### Smaller projects

## 11. Implement Web Sockets for real-time communication in your application (Implement Socket.io library )

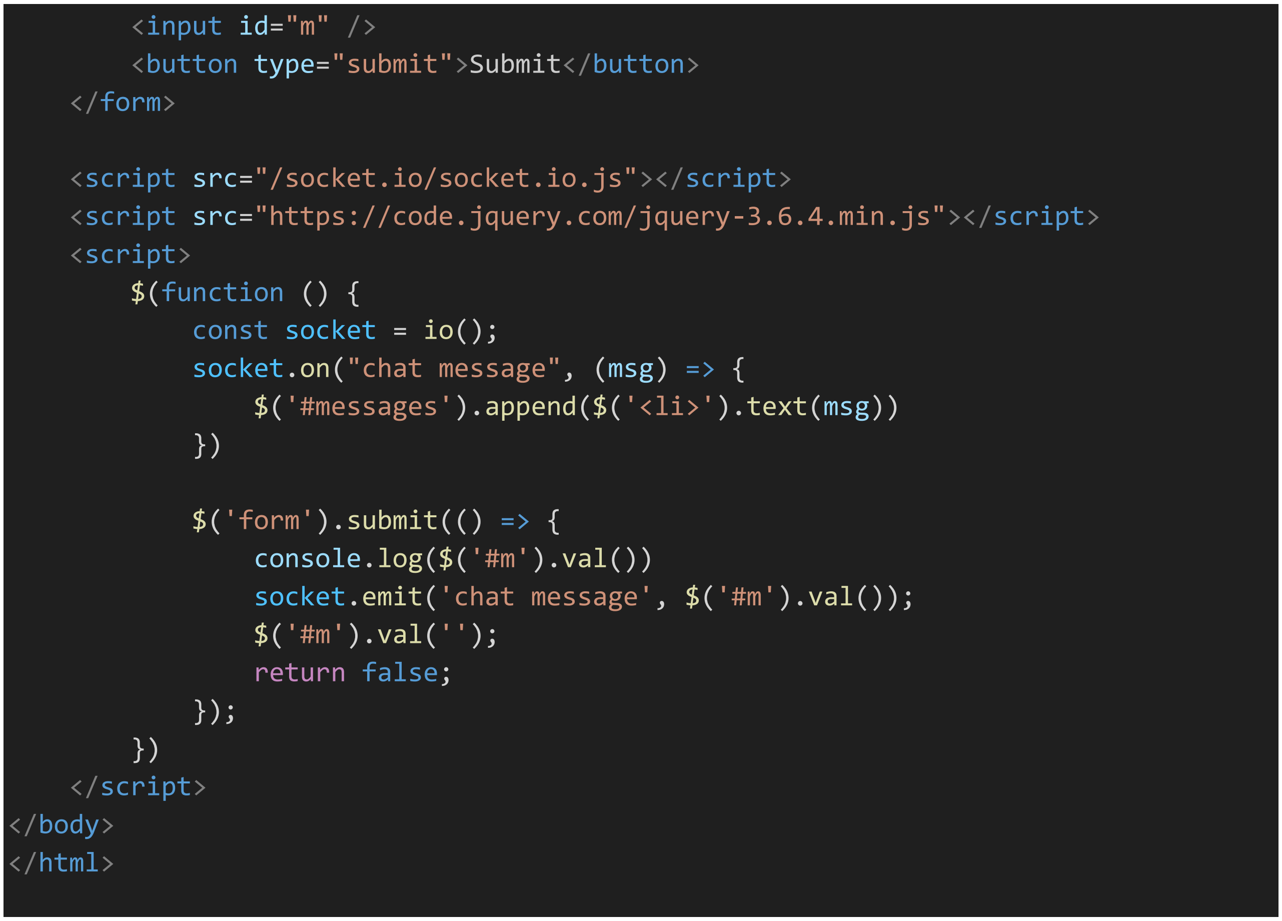
### Code:

