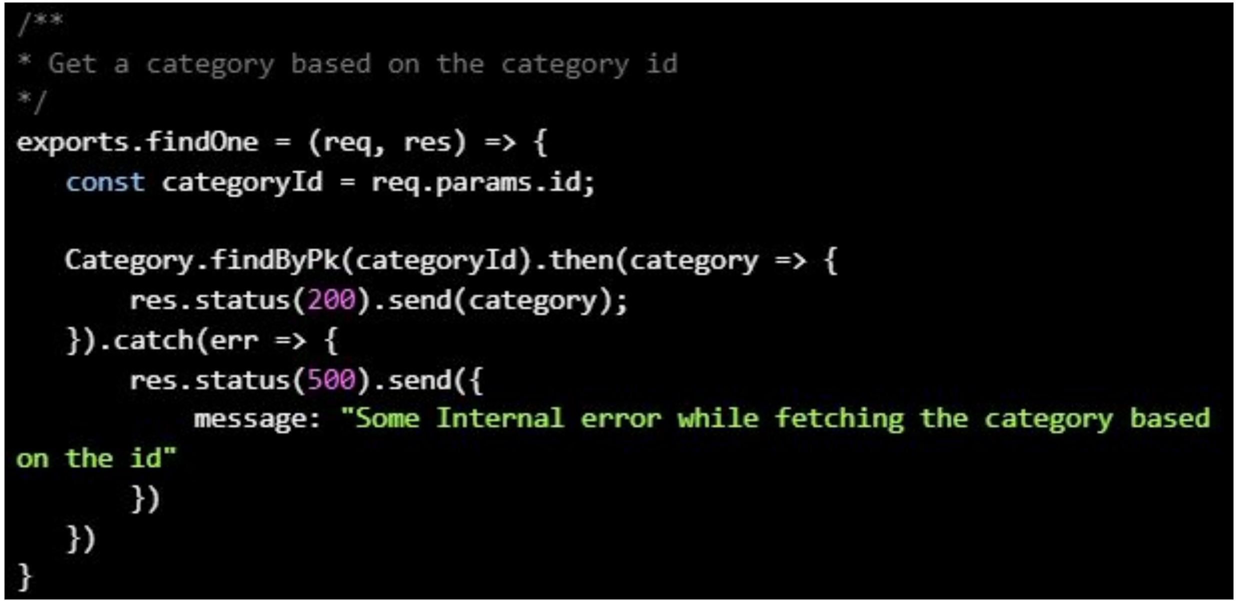
```
    /**
     * Creation of the Category object to be stored in the DB
     */
    const category = {
        name: req.body.name,
        description: req.body.description
    };
    /**
     * Storing the Category object in the DB
     */
    Category.create(category).then(category => {
        console.log(`category name: [ ${category.name}] got inserted in DB`)
        res.status(201).send(category);
    }).catch(err => {
        console.log(`Issue in inserting category name: [ ${category.name}].
Error message : ${err.message}`)
        res.status(500).send({
            message: "Some Internal error while storing the category!"
        })
    })
}
```

Function to get a list of all the Categories



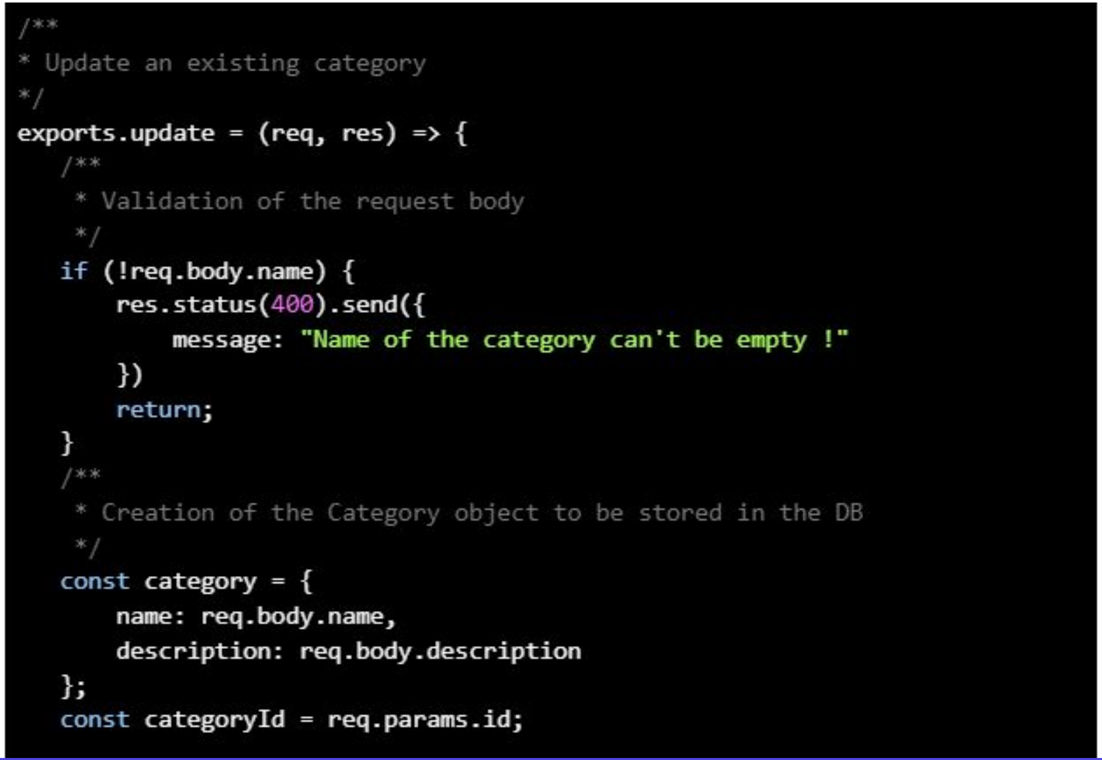
```
/**
* Get a list of all the Categories
*/
exports.findAll = (req, res) => {

   //Supporting the query param
   let categoryName = req.query.name;
   let promise ;
   if(categoryName){
       promise = Category.findAll({
           where : {
               name : categoryName
           }
       });
   }else{
       promise =  Category.findAll();
   }
   promise.then(categories => {
       res.status(200).send(categories);
   }).catch(err => {
       res.status(500).send({
           message: "Some Internal error while fetching all the categories"
       })
   })
}
```

