A GUIDE TO MERN STACK

REACT, NODE JS, EXPRESS & MONGO DB

Suraj Aswal
BCA | MCA | Full-Stack Web Developer
Hemwati Nandan Bahuguna Garhwal University

JS

How To Build Simple RESTful API With NodeJS, ExpressJS And MongoDB

Simple API using ExpressJS and MongoDB with CRUD functions for Contacts.

Required applications

- NodeJS
- Postman
- MongoDB
- IDE

Bootstrapping the Project

To bootstrap our project, we need to verify that we have NodeJS, NPM and MongoDB installed on our machine. To do that, open your terminal or command prompt and run

node -v

This verifies the Nodejs version installed. Run

npm -v

To verify Node Package Manager(npm) installed.

Initialize NodeJS project with **npm init** to setup the project.

```
PS W:\productApp> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (productapp) productapp
version: (1.0.0)
description: Demo project
git repository:
keywords:
author: Suraj Aswal
license: (ISC)
About to write to W:\productApp\package.json:

{
  "name": "productapp",
  "version": "1.0.0",
  "description": "Demo project",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Suraj Aswal",
  "license": "ISC"
}


Is this OK? (yes) yes
PS W:\productApp>
```


Afterwards, we need to install the packages we will be using for our API
The packages are:

1. **ExpressJS:** It's a flexible NodeJS web application that has many features for web and mobile applications
2. **Mongoose:** the mongoDB ODM for Node.JS.
3. **body-parser:** package that can be used to handle JSON requests.

We can install the above mentioned packages via typing the following commands in the command line. Just make sure that you are in the project directory before executing the below command.

npm install --save express body-parser mongoose

Initializing the Server

Create a new file, let's name it ***app.js***

```
//import the express
const express = require('express');
//// Initialize the app
const bodyParser = require('body-parser');
// initialize our express app
const app = express();

//Next step would be dedicating a port number and telling our express app to listen to that port.
let port = 1234;
app.listen(port, () => {
        console.log('Server is up and running on port numner ' + port);
});
```

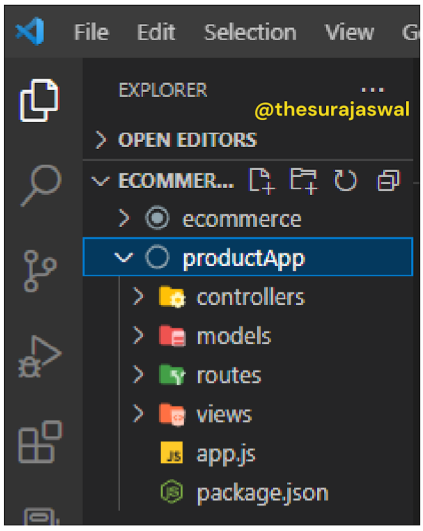

Organizing our application:

We will be working with a design pattern called ***MVC***. Its a neat way of separating parts of our app and grouping them based on their functionality and role. **M** stands for models, this will include all the code for our database models (which in this case will be Products). Then comes the **V** which stands for the views or the layout. The remaining part now is the **C**, which stands for controllers which is the logic of how the app handles the incoming requests and outgoing responses. There will be one more thing, called Routes, Routes are our guide, they tell the client (browser/mobile app) to go to which Controller once a specific URL/path is requested.

Inside the ProductsApp directory, I will create the following four subdirectories

1. **controllers**
2. **models**
3. **routes**
4. **views**

Project Structure!



Now we have a server that is ready to handle our requests and some directories that would have our code.

Let's start by defining our model. Create a new file in the models directory and let's name it ***productModel.js*** file in models folder.

```
//import mongoose
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
//defining product schema
let ProductSchema = new Schema({
    name: {
      type: String,
      required: true,
      max: 100
    },
    price: {
      type: Number,
      required: true
    },
});
// Export the model
module.exports = mongoose.model('Product', ProductSchema);
```

@thesurajaswal

First we started with requiring mongoose and then we define the schema for our model. Last thing is exporting the model so it can be used by other files in our project. Now we are done with the **M** part

Routes: Let's start imagining how the URLs will be like. Let's design our routes. Inside the routes directory,

- create a ***productRoute.js*** file in routes folder.
- This is the file that will include the routes of the products.