### Index.js

```
import express from 'express';
import http from 'http';
import {Server as socketIO} from 'socket.io';

const app = express()
const server = http.createServer(app)
const io = new socketIO(server)

app.use(express.static('public'))

io.on("connection", (socket) => {
    console.log("a user connected ");

    socket.on("chat message", (msg) => {
        console.log("message: " + msg);

        io.emit("chat message", msg)
    })

    socket.on("disconnect",() => {
        console.log("user disconnected")
    })
})

const PORT = 5000
server.listen(PORT, () => {
    console.log(`server is running on ${PORT}`)
})
```

### Public/ index.html

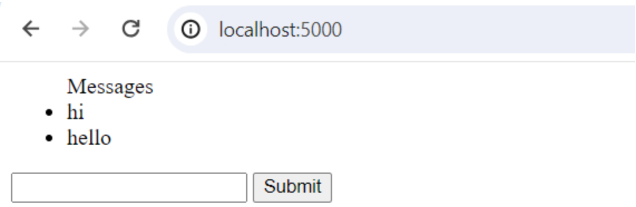
```
<html>

<head>
    <title>Socket IO real time communcation</title>
</head>

<body>
    <ul id="messages">Messages</ul>
    <form id="form" action="">
```



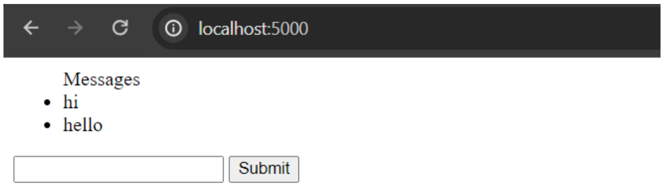
```
        <input id="m" />
        <button type="submit">Submit</button>
    </form>

    <script src="/socket.io/socket.io.js"></script>
    <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
    <script>
        $(function () {
            const socket = io();
            socket.on("chat message", (msg) => {
                $('#messages').append($('<li>').text(msg))
            })

            $('form').submit(() => {
                console.log($('#m').val())
                socket.emit('chat message', $('#m').val());
                $('#m').val('');
                return false;
            });
        })
    </script>
</body>
</html>
```

Run node index.js

**Output:**



Messages
- hi
- hello



Messages
- hi
- hello

### 15. Read and Parse a JSON file then display its contents in structured format

**Code:**

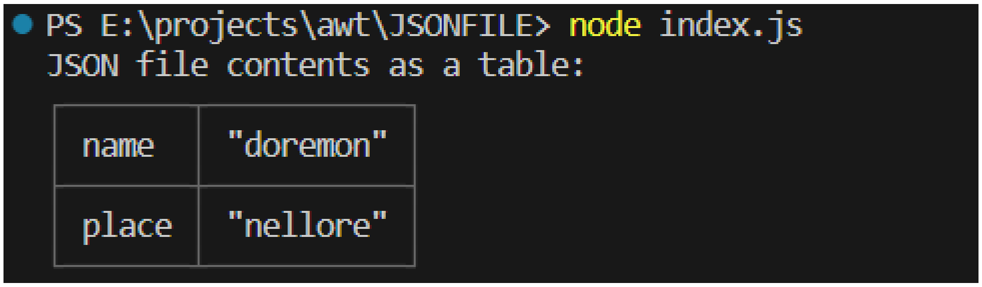
**Index.js**

```
import fs from 'fs/promises';
import Table from 'cli-table';

// Step 1: Read the JSON file
const filePath = 'sample.json';

try {
  const jsonData = await fs.readFile(filePath, 'utf-8');

  // Step 2: Parse the JSON data
  const data = JSON.parse(jsonData);

  // Step 3: Display the contents in a table format
  const table = new Table();

  for (const key in data) {
    if (Object.prototype.hasOwnProperty.call(data, key)) {
      table.push([key, JSON.stringify(data[key])]);
    }
  }

  console.log('JSON file contents as a table:');
  console.log(table.toString());
} catch (error) {
  if (error.code === 'ENOENT') {
    console.error(`Error: File not found at path '${filePath}'`);
  } else {
    console.error(`An unexpected error occurred: ${error.message}`);
  }
}
```

