ASSIGNMENT-3

Aim: Create a RESTful API using express.js and create a database and index in MongoDB.

Source code:

```
const express = require('express')
const mongoose = require("mongoose");
const Task = require("./model/task");
require("dotenv").config(); //for using variables from .env file.
const app = express()
const port = 3000

//middleware provided by Express to parse incoming JSON requests.
app.use(express.json());
mongoose.connect(process.env.MONGODB_URL).then(() => {
   console.log("MongoDB is connected!");
 });
app.get('/', (req, res) => {
 res.send('Hello World!')
})
```

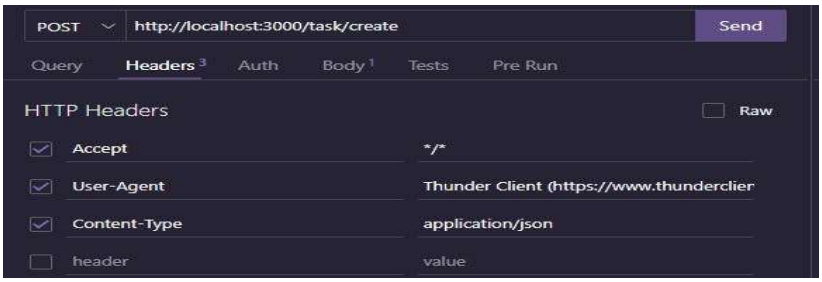
Implementing CRUD Operations:

1. POST request:

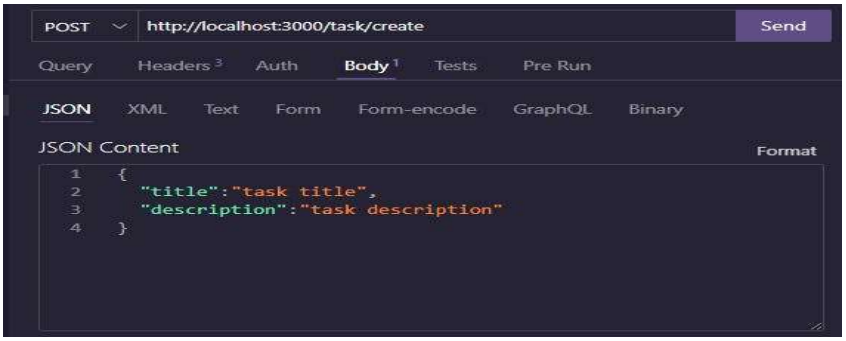
```
// Create an task item
app.post("/tasks/create", async (req, res) => {
 try {
   const newTask = await Todo.create(req.body);
   res.status(201).json(newTask);
 } catch (error) {
   res.status(500).json({ error: "Internal Server Error" });
 }
});
```

ASSIGNMENT-3

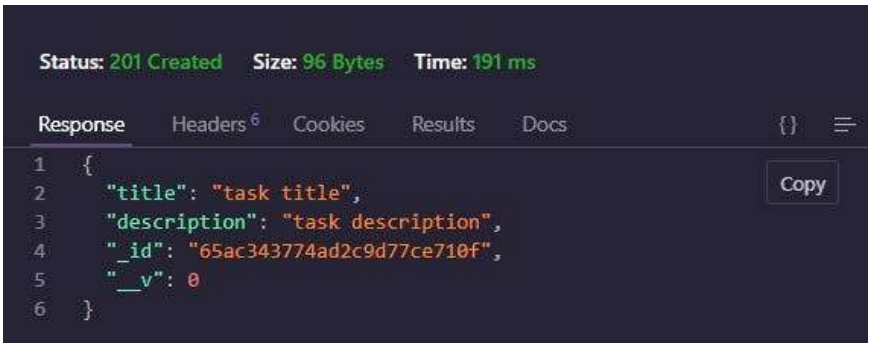
Make sure that the Content-Type is set to application/json in the Headers tab, then select POST and write the URL along the lines as in the image below.