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

const router = express.Router();

// Demo Routes URL
router.route("/products").get((req, res, next) => {
  res.status(200).send("Hello");
});

module.exports = router;
```

@thesurajaswal

Controllers: Next step is to implement the controllers we referenced them in the routes:

- create a new ***productController.js*** file in controllers folder**.**
- This will be the placeholder for our controllers.

```
const productModel = require("../models/productModel");

//get all product controller function
exports.getAllProducts = (req, res, next) => {
  const product = `Hello World`;

  res.status(200).send(product);
};
```

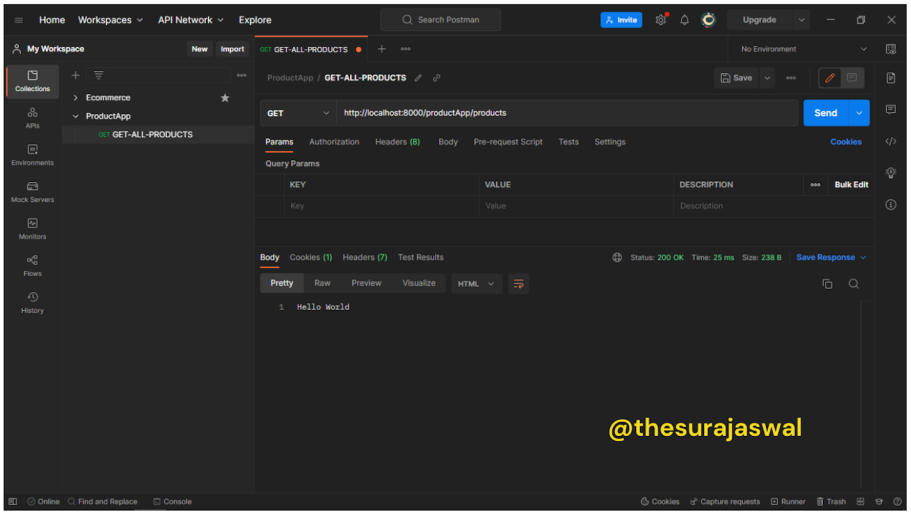
@thesurajaswal

product controller Snippet

Postman:

Postman is a very powerful HTTP client that is used for testing, documenting and the development of APIs. We will be using Postman here to test our endpoints that we will be implementing through out the rest of the project.

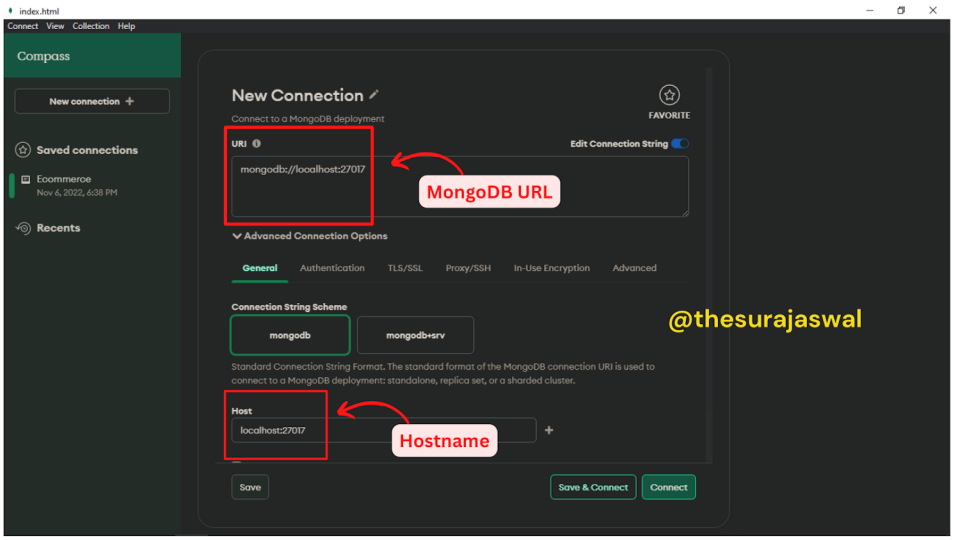
