

## REST API WITH EXPRESS

**Aim:** To create a REST API with Express.js

### Theory:

#### What is Express?

[Express.js](https://expressjs.com/) is a web application framework for Node.js. It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.

Express.js is based on the Node.js middleware module called connect which in turn uses the http module. So, any middleware which is based on connect will also work with Express.js. Express provides a thin layer of fundamental web application features, without obscuring the Node.js features.

#### APIs with Express

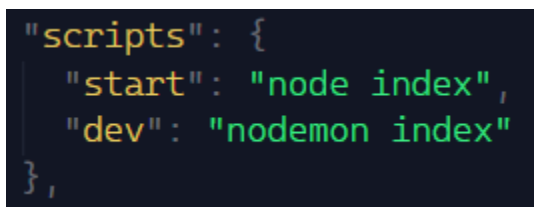
With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

#### Code:

1. Setup Node Project and install dependencies.

```
npm init -y
npm install express express-handlebars uuid
npm install -D nodemon
```

2. Update scripts in package.json to use nodemon to start the server.

A screenshot of a code editor showing the 'scripts' section of a package.json file. The code is as follows:

```
"scripts": {
  "start": "node index",
  "dev": "nodemon index"
},
```

3. Create a file todos.js in the root directory to add some default todos.

```
const todos = [  
  {  
    id: "a3sd0g-322s-see6",  
    task: "Solve Hackerrank problems",  
    status: "active",  
  },  
  {  
    id: "42s9-4d23-sfd1",  
    task: "Complete SaaS Dashboard",  
    status: "inactive",  
  },  
  {  
    id: "1a3b-2aox-113x-gkl2",  
    task: "Learn Graph Search Algorithms",  
    status: "active",  
  },  
];  
  
module.exports = todos;
```

4. Create a file index.js to initialize the application.

```
const express = require("express");  
const { engine } = require("express-handlebars");  
const todos = require("./todos");  
const app = express();  
  
// Handlebars Middleware for HTML templating  
app.engine("handlebars", engine());  
app.set("view engine", "handlebars");  
  
// Body Parser Middleware  
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));  
  
// Homepage Route  
app.get("/", (req, res) =>  
  res.render("index", {  
    title: "Todo List",  
    todos,  
  })  
);  
const PORT = process.env.PORT || 5000;  
app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
```

5. Create a folder views. Inside folder layouts, create main.handlebars. Here, we create HTML template files to display all todos and a button to add a new one.

```
{{!-- ./views/layouts/main.handlebars --}}
<html lang="en">

  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.
min.css"
      integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9J
voRxT2MZw1T"
      crossorigin="anonymous"
    />
    <title>Todo List</title>
  </head>

  <body>
    <div class="container mt-4">
      {{{body}}}
    </div>
  </body>

</html>
```

6. Create index.handlebars in the views folder.

```
{{!-- ./layouts/index.handlebars --}}
<h1 class="text-center mb-3">{{{title}}}</h1>
<form action="/api/todos" method="POST" class="mb-4">
  <div class="form-group">
    <label for="task">Task Name</label>
    <input type="text" name="task" class="form-control" />
  </div>
  <input type="submit" value="Add Task" class="btn btn-primary
btn-block" />
</form>
<h4>To Do List</h4>
```

```
<ul class="list-group">
  {{#each todos}}
    <li class="list-group-item my-2">{{this.task}}</li>
  {{/each}}
</ul>

<a href="/api/todos" class="btn btn-light mt-4">Visit API</a>
```