Function to get a category based on the category id

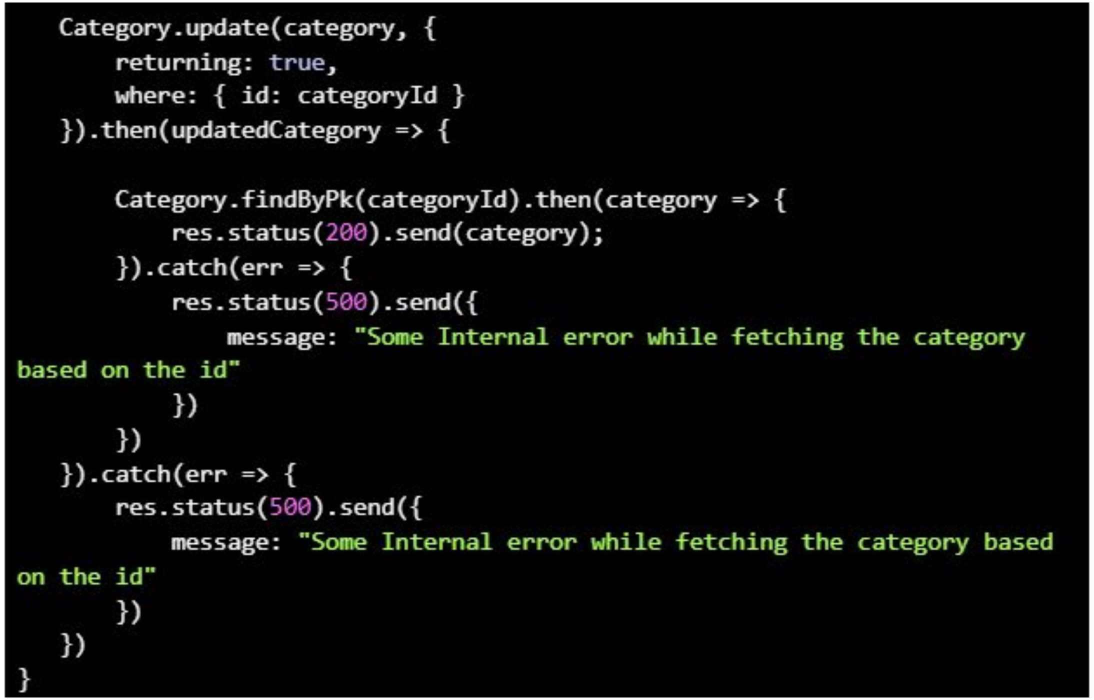


```
/**
* Get a category based on the category id
*/
exports.findOne = (req, res) => {
   const categoryId = req.params.id;

   Category.findByPk(categoryId).then(category => {
       res.status(200).send(category);
   }).catch(err => {
       res.status(500).send({
           message: "Some Internal error while fetching the category based
on the id"
       })
   })
}
```

Function to update an existing category

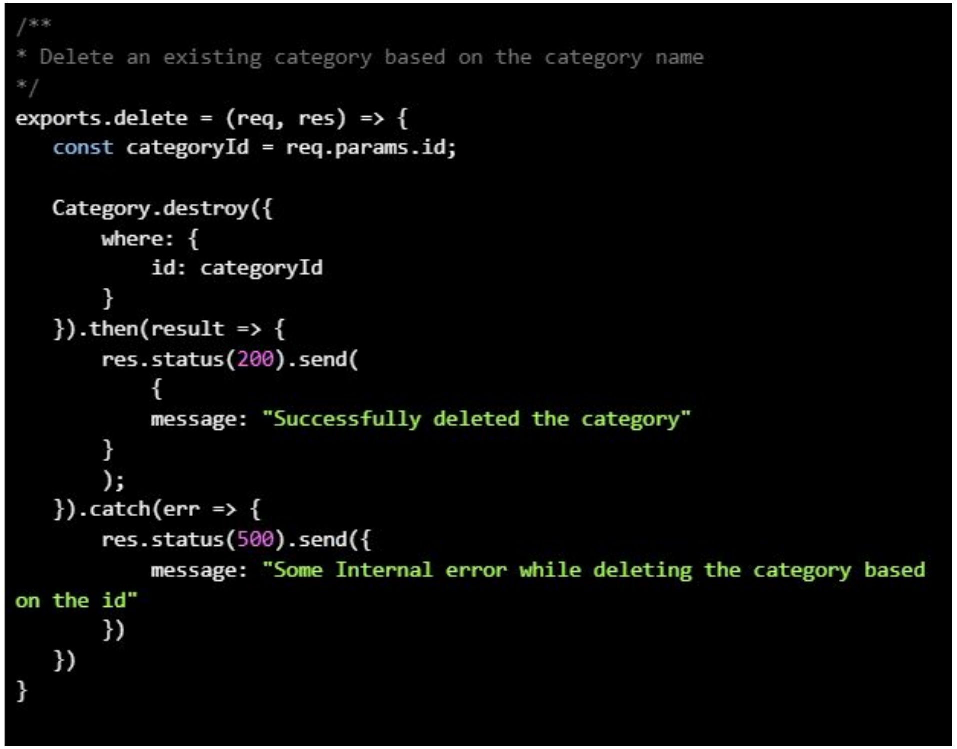


```
/**
 * Update an existing category
 */
exports.update = (req, res) => {
    /**
     * Validation of the request body
     */
    if (!req.body.name) {
        res.status(400).send({
            message: "Name of the category can't be empty !"
        })
        return;
    }
    /**
     * Creation of the Category object to be stored in the DB
     */
    const category = {
        name: req.body.name,
        description: req.body.description
    };
    const categoryId = req.params.id;
```



```
    Category.update(category, {
        returning: true,
        where: { id: categoryId }
    }).then(updatedCategory => {

        Category.findByPk(categoryId).then(category => {
            res.status(200).send(category);
        }).catch(err => {
            res.status(500).send({
                message: "Some Internal error while fetching the category
based on the id"
            })
        })
    }).catch(err => {
        res.status(500).send({
            message: "Some Internal error while fetching the category based
on the id"
        })
    })
}
```

