Build Crud API with Node.js, Express, and MongoDB

Application Programming Interface is the abbreviation for API. An API is a software interface that enables two apps to communicate with one another. In other words, an API is a messenger that sends your request to the provider and then returns the response to you.

we'll be building a RESTful CRUD (Create, Retrieve, Update, Delete) API with Node.js, Express, and MongoDB.

Creating the application

On your Desktop (or any other place) create a new folder named nodejs-api and open it in any Code Editor (for this Tutorial I am using VS Code). Once you done open terminal (You can either use VS Code terminal or external terminal) and run

npm init -y

This will generate a simple package.json and now we need to install some dependencies which we need. Fire up your terminal and run

npm install express body-parser mongoose --save

💡 Mongoose is an ODM (Object Document Mapping) tool for Node.js and MongoDB. It helps you convert the objects in your code to documents in the database and vice versa.

This will install Express (for server), Mongoose, and Body Parse for parsing data

💡 The body-parser middleware converts text sent through an HTTP request to a target format or in other words body-parser parses your request and converts it into a format from which you can easily extract relevant information that you may need

Now once everything is installed, we can start creating our web server.

Setting up the webserver

Create a new file named `server.js` in the root folder of the application and add the following code to it

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();

app.use(bodyParser.urlencoded({ extended: true }))

app.use(bodyParser.json())

app.get('/', (req, res) => {
    res.json({"message": "Server is running :D"});
});

let PORT = 8080

app.listen(PORT, () => {
    console.log(`Server is listening on port ${PORT}`);
});
```

In the above code

- First, we imported the dependencies which we need i.e Express and Body Parser
- Second, Once we imported them we added body-parser middlewares using express's `app.use()` method
- Then, We defined a simple GET route that returns a message that the server is running.
- Finally, we defined the port and listen to that port for incoming connections

Now in your terminal, run `node server.js` and go to `http://localhost:8080` to access the route we just defined. and you should see

```
{
  message: "Server is running :D"
}
```

Connecting our application to MongoDB

In **server.js** import mongoose, just like the code below

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

mongoose.Promise = global.Promise;
```

and add the below code after it

```
mongoose.connect(YOUR_MONGODB_URL, {
    useNewUrlParser: true
}).then(() => {
    console.log("Successfully connected to the database");
}).catch(err => {
    console.log('Could not connect to the database. Error...', err);
    process.exit();
});
```

finally, this is how your **server.js** should look like now

```
const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");

mongoose.Promise = global.Promise;
mongoose.connect(YOUR_MONGODB_URL,
    {
      useNewUrlParser: true,
    }
  )
  .then(() => {
    console.log("Successfully connected to the database");
  })
  .catch((err) => {
    console.log("Could not connect to the database. Error...", err);
    process.exit();
  });

const app = express();

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

```
app.use(bodyParser.json());

app.get("/", (req, res) => {
  res.json({ message: "Server is running :D" });
});

let PORT = 8080;

app.listen(PORT, () => {
  console.log(`Server is listening on port ${PORT}`);
});
```

Defining the model in Mongoose

Now, in the root directory create a new folder named **app** and inside of it create another folder named **models.**
Create a new file named **app.model.js** and add the following code inside of it

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

const AppSchema = mongoose.Schema({
  message: String,
});

module.exports = mongoose.model("App", AppSchema);
```

This model contains one field that is **message**

Defining the routes

Now create a new folder called routes inside the app folder and add the following code inside of it

```
module.exports = (app) => {
  const App = require("../controllers/app.controller.js");

  app.post("/create", App.create);

  app.get("/get-all", App.findAll);

  app.get("/message/:messageId", App.findOne);

  app.put("/message/:messageId", App.update);

  app.delete("/message/:messageId", App.delete);
```

```
};
```

include the routes in server.js. Add the following require statement before app.listen() line inside **server.js file**.

```
// ........
require('./app/routes/app.routes.js')(app);
// ........
```