7. Create a folder routes/api to create the API routes to create, update, insert and delete Todos. Create a file todos.js. In this file, we create routes to get all todos, get a certain todo, create a new todo, update and delete it. We use the UUID library to create new string-based identities.

```
// ./routes/api/todos.js
const express = require("express");
const uuid = require("uuid");
const router = express.Router();
const todos = require("../../todos");

const idFilter = (req) => (todo) => todo.id === req.params.id;

// Gets All Todos
router.get("/", (req, res) => res.json(todos));

// Get Single Todo
router.get("/:id", (req, res) => {
  const found = todos.some(idFilter(req));

  if (found) {
    res.json(todos.filter(idFilter(req)));
  } else {
    res.status(400).json({
      msg: `No todo with the id of ${req.params.id}`,
    });
  }
});

// Create task
router.post("/", (req, res) => {
  const newTask = {
    ...req.body,
    id: uuid.v4(),
    status: "active",
  };
  todos.push(newTask);
  res.json(newTask);
});
```

```
    };

    if (!newTask.task) {
        return res.status(400).json({ msg: "Please include a task
title" });
    }

    todos.push(newTask);
    res.redirect("/");
});

// Update Task
router.put("/:id", (req, res) => {
    const found = todos.some(idFilter(req));

    if (found) {
        todos.forEach((task, i) => {
            if (idFilter(req)(task)) {
                const updTask = { ...task, ...req.body };
                todos[i] = updTask;
                res.json({ msg: "Task updated", updTask });
            }
        });
    } else {
        res.status(400).json({
            msg: `No task with the id of ${req.params.id}`,
        });
    }
});

// Delete Task
router.delete("/:id", (req, res) => {
    const found = todos.some(idFilter(req));

    if (found) {
        res.json({
            msg: "Task deleted",
            todos: todos.filter((task) => !idFilter(req)(task)),
        });
    } else {
        res.status(400).json({
            msg: `No item with the id of ${req.params.id}`,
        });
    }
});
```

```
    }  
  });  
  module.exports = router;
```

8. Now that the routes are set up, append these routes to the main index.js file.

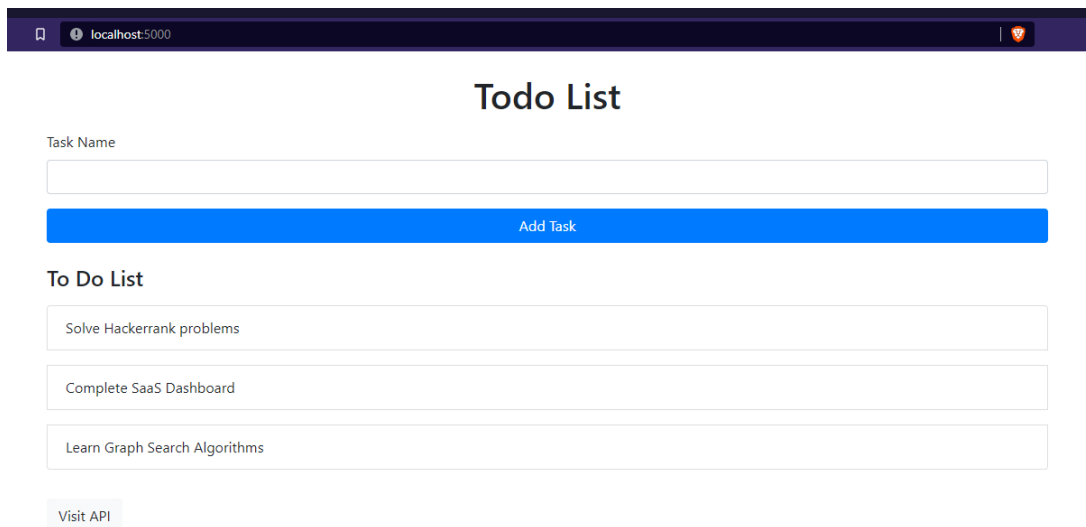
```
// ./index.js  
//Todos API Routes  
app.use("/api/todos", require("./routes/api/todos"));
```

9. Run the server locally by running **npm run dev** in the command line.

```
→ express-api git:(main) npm run dev  
  
> express-api@1.0.0 dev  
> nodemon index  
  
[nodemon] 2.0.15  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *,*  
[nodemon] watching extensions: js,mjs,json  
[nodemon] starting `node index index.js`  
Server started on port 5000
```

10. You can view the frontend in localhost:5000

You can add tasks by writing something in Task Name and hitting 'Add Task'