Make sure to include a title and description of the Task item in the Body tab, just like we see in the image.



The following response will be displayed after you click the Send button:

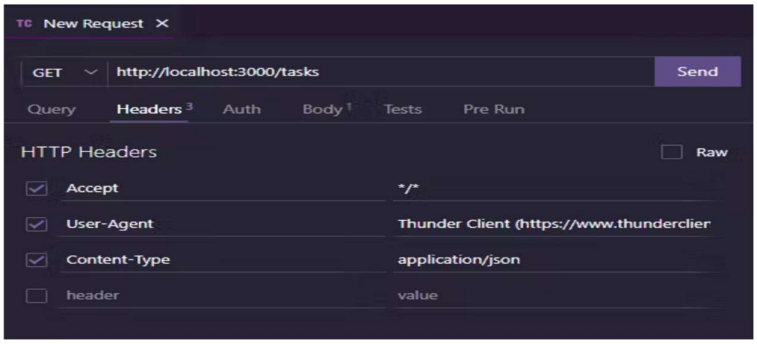


2.GET request:

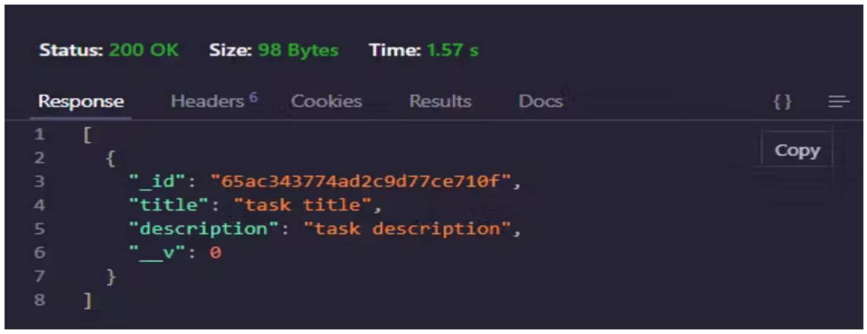
Reading is getting information out of a database using queries and retrieval. You can use this operation to retrieve data according to specific criteria, like whether you need to get one document, a few documents, or the entire collection.

```
// Read all todos
app.get("/tasks", async (req, res) => {
  try {
    const tasks = await Task.find();
    res.status(200).json(tasks);
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: "Internal Server Error" });
  }
});
```

To access the tasks, select GET, copy the URL (as seen in the figure below), and then hit the Send button.



As can be seen in the image, you will receive the response.



Make sure to include a title and description of the Task item in the Body tab, just like we see in the image.

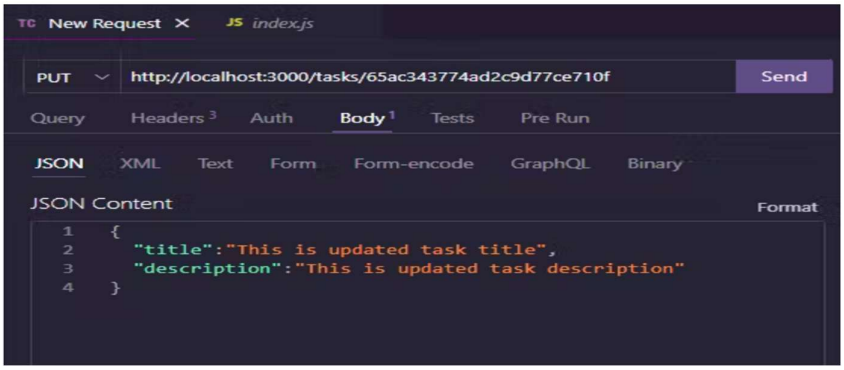
3. PUT request:

We are going to require the id of the task item in order to properly update the task. It is therefore necessary for you to copy the id of the task that you obtained in the previous response.

