RESTful API

That can /create/read/update/delete **Product** and **Category** data from a persistence database.

An assignment for the selection of

Node.js Development work from home job/internship at SoluLab

Submitted by:

Biswajit Chanda

(Immediately Available)

Email: biswajitchanda19@gmail.com

Phone: 8876624331

Silchar(Assam).

Objective

Build a RESTful API that can /create/read/update/delete **Product** and **Category** data from a persistence database.

Database Structure

Product Model:

```
{
        productId : xxx,            // Product ID
        productName : xxx,              // Product Name
        qtyPerUnit : xxx,           // Quantity of the Product
        unitPrice : xxx,                // Unit Price of the Product
        unitInStock : xxx,          // Unit in Stock
        discontinued :  xxx,            // Boolean (yes/no)
        categoryId : xxx,           // Category ID
}
```

Category Model:

```
{
        categoryId : xxx,           // Category ID
        categoryName : xxx,             // Category Name
}
```

Functionality:

- The API should follow typical RESTful API design patterns.
- The data should be saved in the DB.
- Category ID in product table should be referenced in the category table.
- Provide proper unit tests.
- Provide proper API documents.
- /create should create the product and category.
- /read should read particular record from the product table (if product has any category then category should be fetched in the response)
- /readAll should read all the records from the product table (if product has any category then category should be fetched in the response)
- /update should update one particular record of the product
- /delete should delete one particular record of the product.

GitHub Link(for source code):

https://github.com/Biswa-Dev/RESTfulAPI.git

Tools that are used:

- Visual Studio Code,
- Command Lind Interface (Hyper terminal),
- Postman,
- Robo 3T

Platform/Libraries/Packages used:

- Node js (Platform)
- Express js (frame work for Node js)
- Body-parser (Node js body parsing middleware)
- Ejs
- Mongoose

Database used: NoSQL MongoDB

Visual Studio Code:

It is a source-code editor used for writing source-codes. Used by most of the developer nowadays. It is made by Microsoft.

Command Line Interface:

It is a tool used by almost all the developers. It helps us to interact with our computer without GUI.

Postman:

It is an API client platform that makes the job easier for developer for creating, testing, sharing, and documentation of API. It allows us to create HTTP requests and also allows as to read their responses.

Robo 3T:

It enable us to use database MongoDB with a Graphical User Interface.

Node js:

It allows us to create backend part using JavaScript.

Express js:

It is a framework or basically a bunch of code that somebody else wrote that adds structure features and helps to organize

And structure your code specifically for web application built with node.

Mongoose:

It package for mongoDB that allows node js app which speaks

Language of javaScript to be able to talk with mongoDB database.

What is an API?

An Application Programming Interface (API) is a set of commands, functions, protocols and objects that programmers can use to create software or interact with an external system.

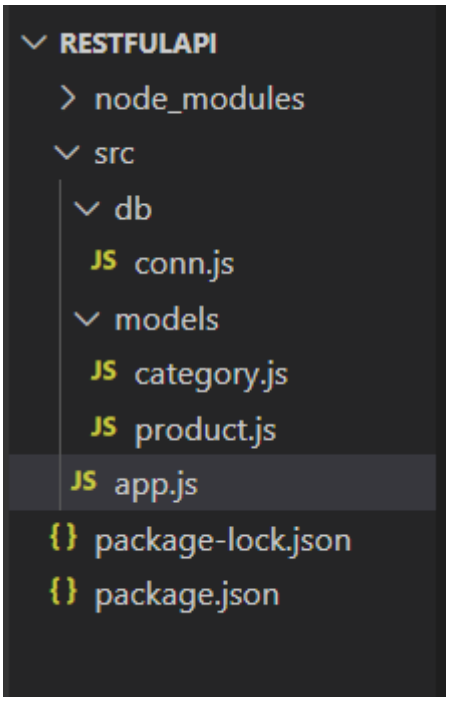
What is RESTful API?

Representational State Transfer is an architectural style for designing API.

How it’s designed?

Setting up the Server Using Hyper Terminal

- First I have created a new Directory called “RESTfulAPI”.
- Then I have initialize NPM and have installed express, body-parser, mongoose and ejs.
- Then I have created a new directory inside “RESTfulAPI” called “src”.
- Inside src I have created two more directories “models”,”db” and also app.js file.
- Inside models I have created two files product.js and category.js for their schema.
- Inside db I have created conn.js a connection file for making connection with database.
- Then I have added server code inside “app.js” file