Function to delete an existing category by it’s id



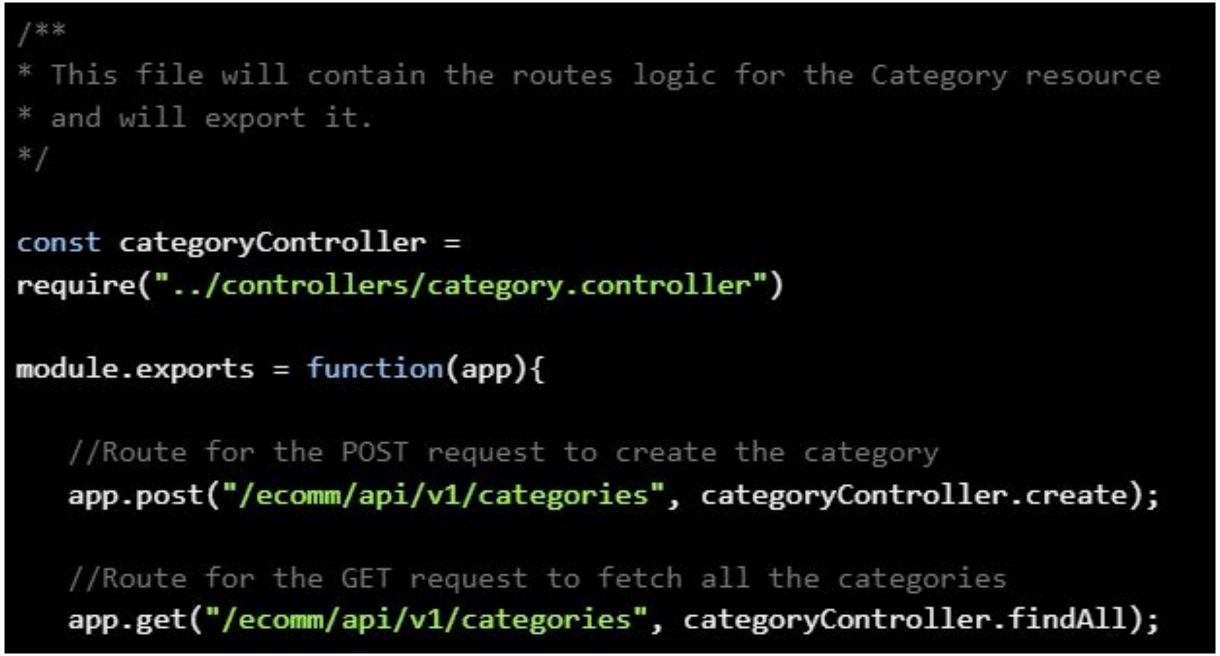
```
/**
* Delete an existing category based on the category name
*/
exports.delete = (req, res) => {
   const categoryId = req.params.id;

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

6. Define the routes for the category

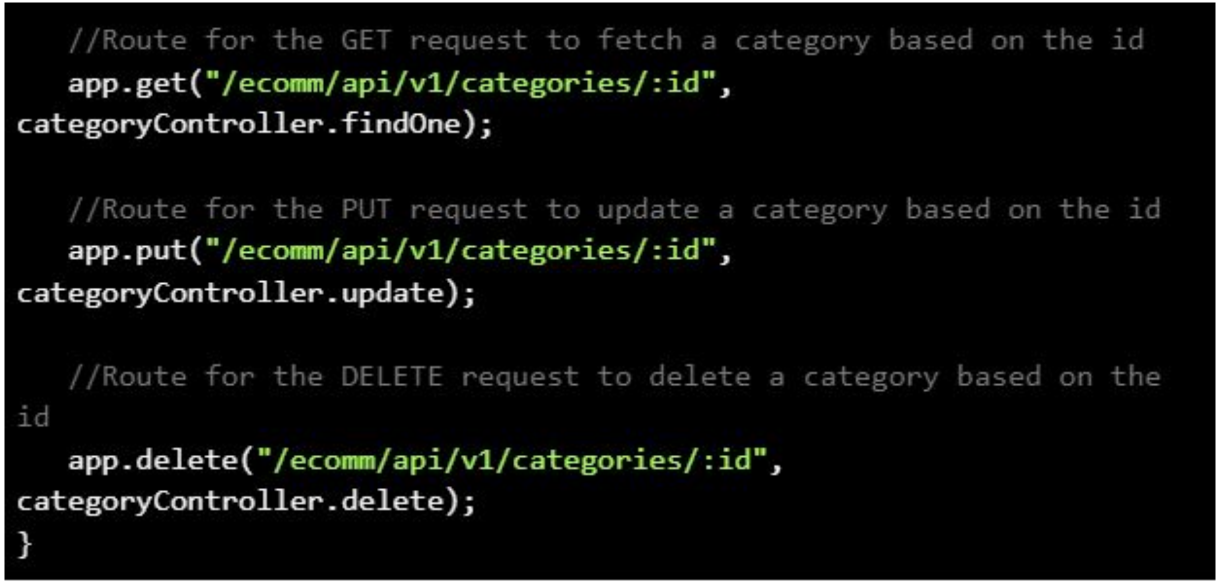
Category.routes.js :

https://github.com/Vishwa07dev/eCommerce/blob/session2/routes/category.routes.js

Category controller



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

const categoryController =
require("../controllers/category.controller")

module.exports = function(app){

   //Route for the POST request to create the category
   app.post("/ecomm/api/v1/categories", categoryController.create);

   //Route for the GET request to fetch all the categories
   app.get("/ecomm/api/v1/categories", categoryController.findAll);
```