Writing the Controller functions

Create a new folder called controllers inside the app folder, then create a new file called app.controller.js inside app/controllers folder with the following contents -

```
const App = require("../model/app.model.js");

// Create and Save a new Message
exports.create = (req, res) => {
  const message = new App({
    message: req.body.message,
  });
  message
    .save()
    .then((data) => {
      res.send(data);
    })
    .catch((err) => {
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the Message.",
      });
    });
};

// Retrieve all messages from the database.
exports.findAll = (req, res) => {
  App.find()
    .then((data) => {
      res.send(data);
    })
    .catch((err) => {
      res.status(500).send({
        message:
          err.message || "Some error occurred while retrieving messages.",
      });
```

```
    });
};

// Find a single message with a messageId
exports.findOne = (req, res) => {
  App.findById(req.params.messageId)
    .then((data) => {
      if (!data) {
        return res.status(404).send({
          message: "Message not found with id " + req.params.messageId,
        });
      }
      res.send(data);
    })
    .catch((err) => {
      if (err.kind === "ObjectId") {
        return res.status(404).send({
          message: "Message not found with id " + req.params.messageId,
        });
      }
      return res.status(500).send({
        message: "Error retrieving message with id " + req.params.messageId,
      });
    });
};

// Update a message identified by the messageId in the request
exports.update = (req, res) => {
  App.findByIdAndUpdate(
    req.params.messageId,
    {
      message: req.body.message,
    },
    { new: true }
  )
    .then((data) => {
      if (!data) {
        return res.status(404).send({
          message: "Message not found with id " + req.params.messageId,
        });
      }
      res.send(data);
    })
    .catch((err) => {
      if (err.kind === "ObjectId") {
        return res.status(404).send({
          message: "Message not found with id " + req.params.messageId,
        });
      }
      return res.status(500).send({
```

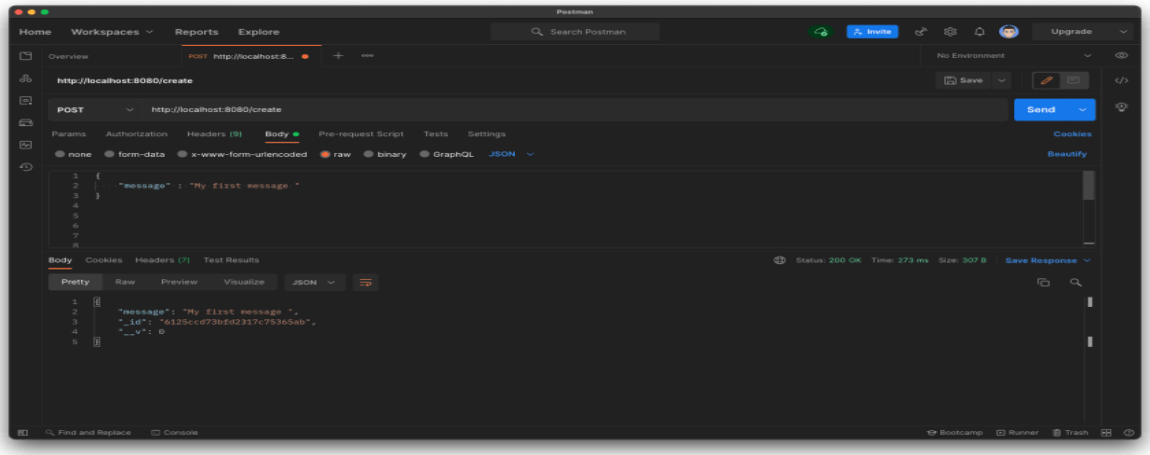
```
        message: "Error updating message with id " + req.params.messageId,
      });
    });
};

// Delete a message with the specified messageId in the request
exports.delete = (req, res) => {
  App.findByIdAndRemove(req.params.messageId)
    .then((data) => {
      if (!data) {
        return res.status(404).send({
          message: "Message not found with id " + req.params.messageId,
        });
      }
      res.send({ message: "Message deleted successfully!" });
    })
    .catch((err) => {
      if (err.kind === "ObjectId" || err.name === "NotFound") {
        return res.status(404).send({
          message: "Message not found with id " + req.params.messageId,
        });
      }
      return res.status(500).send({
        message: "Could not delete message with id " + req.params.messageId,
      });
    });
};
```