Setting up MongoDB

- First I have started my mongoDB server using hyper terminal.
- Then by using Robo 3T I had created a new database called “pcDB”.
- Then inside pcDB database I have created two collections products and categories.
- Then I have inserted some documents for both the collections

products collection


```
{
    {
        "_id" : ObjectId("6117af719ad9696574382875"),
        "productId" : 1,
        "productName" : "Keyboard",
        "qtyPerUnit" : 10,
        "unitPrice" : 3000,
        "unitInStock" : 5,
        "discontinued" : false,
        "categoryId" : 1
    },
    {
        "_id" : ObjectId("6117af719ad9696574382878"),
        "productId" : 2,
        "productName" : "Mouse",
        "qtyPerUnit" : 5,
        "unitPrice" : 500,
        "unitInStock" : 10,
        "discontinued" : false,
        "categoryId" : 1
    },
    {
```

```
    "_id" : ObjectId("6117af719ad969657438287b"),
    "productId" : 3,
    "productName" : "Tempered Glass",
    "qtyPerUnit" : 50,
    "unitPrice" : 2500,
    "unitInStock" : 5,
    "discontinued" : false,
    "categoryId" : 2
},
{
    "_id" : ObjectId("6117af719ad969657438287e"),
    "productId" : 4,
    "productName" : "Back Cover",
    "qtyPerUnit" : 10,
    "unitPrice" : 500,
    "unitInStock" : 10,
    "discontinued" : false,
    "categoryId" : 2
},
{
    "_id" : ObjectId("6117af719ad9696574382881"),
    "productId" : 5,
    "productName" : "Think and Grow Rich",
    "qtyPerUnit" : 5,
    "unitPrice" : 1000,
    "unitInStock" : 1,
    "discontinued" : true,
    "categoryId" : 3
},
{
    "_id" : ObjectId("6117af719ad9696574382884"),
    "productId" : 6,
    "productName" : "Hook",
```

```
        "qtyPerUnit" : 3,
        "unitPrice" : 600,
        "unitInStock" : 2,
        "discontinued" : true,
        "categoryId" : 3
    }
}
```


categories collection


```
{
    {
        "_id" : ObjectId("6117ac329ad969657438252a"),
        "categoryId" : 1,
        "categoryName" : "Computer Accessories"
    }
    {
        "_id" : ObjectId("6117ac569ad9696574382555"),
        "categoryId" : 2,
        "categoryName" : "Mobile Accessories"
    }
    {
        "_id" : ObjectId("6117ac6b9ad9696574382564"),
        "categoryId" : 3,
        "categoryName" : "Books"
    }
    {
        "_id" : ObjectId("61180008800f735148c5c4f0"),
        "categoryId" : 4,
        "categoryName" : "Music",
        "__v" : 0
    }
}
```

RESTfulAPI > src > db > conn.js

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

//creating a new database 'pcDB' and getting a connection to local mongo serve
r at 27017
mongoose.connect("mongodb://localhost:27017/pcDB",{
    useNewUrlParser: true,//to handle deprecation warning
    useUnifiedTopology: true,//to handle deprecation warning
    useCreateIndex: true
}).then(() => {
    console.log("Connection with db is successful.")
}).catch((err) => {
    console.log("Connection with db failed!");
})
```

RESTfulAPI > src > models > category.js

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

//creating categorySchema
const categorySchema = new mongoose.Schema({
    categoryId: {
        type: Number,
        required: true,
        unique: [true, "Category ID already present"],
        min: 3,
        max: 3
    },
    categoryName: {
        type: String,
        required: true
    }
})

//Creating new collection based on the schemas
const Category = mongoose.model("Category",categorySchema);

module.exports = Category;
```

RESTfulAPI > src > models > product.js

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

//creating productSchema
const productSchema = new mongoose.Schema({
    producttId: {
        type: Number,
        required: true,
        unique: [true, "Category ID already present"],
        min: 3,
        max: 3
    },
    productName: {
        type: String,
        required: true
    },
    qtyPerUnit: {
        type: Number,
        required: true
    },
    unitPrice: {
        type: Number,
        required: true
    },
    unitStock: {
        type: Number,
        required: true
    },
    discontinued: {
        type: Boolean,
        required: true
    },
    categoryId: {
        type: Number,
        required: true,
        min: 3,
        max: 3
    }
})

//Creating new collection based on the schemas
const Product = mongoose.model("Product",productSchema);

module.exports = Product;
```

RESTfulAPI > src > app.js

```
//requiring all the required packages
const express = require("express");
const bodyParser = require("body-parser");
const ejs = require("ejs");
require("./db/conn"); //requiring db connection
const mongoose = require("mongoose");
const { urlencoded } = require("body-parser");

const port = process.env.PORT || 3000;

//setting our app
const app = express();

const Category = require("./models/category.js");
const Product = require("./models/product.js");

//get method is used for /read/productName route
// to read a specific product and fetching its category if exists
app.get("/read/:pName",function(req,res){
    //findOne() method is used to read a specific product having name pName
    Product.findOne({productName: req.params.pName},function(err,foundProduct)
{
        if(!err){// if every things work perfect
            if(foundProduct){
                Category.findOne({categoryId: foundProduct.categoryId},functio
n(err,foundCategory){// categoryId of the product is used to fetch its categor
y
                    if(!err){
                        if(foundCategory){
                            res.send(foundCategory);// Sending category as res
ponse to req
                        }
                        else{
                            res.send(err);
                        }
                    }else{
                        console.log(err);
                    }
                });
            }else{
                res.send("No Product matching that product name was found.");
            }
        }else{
            console.log(err);
```

```
        }
    });
});