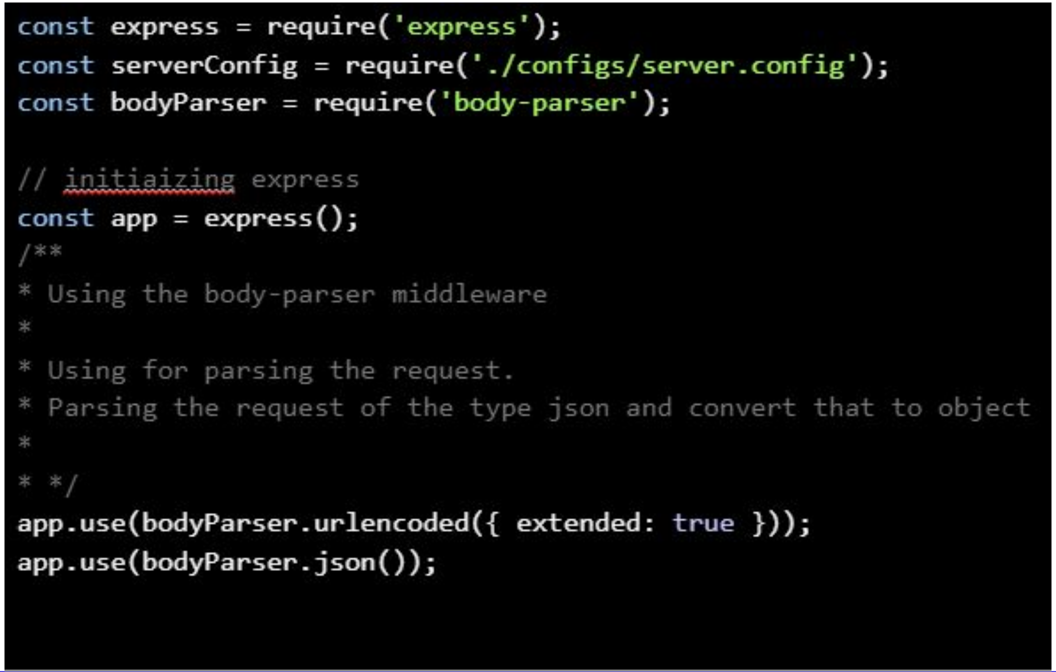
```
    //Route for the GET request to fetch a category based on the id
    app.get("/ecomm/api/v1/categories/:id",
categoryController.findOne);

    //Route for the PUT request to update a category based on the id
    app.put("/ecomm/api/v1/categories/:id",
categoryController.update);

    //Route for the DELETE request to delete a category based on the
id
    app.delete("/ecomm/api/v1/categories/:id",
categoryController.delete);
}
```

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

server.js : https://github.com/Vishwa07dev/eCommerce/blob/session2/server.js

Server.js

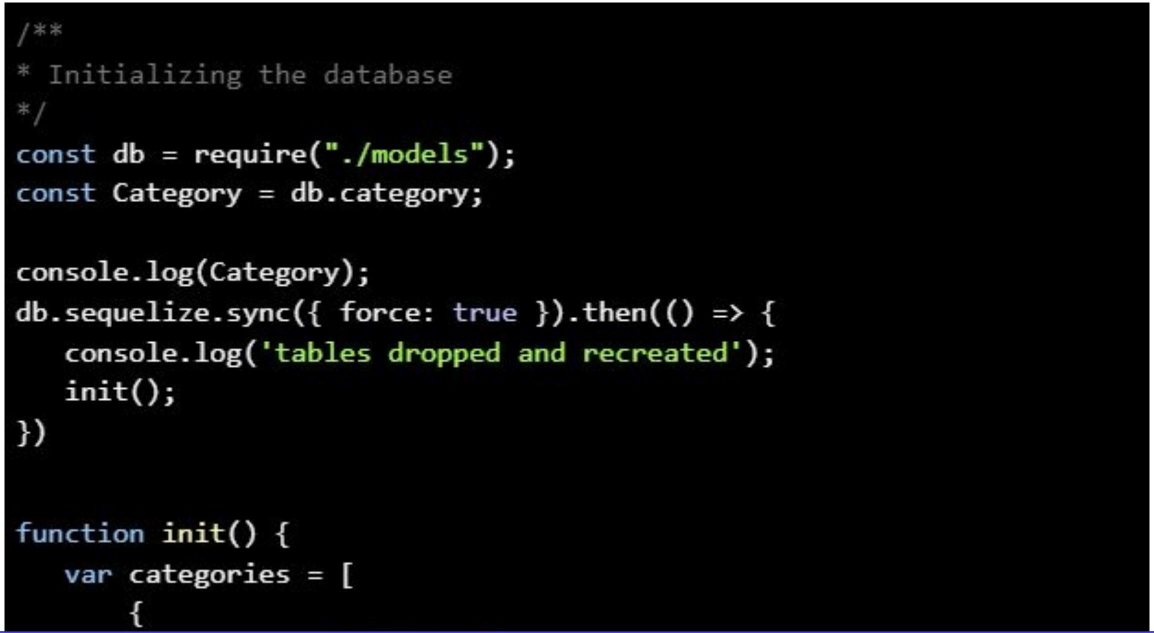
- Initializing express and adding bodyparser to read request body data



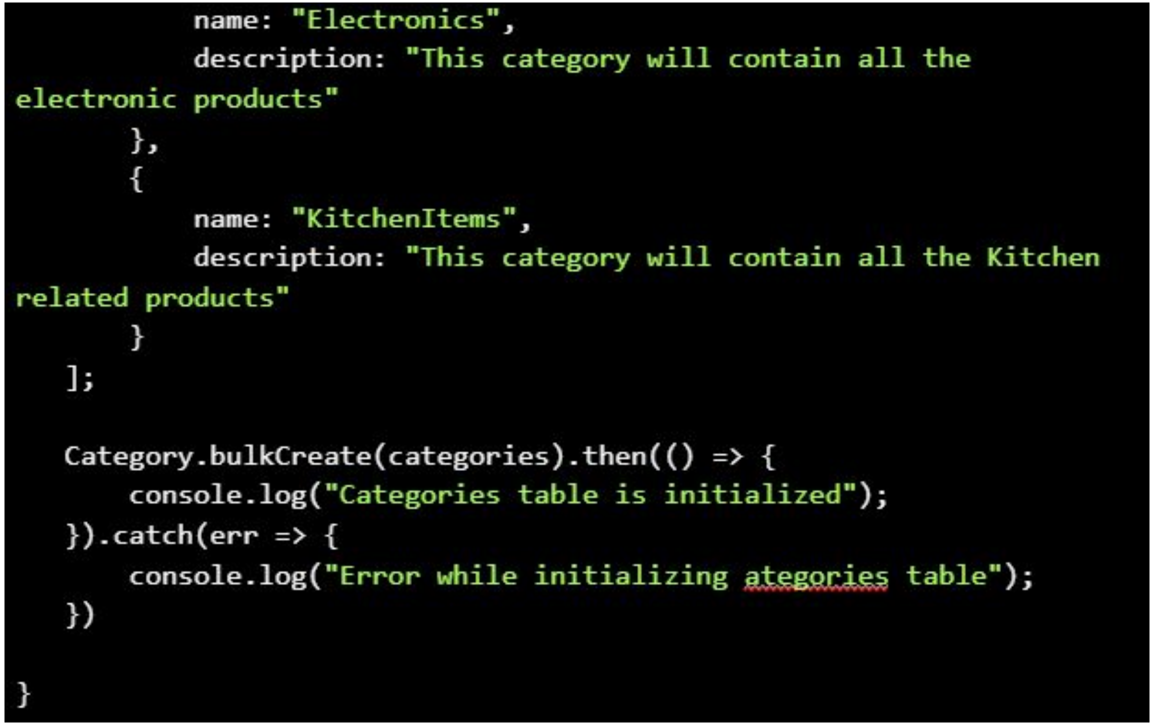
```
const express = require('express');
const serverConfig = require('./configs/server.config');
const bodyParser = require('body-parser');

// 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());
```

