Backend (Node + Express + MongoDB)

---

1. Import Libraries

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

Imports Express framework
Used to create server and API routes

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

Imports Mongoose
Helps connect Node.js with MongoDB

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

Allows frontend (React) to communicate with backend
Without this, browser blocks requests from different ports

```
const app = express();
```

Creates Express application
app will handle routes and server

```
app.use(cors());
```

Enables Cross-Origin Resource Sharing
Allows React app to call backend API

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

Converts incoming JSON data into JavaScript object
Required to read data from request body

---

2. MongoDB Connection

```
mongoose.connect("mongodb://127.0.0.1:27017/taskdb")
```

Connects to MongoDB
taskdb is database name

.then(() => console.log("MongoDB Connected"))

Runs when connection is successful

.catch((err) => console.log(err));

Runs if connection fails

---

3. Schema Creation

const TaskSchema = new mongoose.Schema({

Creates structure of database document

---

taskName: { type: String, required: true },

taskName field
Must be String
Cannot be empty

---

status: { type: String, default: "Pending" }

status field
Default value = Pending

});

Ends schema

---

4. Model Creation

const TaskModel = mongoose.model("tasks", TaskSchema);

Creates collection named **tasks** in MongoDB
Uses schema to store data

CRUD OPERATIONS

CREATE Operation

app.post("/tasks", async (req, res) => {

Creates POST API route
/tasks is endpoint
req = request from frontend
res = response to frontend

---

const newTask = new TaskModel(req.body);

Creates new task object
Takes data from frontend

---

await newTask.save();

Saves data into MongoDB
await waits until save completes

---

res.json(newTask);

Sends saved data back to frontend

});

Ends API route

READ Operation

app.get("/tasks", async (req, res) => {

GET route to fetch tasks

---

const tasks = await TaskModel.find();

Finds all documents from tasks collection

---

res.json(tasks);

Sends tasks data to frontend

```
});
```

Ends route

UPDATE Operation

```
app.put("/tasks/:id", async (req, res) => {
```

PUT route
:id means dynamic task id

```
await TaskModel.findByIdAndUpdate(
```

Finds task by ID and updates

```
req.params.id,
```

Gets ID from URL

```
req.body
```

Updated data from frontend

```
);
```

Ends update function

```
res.json({ message: "Updated" });
```

Sends confirmation message

```
});
```

Ends route

DELETE Operation

```
app.delete("/tasks/:id", async (req, res) => {
```

DELETE route

```
await TaskModel.findByIdAndDelete(req.params.id);
```

Deletes task using ID

```
res.json({ message: "Deleted" });
```

Sends delete confirmation

```
});
```

Ends route

Server Start

```
app.listen(5000, () => {
```

Runs server on port 5000

```
console.log("Server running");
```

Prints message when server starts

```
});
```

Ends server setup

Frontend (React) Line-wise Explanation

CREATE (Add Task)

```
const addTask = async () => {
```

Creates function to add task

```
await fetch("http://localhost:5000/tasks", {
```

Sends request to backend

```
method: "POST",
```

Specifies create operation

```
headers: {
  "Content-Type": "application/json"
}
```

Tells backend data is JSON

```
body: JSON.stringify({ taskName: task })
```

Converts task into JSON string

```
});
```

Ends fetch request

READ (Fetch Tasks)

```
useEffect(() => {
```

Runs when page loads

```
fetch("http://localhost:5000/tasks")
```

Calls backend GET API

```
.then(res => res.json())
```

Converts response to JSON

```
.then(data => setTasks(data));
```

Stores tasks in React state

```
}, []);
```

Empty array means run only once

UPDATE

const updateTask = async (id) => {

Function to update task

---

await fetch(`http://localhost:5000/tasks/${id}`, {

Sends request with task ID

---

method: "PUT",

Specifies update operation

---

body: JSON.stringify({ status: "Completed" })

Sends updated data

});

Ends request

DELETE

const deleteTask = async (id) => {

Function to delete task

---

await fetch(`http://localhost:5000/tasks/${id}`, {

Calls delete API

---

method: "DELETE"

Specifies delete operation

```
});
```

Ends request