- Install Postman from their website.
- Open the app, make sure it's a GET request and type the following URL ***'localhost:8000/products/'***. Just make sure that your server is still running on the port number 8000.



Postman Screenshot

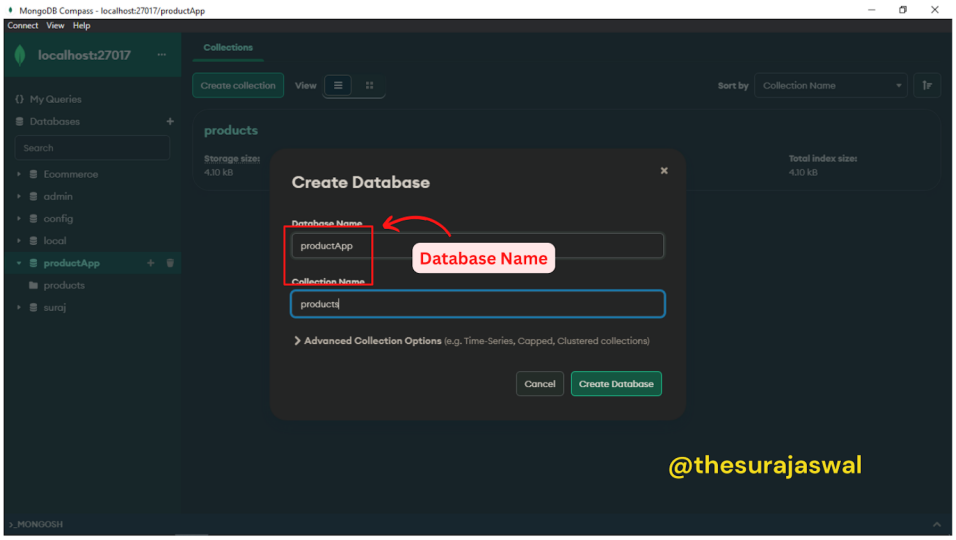
The Database

- Our database will be hosted locally on our device.
- Install MongoDB from https://www.mongodb.com/try/download/community
- Install MongoDB Compass.
- MongoDB Compass is a powerful GUI for querying, aggregating, and analyzing your MongoDB data in a visual environment.



MongoDB Compass Screenshot

Creating Database(**productApp**) with the help of MongoDB Compass GUI



Database Creation Screenshot

Connecting our app to the Database

- We need to inform our app that it should be communicating with the database we have just created ***"productApp"***
- Remember the 'mongoose' package we installed before? Now is the right time to use.
- create a separate file ***database.js*** and import that file in ***app.js*** file and call ***connectDB()*** function.