Server.js

- Initializing the database
- Adding some initial data for testing



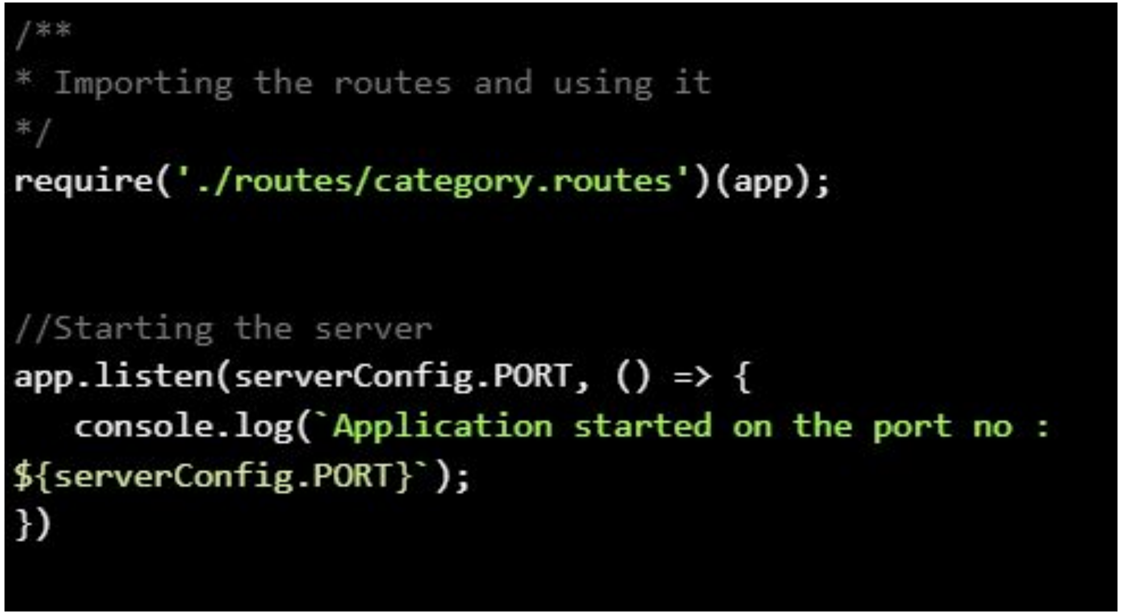
```
/**
* Initializing the database
*/
const db = require("./models");
const Category = db.category;

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


function init() {
   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");
    })

}
```

Server.js

- Importing the routes and using it
- Starting the server



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


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

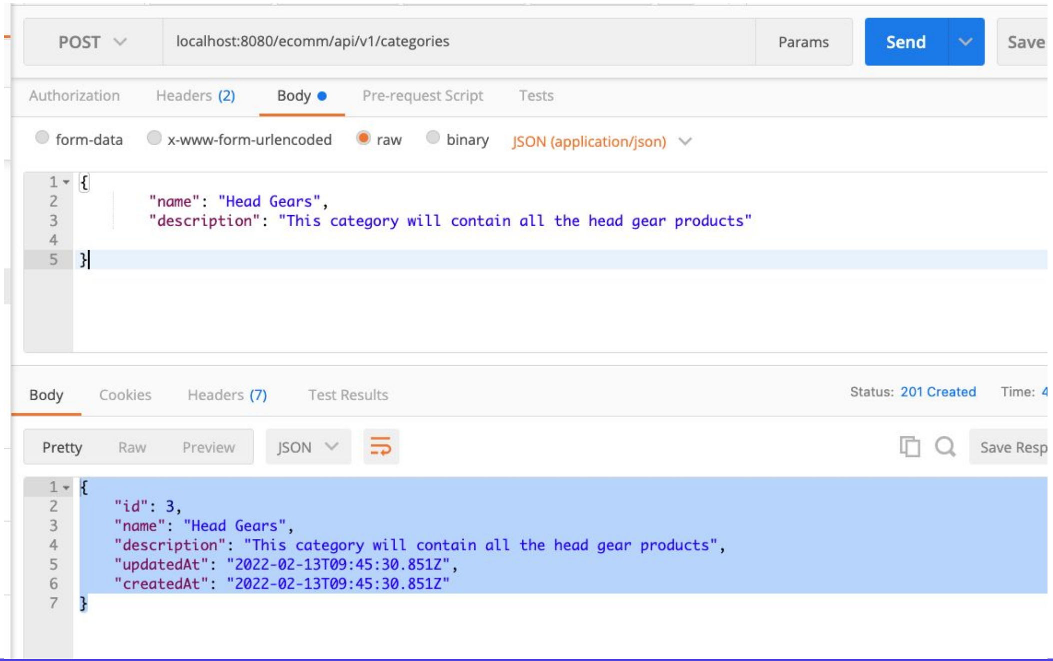
8. Start the node.js server using

- Run the below command from git terminal or command prompt, make sure to at root folder of the project

```
node server.js
```