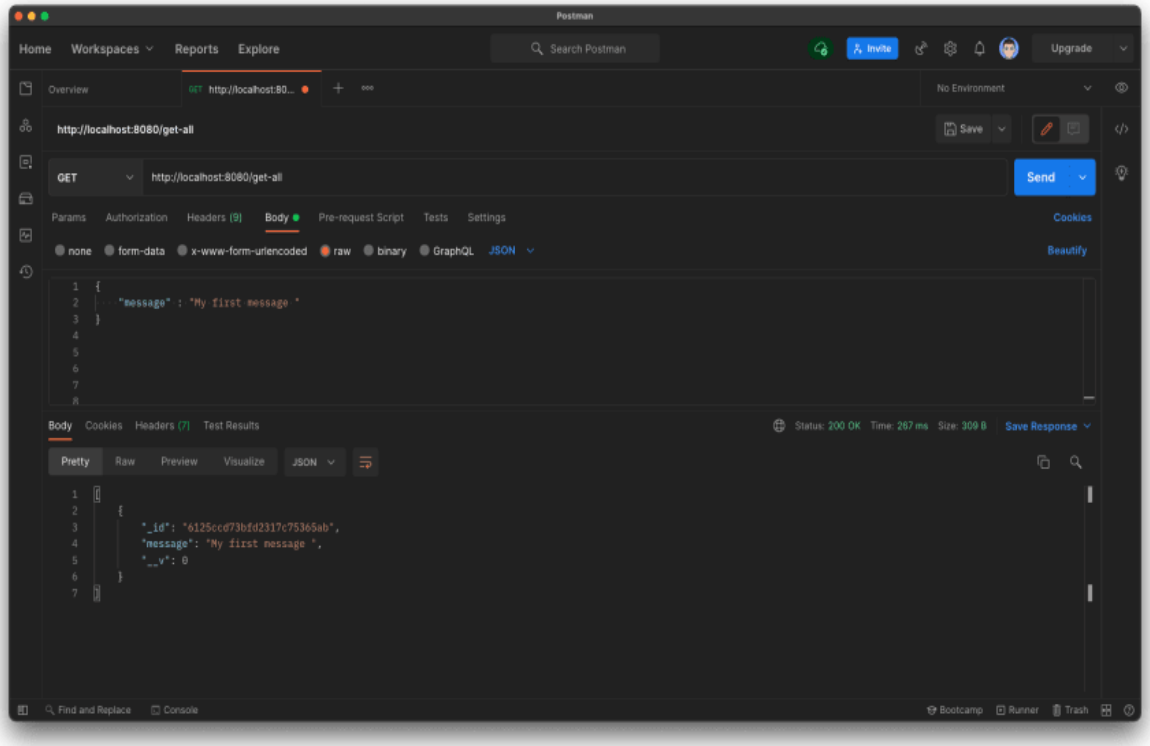
restart your node.js server an ✸ now we have our API ready