```
//importing database.js file in app.js
const connectDB = require("./database.js");
//connecting to mongoDB Database
connectDB();
```

JS

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

const connectDB = () => {
  mongoose
    .connect('mongodb://localhost:27017/productApp', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    })
    .then((data) => {
      console.log(
        `Mongo DB Connected Succesfully with=> ${data.connection.host}`
      );
    });
};

module.exports = connectDB;
```

@thesurajaswal

Database Connection Snippet

Body Parser

Body Parser is an npm package that is used to parse the incoming request bodies in a middleware.

In app.js file, add the following couple of lines.

```
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: false}));
```

Implementing the endpoints

CREATE

The first task in our CRUD task is to create a new product. Let's start by defining our route first. Head to routes and start designing the expected path that the browser would hit and the controller that would be responsible for handling that request.

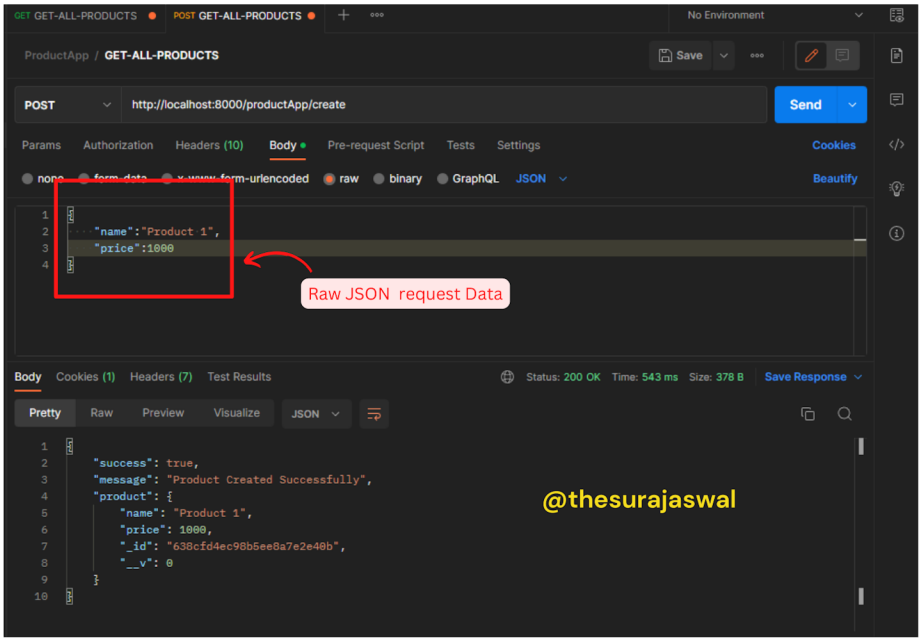
```
router.route('/create').post(createProduct); //productRoute.js
```

```
// importing product schema
const Product = require("../models/productModel");

// creating a new product
exports.createProduct = async (req, res, next) => {
  const product = await Product.create({
    name: req.body.name,
    price: req.body.price,
  });

  product.save();

  res.status(200).json({
    success: true,
    message: "Product Created Successfully",
    product,
  });
};
```

product controller- Create Product Snippet

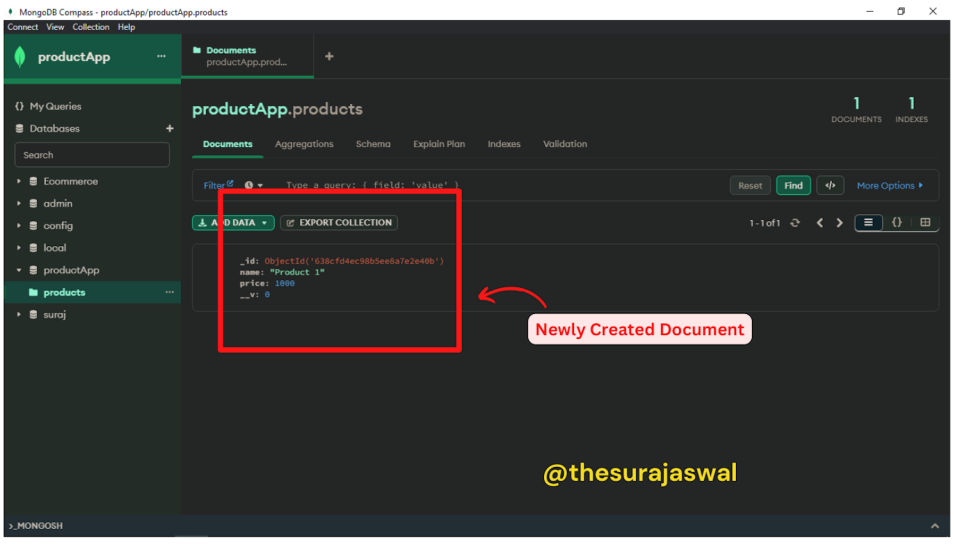
Last step would be validating that we can easily create a new product.