4. Testing the APIs using Postman

POST API



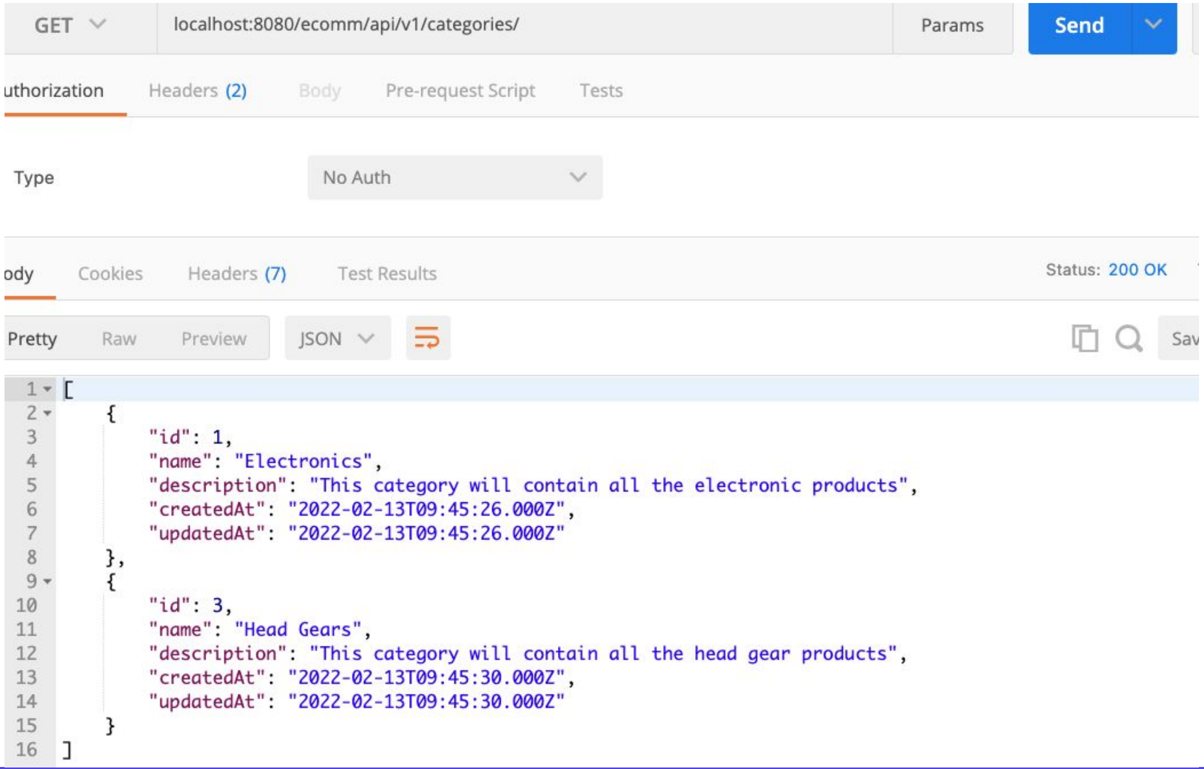
localhost:8080/ecomm/api/v1/categories

```
{
        "name": "Head Gears",
        "description": "This category will contain all the head gear products"

}
```

Status: 201 Created   Time: 4

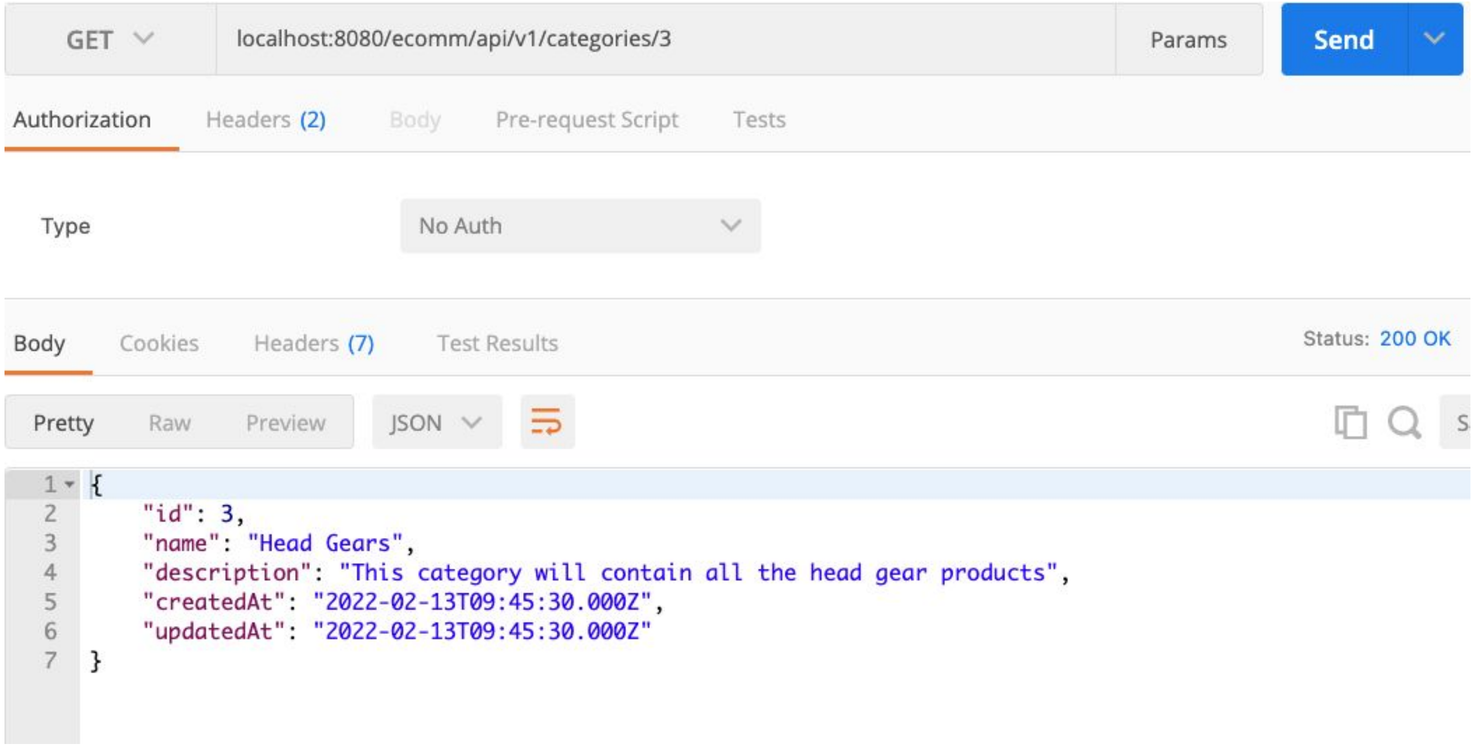
```
{
    "id": 3,
    "name": "Head Gears",
    "description": "This category will contain all the head gear products",
    "updatedAt": "2022-02-13T09:45:30.851Z",
    "createdAt": "2022-02-13T09:45:30.851Z"
}
```