// using post method for /create route for creating the product and category.
app.post("/create",function(req,res){
    //creating new Product by taking req from body
    const newProduct = new Product({
        producttId: req.body.producttId,
        productName: req.body.productName,
        qtyPerUnit: req.body.qtyPerUnit,
        unitPrice: req.body.unitPrice,
        discontinued: req.body.discontinued,
        categoryId: req.body.categoryId
    });
    //creating new Category by taking req from body
    const newCategory = new Category({
        categoryId: req.body.categoryId,
        categoryName: req.body.categoryName
    });
    newProduct.save(function(err){// saving new Product document to databse
        if(!err){
            newCategory.save(function(err){// saving new Category document to
database
                if(err){
                    console.log(err);
                }else{
                    res.send("Product and Category created successfully.");
                }
            });
        }else{
            console.log(err);
        }
    });
});

var catArray = [];// array for storing all categories


//get method is used for /readall route
// to read a all product and fetching there category if exists in response
app.get("/readall",function(req,res){
    // reading all the products form databse using find method
    Product.find(function(err,foundProducts){
        if(!err){
            if(foundProducts){
                foundProducts.forEach(product => { // for each product
                    if(product.categoryId){ // if categoey id of product exist
s
```

```
                        Category.findOne({categoryId: product.categoryId},func
tion(err,foundCategory){ // reading category of id categoryId of product
                            if(!err){ //if there is no error
                                catArray.push(foundCategory); // pushing categ
ory to an array catArray
                            }else{
                                res.send(err);
                            }
                        });
                    }
                });
            }else{
                res.send("No product found.");
            }
        }else{
            res.send("err");
        }
    });
    res.send(catArray);// sending all category documents as a response
    //setting the catArray empty
    if(catArray.length > 0){
        catArray = [];
        catArray.length = 0;
        catArray.splice(0,catArray.length);
    }
});

//usint put and patch method for /update/:pName route
// to update one particular record of the product
app.route("/update/:pName")
.put(function(req,res){// put is used to replace a product document with a new
 one
    Product.replaceOne(
        {productName: req.params.pName},
        {
            producttId: req.body.producttId,
            productName: req.body.productName,
            qtyPerUnit: req.body.qtyPerUnit,
            unitPrice: req.body.unitPrice,
            discontinued: req.body.discontinued,
            categoryId: req.body.categoryId
        },
        {overwrite: true},
        function(err){
            if(!err){
                res.send("Product updated successfully.");
            }else{
                console.log(err);
```

```
            }
        }
    );
})
.patch(function(req,res){// patch is used to modify the existing product docum
ent
    Product.updateOne(
        {productName: req.params.pName},
        {$set: req.body},
        function(err){
            if(!err){
                res.send("Successfully updated product.");
            }
            else{
                res.send(err);
            }
        }
    );
});

//usint delete method for /delete/:pName route
// to delete one particular record of the product having name specified in the
 url by user
app.delete("/delete/:pName",function(req,res){
    console.log(req.params.pName);
    Product.deleteOne(
        {productName: req.params.pName},
        function(err){
            if(!err){
                res.send("Deleted product successfully.");
            }else{
                console.log(err);
            }
        }
    );
});

//lisetening to port whatever is ther in environment variable port or 3000
app.listen(port,function(){
    console.log("Server started on port "+port);
});
```

How I used to design app.js is written in each section of code using comments.

How to run the code?

Before running the code in your system make sure you have all the following software's in your system:

- A code editor e.g., Visual Studio Code
- A Command Lind Interface e.g., Hyper Terminal
- An API client platform e.g., Postman
- Make sure you have installed MongoDB in your system
- A platform for using MongoDB database with GUI e.g., Robo 3T.

Make sure all of them are up to date.

- Download the project directory in your system.
- Make a database named "pcDB" with two collections "categories" and "products".
- Create all the documents as shown above.
- Before running app.js make sure node_modules folder is there inside RESTfulAPI directory.
- Run the app.js file using command line (hyper terminal).
- Use Postman for making request and receiving resposes.

API Documentation

- For reading a specific product and fetching its category.
  Set method to get.
  URL: localhost:3000/read/productName
  Where productName is name of the product.

- For creating new product and category.
  Set method to post.
  URL: localhost:3000/create
  And provide all the key value pair for product and category
  productId: xxx // Number max=3

productName: xxx //String
qtyPerUnit: xxx // Number
unitPrice: xxx // Number
unitStock: xxx //Number
discontinued: xxx //Boolean(true/false)
categoryId: xxx // Number max = 3
categoryName: xxx // String

- For reading all products and fetching their categories.
  Set method to get.
  URL: localhost:3000/readall
  This will return an array of json objects for all categories.

- For updating a specific product record.
  Set method to put to replace a product document with new one.
  URL: localhost:3000/update/productName
  Where productName is name of the product.

  Set method to patch to modify existing product document.
  URL: localhost:3000/update/productName
  Where productName is name of the product.

- For deleting a specific product.
  Set method to delete.
  URL: localhost:3000/delete/productName
  Where productName is name of the product.

GitHub Link(for source code):

https://github.com/Biswa-Dev/RESTfulAPI.git

Thank you very much

Solulab for choosing me and giving me a chance for this assignment.