**Sample.json**

```
{
    "name" : "doremon",
    "place" : "nellore"
}
```

**Output:**



```
PS E:\projects\awt\JSONFILE> node index.js
JSON file contents as a table:

  name    "doremon"

  place   "nellore"
```

### Bigger project

1. **Develop the important backend (Node Js, Express Js, MongoDB) functionalities for a task manager applications**

### Code

### Index.js

```
import express from 'express';
import mongoose from 'mongoose'
import bodyParser from 'body-parser';
import methodOverride from 'method-override'

const app = express();
const PORT = 3000;

// MongoDB connection
mongoose.connect('mongodb+srv://Misbah_Khanam:Misbah123@cluster0.jfyyvte.mongodb.net/?retryWrites=true&w=majority', { useNewUrlParser: true, useUnifiedTopology: true });

// Middleware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.set('view engine', 'ejs');

app.use(methodOverride('_method'));

// Task model
const Task = mongoose.model('Task', {
  title: String,
  description: String,
});

// Routes
app.get('/', async (req, res) => {
  const tasks = await Task.find();
  res.render('index', { tasks });
});

app.post('/tasks', async (req, res) => {
  const { title, description } = req.body;
  const task = new Task({ title, description });
  await task.save();
  res.redirect('/');
});
```

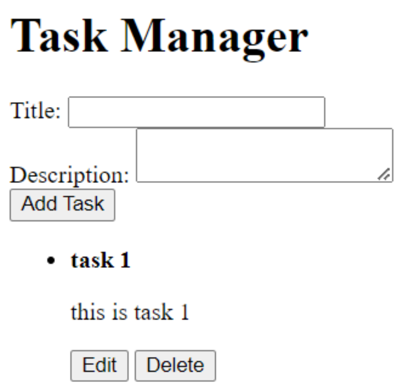
```
app.put('/tasks/:id', async (req, res) => {
  const { id } = req.params;
  const { title, description } = req.body;
  await Task.findByIdAndUpdate(id, { title, description });
  res.redirect('/');
});

app.delete('/tasks/:id', async (req, res) => {
  const { id } = req.params;
  await Task.findByIdAndDelete(id);
  res.json({ message: 'Task deleted successfully' });
});

// Start server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

## View/index.ejs

```
<!-- views/index.ejs -->

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Task Manager</title>
</head>
<body>
  <h1>Task Manager</h1>
  <form method="POST" action="/tasks">
    <label for="title">Title:</label>
    <input type="text" id="title" name="title" required>
    <br>
    <label for="description">Description:</label>
    <textarea id="description" name="description" required></textarea>
    <br>
    <button type="submit">Add Task</button>
  </form>

  <ul id="taskList">
    <% tasks.forEach(task => { %>
      <li>
        <strong><%= task.title %></strong>
        <p><%= task.description %></p>
        <form method="POST" action="/tasks/<%= task._id %>?_method=PUT"
style="display:inline;">
          <button type="button" onclick="editTask('<%= task._id %>', '<%=
task.title %>', '<%= task.description %>')">Edit</button>
        </form>
        <form method="POST" action="/tasks/<%= task._id %>?_method=DELETE"
style="display:inline;">
          <button type="submit">Delete</button>
        </form>
      </li>
    <% }); %>
  </ul>

  <div id="editTaskForm" style="display:none;">
    <h2>Edit Task</h2>
    <form id="editForm">
      <label for="editTitle">Title:</label>
      <input type="text" id="editTitle" name="editTitle" required>
      <br>
      <label for="editDescription">Description:</label>
      <textarea id="editDescription" name="editDescription" required></textarea>
      <br>
      <button type="button" onclick="updateTask()">Update Task</button>
      <button type="button" onclick="cancelEdit()">Cancel</button>
    </form>
  </div>