localhost:5000

## Todo List

Task Name

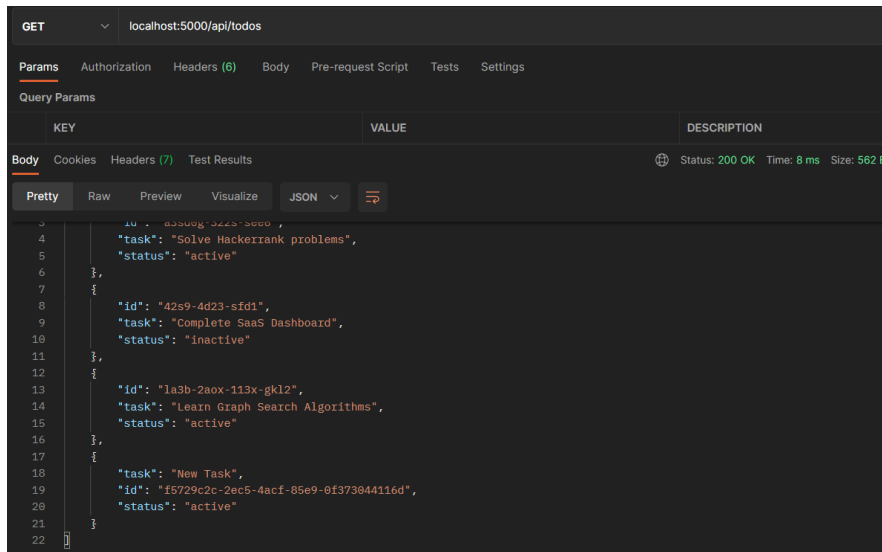
Add Task

### To Do List

- Solve Hackerrank problems
- Complete SaaS Dashboard
- Learn Graph Search Algorithms

Visit API

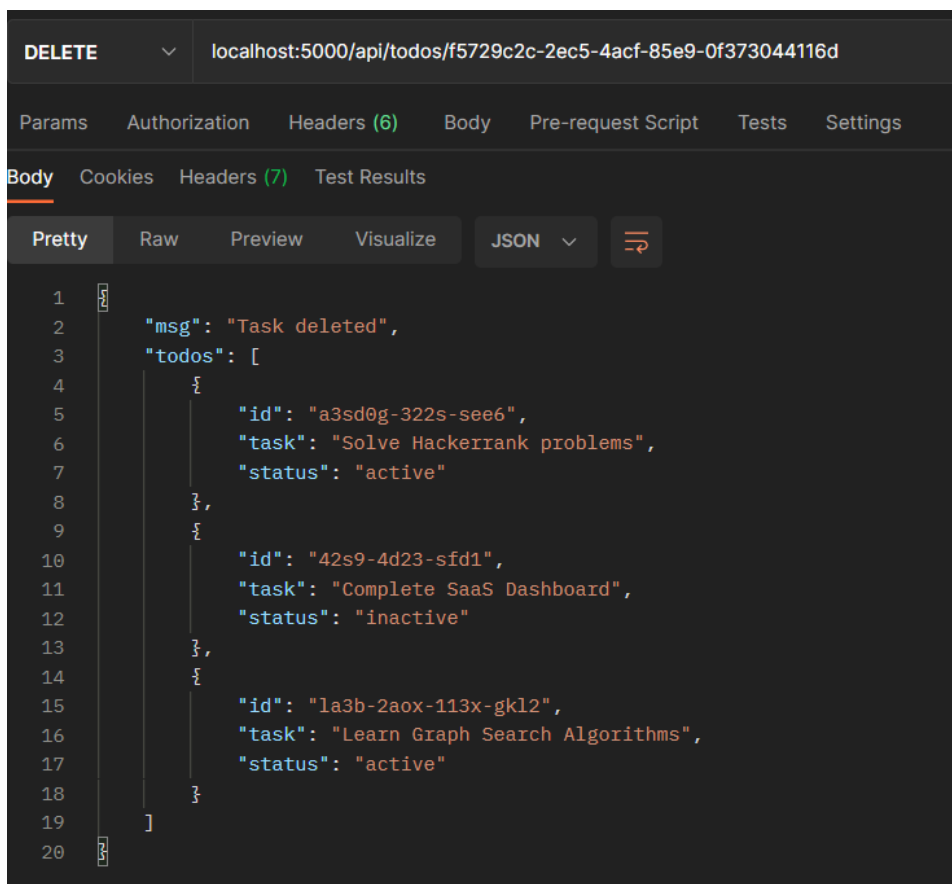
11. You can send a GET request to /api/todos to get this list of tasks.