- open **Postman** App.
- send a **POST** request to the URL ***'http://localhost:8000/productApp/create'***
- send raw **JSON** data through body. For eg: `{"name": "Product 1", "price":1000}`



Validating Create Product API

A new collection is created named 'products' and has one document.



Compass Document

READ

The second task in our project is to read an existing product.

Update the ***productRoute.js*** file with new product routes and define a controller function(***exports.getAllProducts***) in ***productController.js*** file for the ***'/products'*** route.

```
//get all products route
router.route('/products').get(getAllProducts); //productRoute.js
```

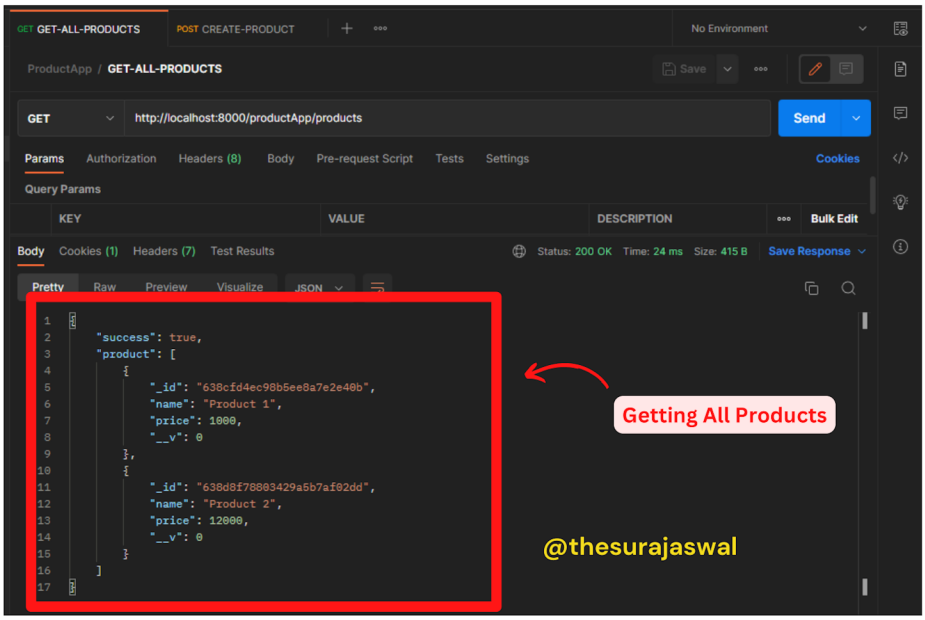
```js
// Get All Products
exports.getAllProducts = async (req, res, next) => {
  const product = await Product.find();

  if (!product) {
    return next();
  }

  res.status(200).json({
    success: true,
    product,
  });
};
```

@thesurajaswal

product controller- Get All Products Snippet

What the ***getAllProducts*** function does is it simply reads the existing products from the ***Product*** collection.

Now let's head to Postman and send a **GET** request to the URL ***'http://localhost:8000/productApp/products'***



Validating Get All Products API

UPDATE

The Third task in our project is to update an existing product.

Update the ***productRoute.js*** file with new product routes and define a controller function(***exports.updateProduct***) in ***productController.js*** file for the ***'/products/:id'*** route.

```
//update product route
router.route('/products/update/:id').put(updateProduct);
```