GET ALL API



```
[
    {
        "id": 1,
        "name": "Electronics",
        "description": "This category will contain all the electronic products",
        "createdAt": "2022-02-13T09:45:26.000Z",
        "updatedAt": "2022-02-13T09:45:26.000Z"
    },
    {
        "id": 3,
        "name": "Head Gears",
        "description": "This category will contain all the head gear products",
        "createdAt": "2022-02-13T09:45:30.000Z",
        "updatedAt": "2022-02-13T09:45:30.000Z"
    }
]
```

GET Category based on Id



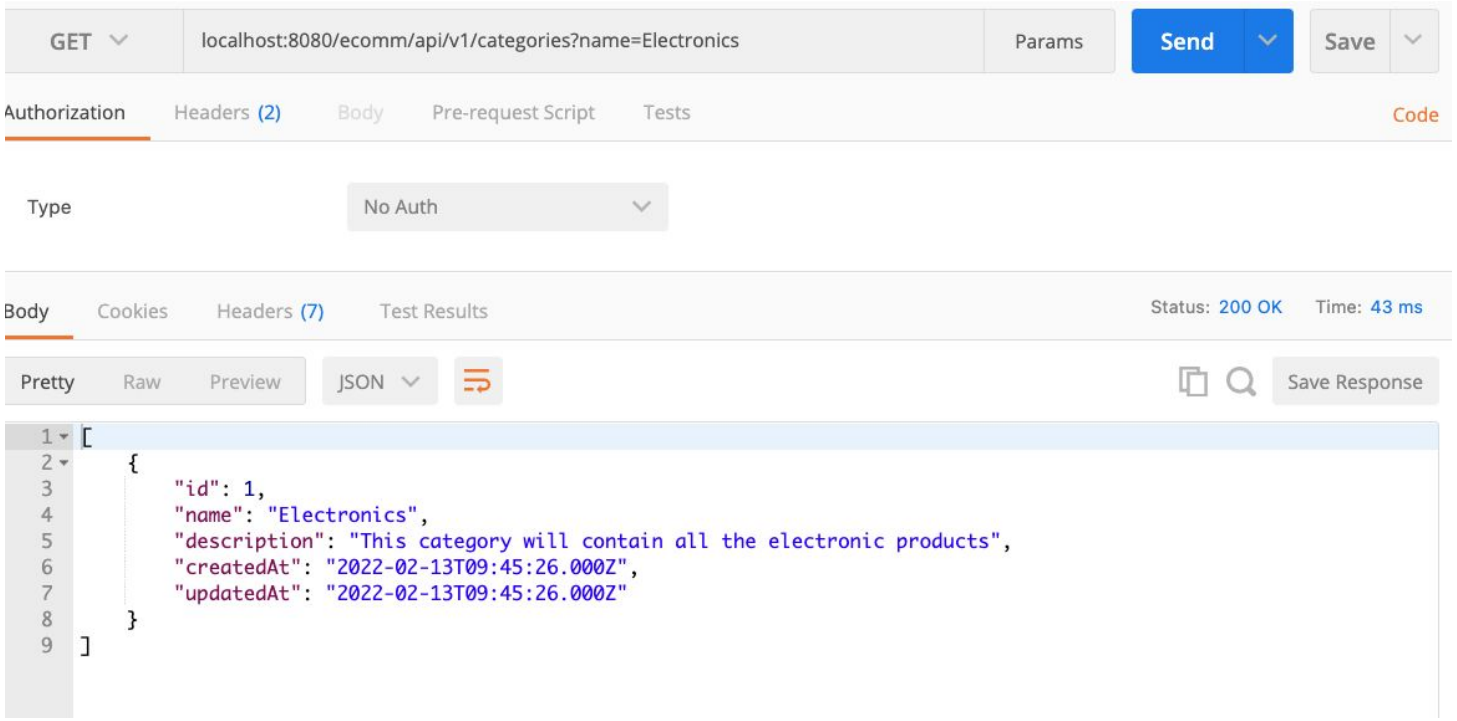
localhost:8080/ecomm/api/v1/categories/3

Type: No Auth

Status: 200 OK

```
{
    "id": 3,
    "name": "Head Gears",
    "description": "This category will contain all the head gear products",
    "createdAt": "2022-02-13T09:45:30.000Z",
    "updatedAt": "2022-02-13T09:45:30.000Z"
}
```

GET category based on name



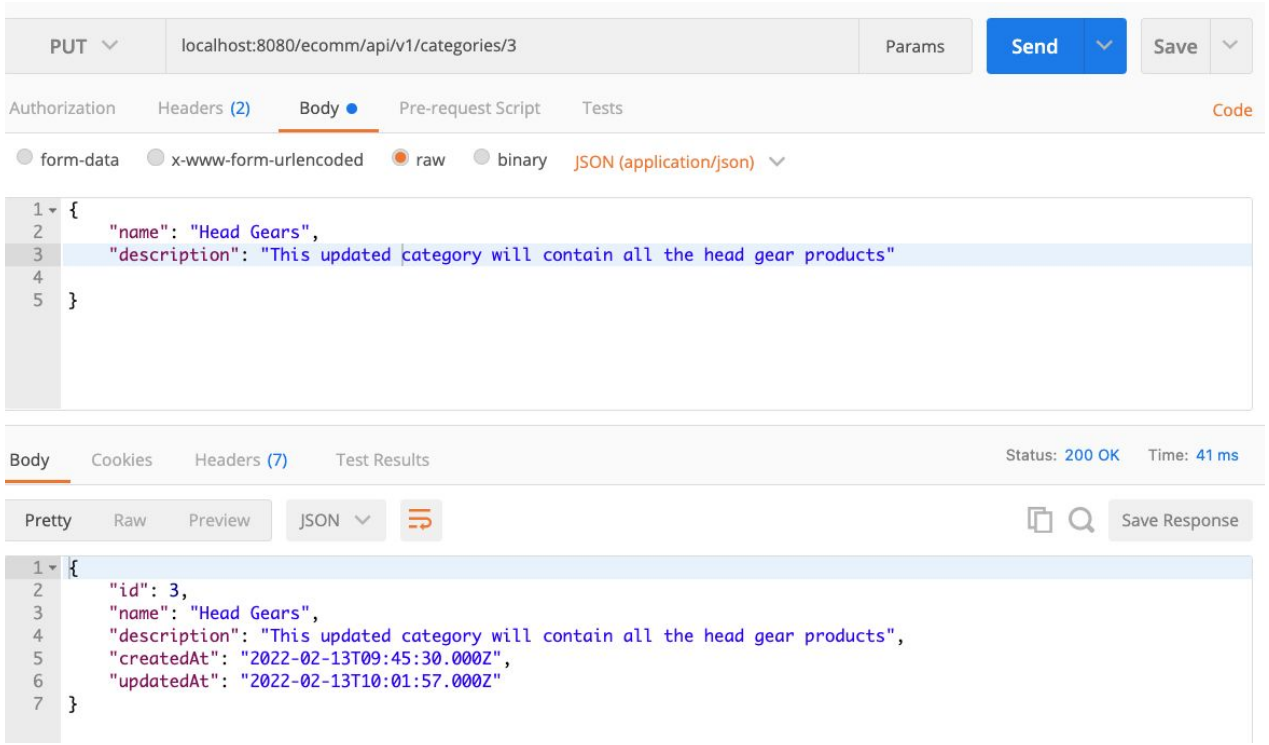
localhost:8080/ecomm/api/v1/categories?name=Electronics