The screenshot shows a REST client interface with a GET request to `localhost:5000/api/todos`. The response is a JSON array of four tasks, each with an `id`, `task`, and `status`.

```
3  {
4    "id": "a3sd0g-322s-see6",
5    "task": "Solve Hackerrank problems",
6    "status": "active"
7  },
8  {
9    "id": "42s9-4d23-sfd1",
10   "task": "Complete SaaS Dashboard",
11   "status": "inactive"
12 },
13 {
14   "id": "1a3b-2aox-113x-gkl2",
15   "task": "Learn Graph Search Algorithms",
16   "status": "active"
17 },
18 {
19   "task": "New Task",
20   "id": "f5729c2c-2ec5-4acf-85e9-0f373044116d",
21   "status": "active"
22 }
```

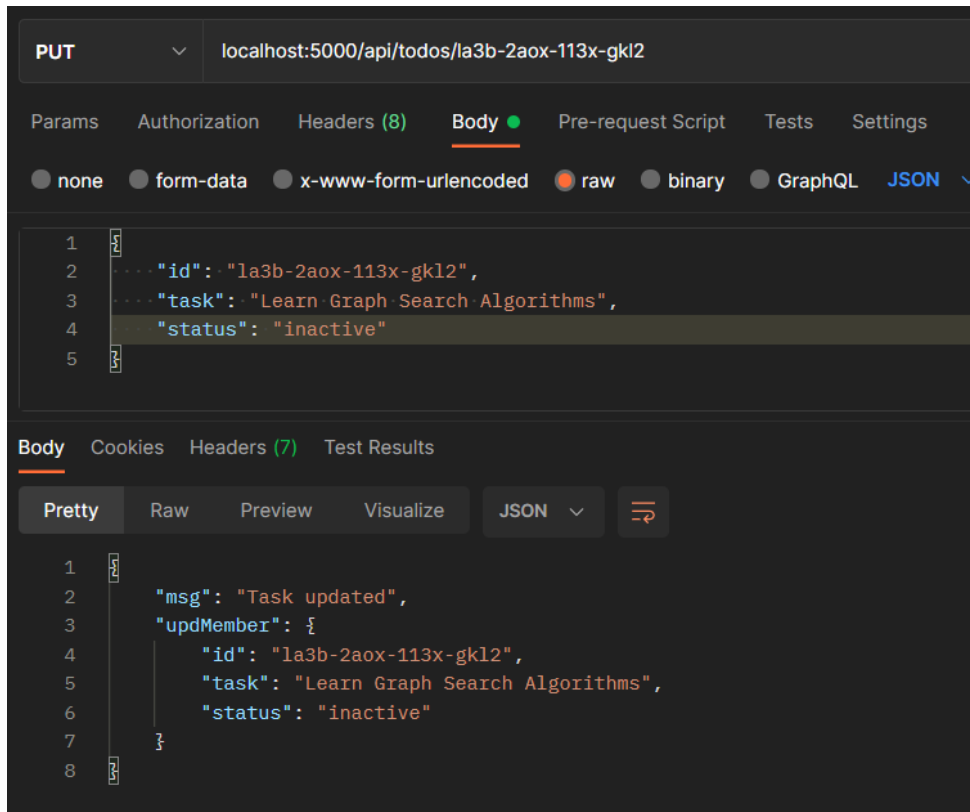
12. You can send a DELETE Request to /api/todos with a Valid ID to delete that particular To-do.



The screenshot shows a REST client interface with a DELETE request to `localhost:5000/api/todos/f5729c2c-2ec5-4acf-85e9-0f373044116d`. The response is a JSON object with a `msg` field and a `todos` array containing three tasks.

```
1  {
2    "msg": "Task deleted",
3    "todos": [
4      {
5        "id": "a3sd0g-322s-see6",
6        "task": "Solve Hackerrank problems",
7        "status": "active"
8      },
9      {
10       "id": "42s9-4d23-sfd1",
11       "task": "Complete SaaS Dashboard",
12       "status": "inactive"
13     },
14     {
15       "id": "1a3b-2aox-113x-gkl2",
16       "task": "Learn Graph Search Algorithms",
17       "status": "active"
18     }
19   ]
20 }
```

13. You can send a PUT request to `/api/todos/1a3b-2aox-113x-gkl2` with a valid ID and valid Body to update that particular To-do.



### Conclusion:

In this way, we can create a well-structured REST API with Express.js and send requests to it to perform different operations. We can also use templating engines like handlebars to send dynamic data from the API.