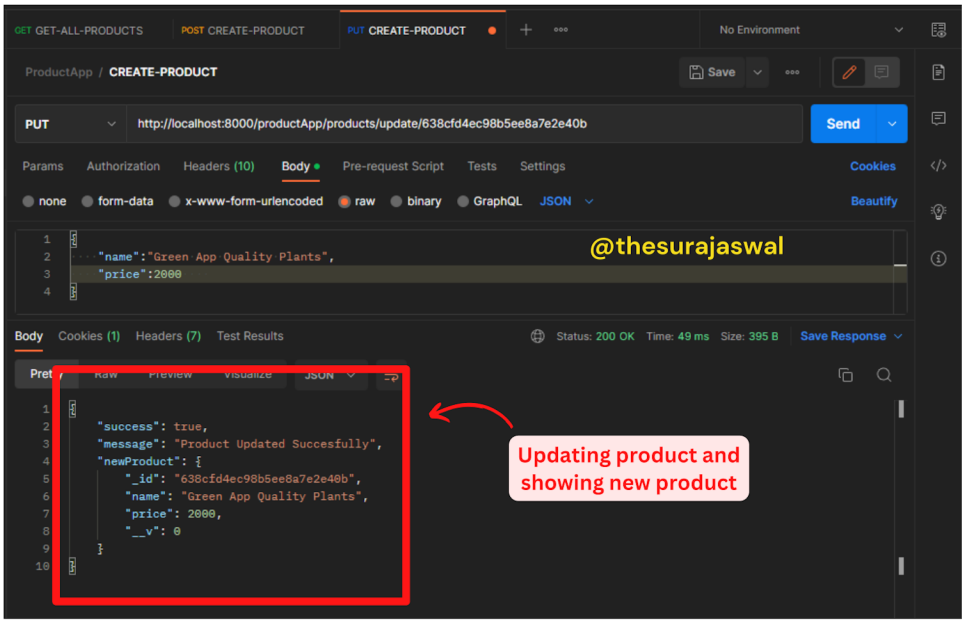
```
// update product
exports.updateProduct = async (req, res, next) => {
  const product = await Product.findByIdAndUpdate(req.params.id, {
    name: req.body.name,
    price: req.body.price,
  });

  res.status(200).json({
    success: true,
    message: "Product Updated Succesfully",
    newProduct,
  });
};
```

@thesurajaswal

Product Controller Update Product Snippet

In the function ***updateProduct*** we have used ***Product.findByIdAndUpdate()*** function which in turn takes two arguments one is the ***_id*** of the product which we want to update and second argument is the object of updated data to be inserted.

Now switch to the postman app and send a **PUT** request to the URL *'http://localhost:8000/productApp/products/update/**product_id**'*



Validating Update Product API

DELETE

The last task in our project is to delete an existing product.

Update the ***productRoute.js*** file with new product routes and define a controller function(***exports.deleteProduct***) in ***productController.js*** file for the ***'/products/delete/:id'*** route.

```
//delete product route
router.route('/products/delete/:id').delete(deleteProduct);
```

JS

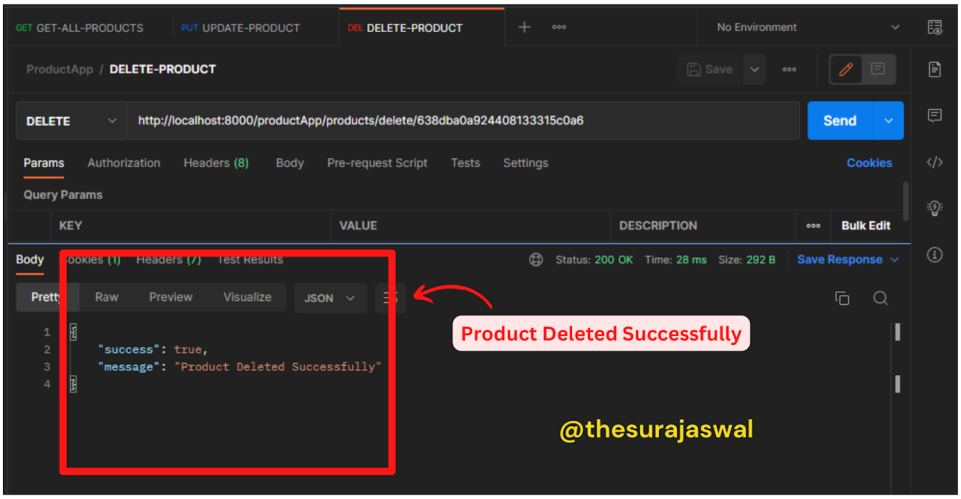
```
// Delete product
exports.deleteProduct = async (req, res, next) => {
  const product = await Product.findById(req.params.id);

  if (!product) {
    return next();
  }

  product.remove();

  res.status(200).json({
    success: true,
    message: "Product Deleted Successfully",
  });
};
```