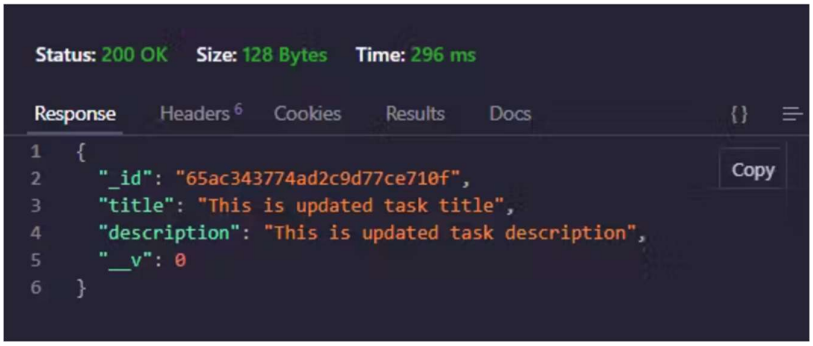
```
// Update a task by ID
app.put("/tasks/:id", async (req, res) => {
  try {
    const updatedTask = await Task.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
    });
    res.status(200).json(updatedTask);
  } catch (error) {
    res.status(500).json({ error: "Internal Server Error" });
  }
});
```

To write the URL, select the PUT option and write it down as seen in the image.

Now, change the title and description in the Body tab.



When you click "Send," you'll get the message below.

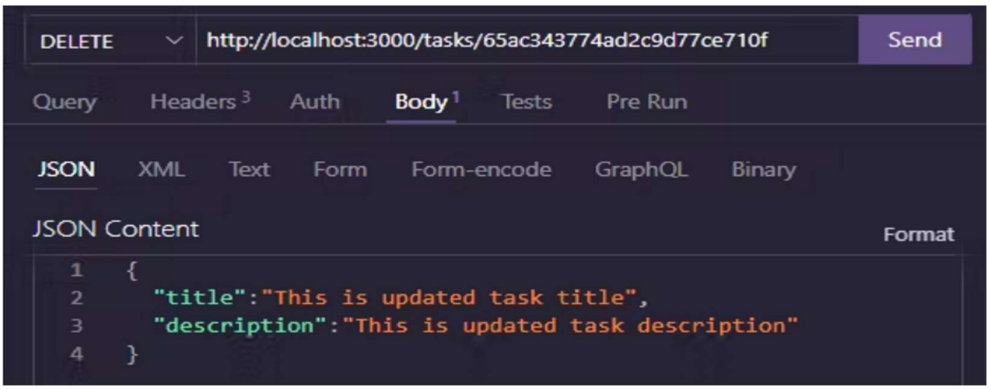


4. DELETE request

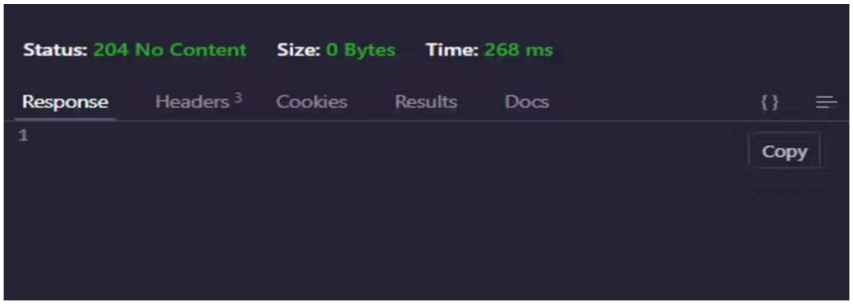
Prior to deleting the task, we will need to know its ID. You need to copy the ID that you got in the last response

```
// Delete a todo by ID
app.delete("/tasks/:id", async (req, res) => {
  try {
    await Task.findByIdAndDelete(req.params.id);
    res.status(204).send();
  } catch (error) {
    res.status(500).json({ error: "Internal Server Error" });
  }
});
```

. Select "DELETE" and write down the URL, as shown in the picture below.



When you click "Send," you'll get the message below.




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

ASSIGNMENT-3

OUTPUT PAGE