Type: No Auth

Status: 200 OK Time: 43 ms

```
[
    {
        "id": 1,
        "name": "Electronics",
        "description": "This category will contain all the electronic products",
        "createdAt": "2022-02-13T09:45:26.000Z",
        "updatedAt": "2022-02-13T09:45:26.000Z"
    }
]
```

Update Category

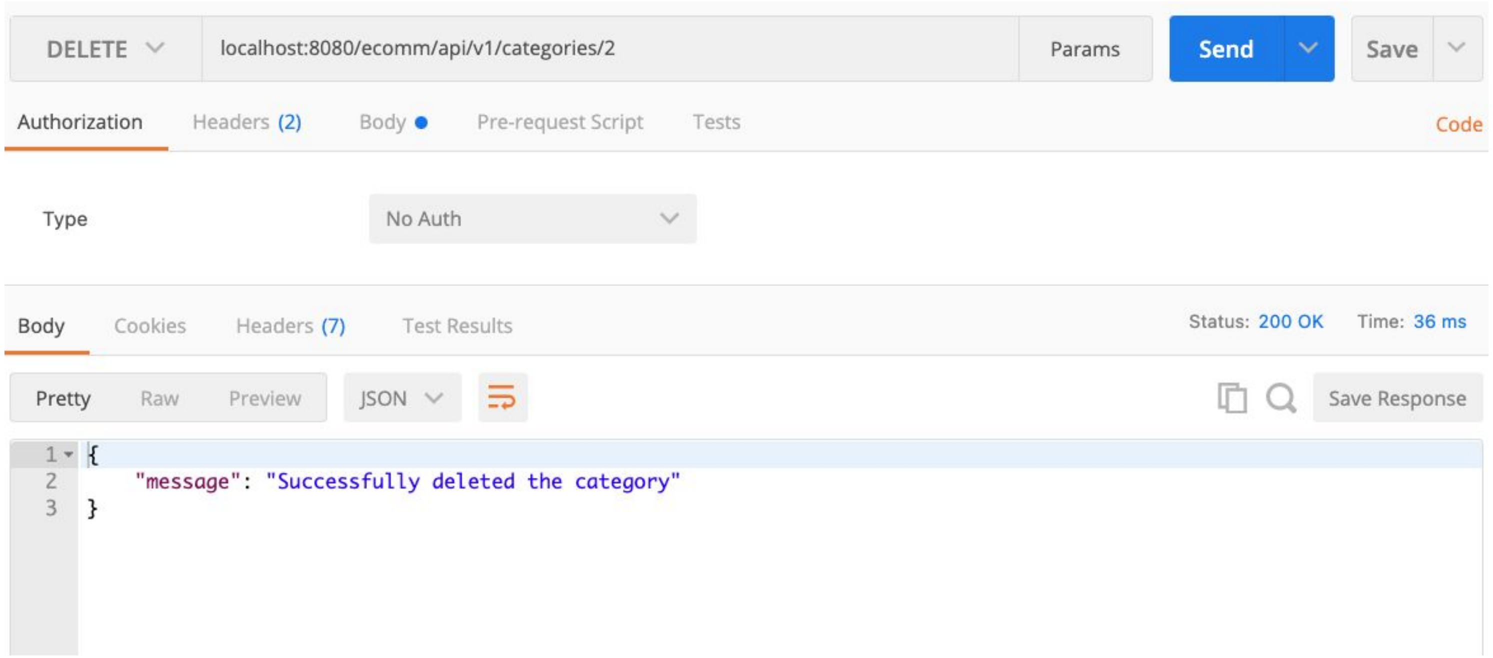


localhost:8080/ecomm/api/v1/categories/3

```
{
    "name": "Head Gears",
    "description": "This updated category will contain all the head gear products"

}
```

```
{
    "id": 3,
    "name": "Head Gears",
    "description": "This updated category will contain all the head gear products",
    "createdAt": "2022-02-13T09:45:30.000Z",
    "updatedAt": "2022-02-13T10:01:57.000Z"
}
```

Delete Category based on the id



localhost:8080/ecomm/api/v1/categories/2

Type: No Auth

Status: 200 OK Time: 36 ms

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

MCQ’s

1. **1. Which of the following is not a valid HTTP method used in RESTful web services?**

A. GET

B. POST

C. TIME

D. PUT

2. Which of the following HTTP Status code means CREATED, when a resource is successfully created using POST or PUT request?

A. 304

B. 204

C. 201

D. 200

3. How would you configure a RESTful URL parameter that supports a search for a book based on its ID?

A. GET /{id}/books/

B. GET /books?id={id}

C. GET /books/{id}

D. GET /book?id={id}

4. CRUD stands for?

A. Create, Receive, Update and Delete

B. Create, Retrieve, Use and Delete

C. Create, Retrieve, Update and Delete

D. Create, Retrieve, Update and Deprecate

5. **Postman is only used to test APIs?**

A. True

B. False

Practice Problems

- From our previous exercise let's continue to add RES API in it and create model and controller on Books and create as many APIs as we can.
  - For example, consider the books as an Object with id, name, author and release date and publisher.
- Create routes and test the APIs using postman.

Next session

- We will define the Product resource.
- We are going to create the Model for Product resource.
- And, then define and implement the RESTful endpoints for the Product resource.

THANK YOU