@thesurajaswal

Product Controller Delete Product Snippet

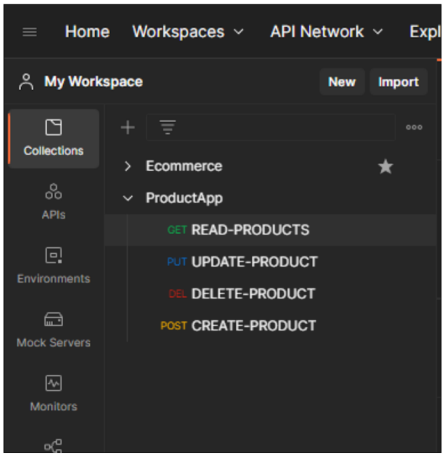
In the function ***deleteProduct*** we have used ***Product.findById()*** function which is used to get the desired product using product ***_id*** and we have also used ***product.remove()*** method to delete the document.

Now switch to the postman app and send a **DELETE** request to the URL *'http://localhost:8000/productApp/products/delete/**product_id**'*



Validating Update Product API

All four CRUD API - CREATE, READ, UPDATE, DELETE



JS

App.js

```
const express = require("express");
const connectDB = require("./database.js");
const bodyParser = require("body-parser");
const product = require("./routes/productRoute");

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

app.use(express.json());
// body-parse
app.use(bodyParser.urlencoded({ extended: true }));
// Routes
app.use("/productApp", product);

// connecting to database
connectDB();

// PORT
let port = 8000;
app.listen(port, () => {
  console.log("Server is up and running on port numner " + port);
});
```

Database.js

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

const connectDB = () => {
  mongoose
    .connect(`mongodb://localhost:27017/productApp`, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    })
    .then((data) => {
      console.log(
        `Mongo DB Connected Succesfully with=>
${data.connection.host}`
      );
    });
};

module.exports = connectDB;
```

productModel.js

```
const mongoose = require("mongoose");
const productSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  price: {
    type: Number,
    required: true,
  },
});

module.exports = mongoose.model("Product",
productSchema);
```

JS

```
// importing product schema
const Product = require("../models/productModel");

// creating a new product
exports.createProduct = async (req, res, next) => {
  const product = await Product.create({
    name: req.body.name,
    price: req.body.price,
  });

  product.save();

  res.status(200).json({
    success: true,
    message: "Product Created Successfully",
    product,
  });
};

// Get All Products
exports.getAllProducts = async (req, res, next) => {
  const product = await Product.find();

  if (!product) {
    return next();
  }

  res.status(200).json({
    success: true,
    totalProducts: product.length,
    product,
  });
};

// update product
exports.updateProduct = async (req, res, next) => {
  const product = await Product.findByIdAndUpdate(req.params.id, {
    name: req.body.name,
    price: req.body.price,
  });

  let newProduct = await Product.findById(req.params.id);

  res.status(200).json({
    success: true,
    message: "Product Updated Succesfully",
    newProduct,
  });
};

// Delete product
exports.deleteProduct = async (req, res, next) => {
  const product = await Product.findById(req.params.id);

  if (!product) {
    return next();
  }

  product.remove();

  res.status(200).json({
    success: true,
    message: "Product Deleted Successfully",
  });
};
```

productController.js

productRoute.js

```
const express = require("express");
const {
  getAllProducts,
  createProduct,
  updateProduct,
  deleteProduct,
} = require("../controllers/productController");

const router = express.Router();

//get all product routes
router.route("/products").get(getAllProducts);

// update product routes
router.route("/products/update/:id").put(updateProduct);

// create product routes
router.route("/products/create").post(createProduct);

// delete product routes
router.route("/products/delete/:id").delete(deleteProduct);

// export routes
module.exports = router;
```