  <script>

    async function editTask(id, title, description) {
      document.getElementById('editTitle').value = title;
      document.getElementById('editDescription').value = description;

      // Display the edit form and hide the task list
      document.getElementById('taskList').style.display = 'none';
      document.getElementById('editTaskForm').style.display = 'block';

      // Set up the updateTask function to use the task ID
      window.updateTask = async () => {
        const editTitle = document.getElementById('editTitle').value;
        const editDescription = document.getElementById('editDescription').value;

        const response = await fetch(`/tasks/${id}?_method=PUT`, {
          method: 'POST', // Using POST to simulate PUT because HTML forms only
support GET and POST
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify({ title: editTitle, description: editDescription }),
        });

        if (response.ok) {
          window.location.reload()
        } else {
          console.error('Error updating task');
        }
      };

      // Set up the cancelEdit function to hide the edit form and show the task list
      window.cancelEdit = () => {
        document.getElementById('taskList').style.display = 'block';
        document.getElementById('editTaskForm').style.display = 'none';
      };
    }

  </script>
</body>
</html>
```

## OUTPUT:

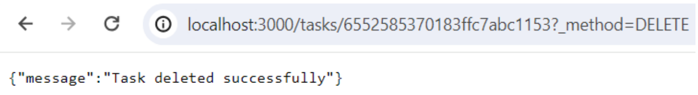
## Add task



- **task 1**

  this is task 1

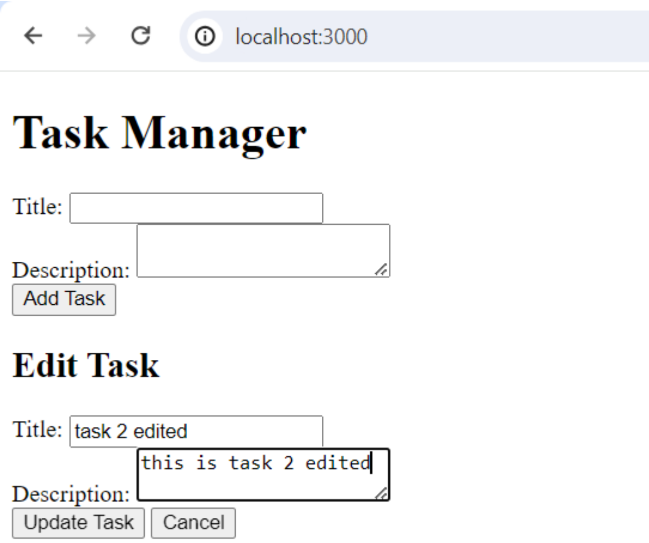
## Delete



localhost:3000/tasks/6552585370183ffc7abc1153?_method=DELETE

```
{"message":"Task deleted successfully"}
```

## Edit



Task Manager

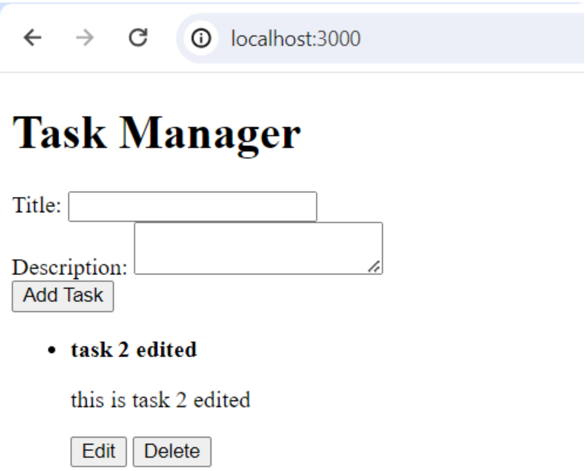
Title:

Description:

Edit Task

Title: task 2 edited

Description: this is task 2 edited



Task Manager

Title:

Description:

- **task 2 edited**

  this is task 2 edited