Testing APIs with postman

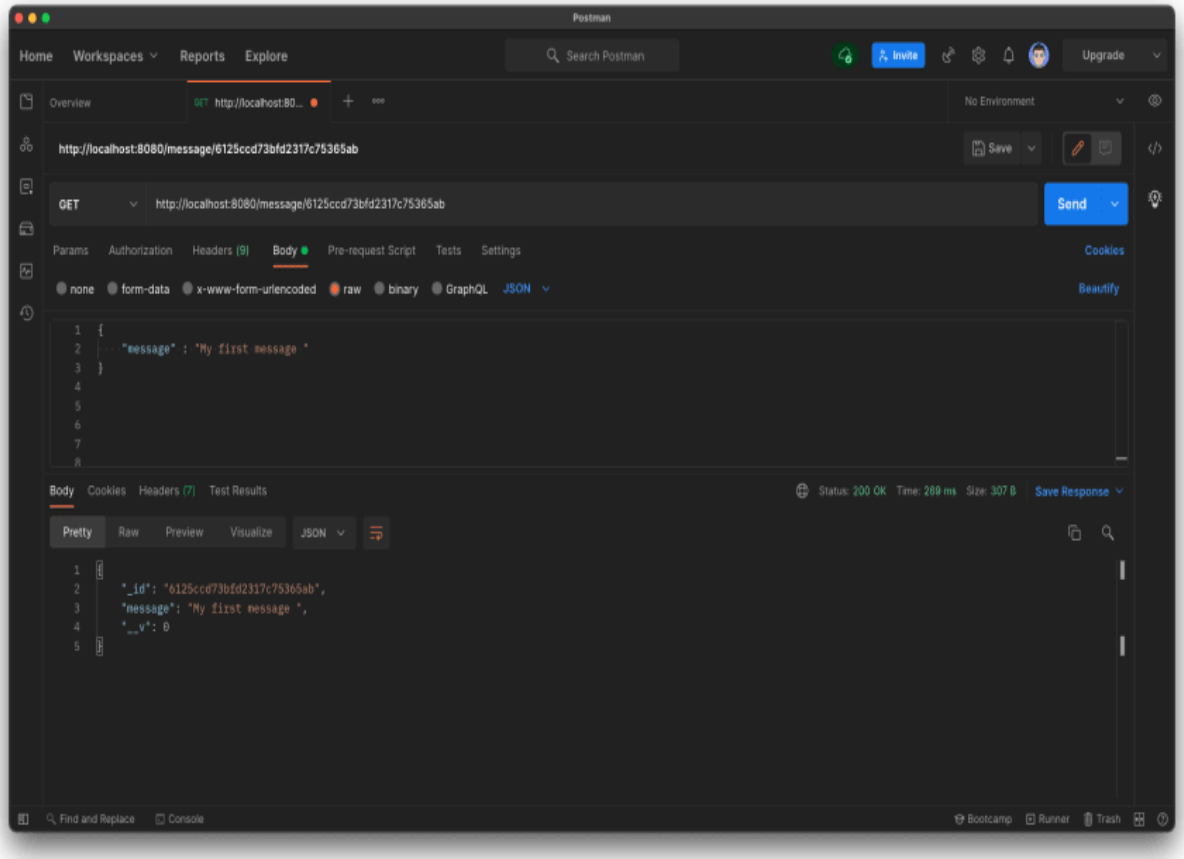
- Create and Save a new Message



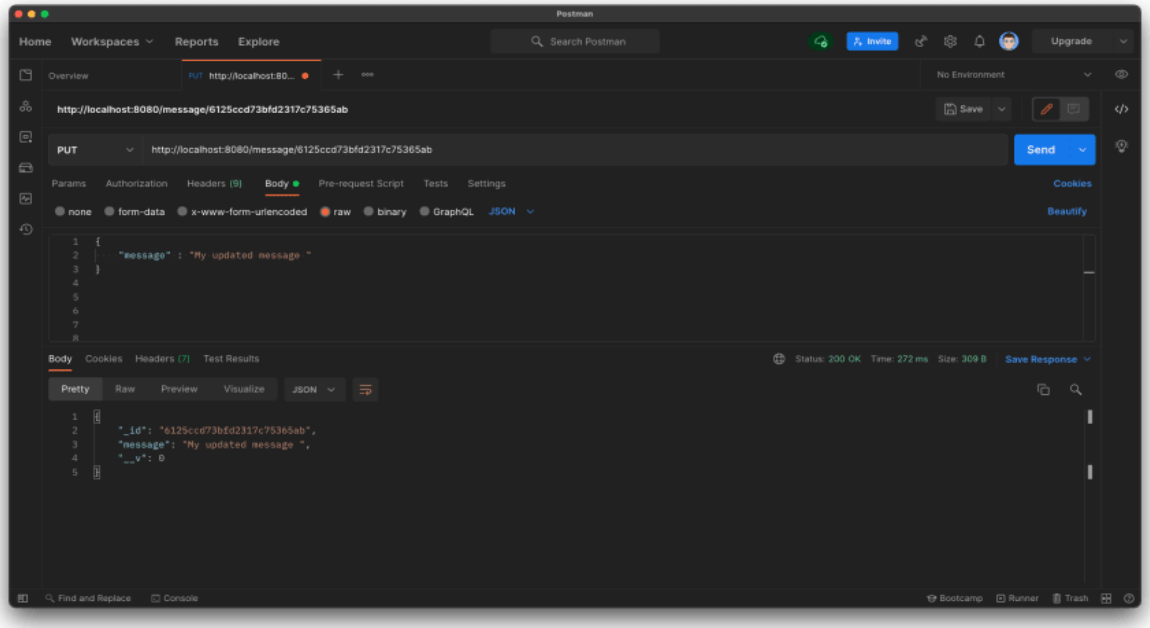
- Retrieve all messages from the database



- Find a single message with a messageId



- Update a message identified by the messageId in the request



- Delete a message with the specified messageId in the request