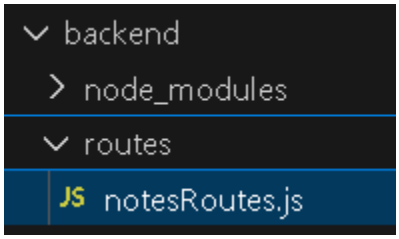
Best practice for routes:

Initial code in server.js:

```
import express from "express"
const app = express()

// http get
app.get("/api/notes", (req, res) => {
    res.status(200).send("You got 5 notes")
})
// http post
app.post("/api/notes", (req, res) => {
    // Gets data in form of json
    res.status(201).json({message: "Note created successfully!"})
})

// :id <- is dynamic id
// Eg: http://localhost:5001/api/notes/2312 <- where id = 2312
// http put
app.put("/api/notes/:id", (req, res) => {
    res.status(200).json({message: "Note updated successfully!"})
})
// http delete
app.delete("/api/notes/:id", (req, res) => {
    res.status(200).json({message: "Note deleted successfully!"})
})

app.listen(5001, () => {
    console.log("Server started on PORT: 5001...")
})
```

If each route has many number of codes. Then its looks complex.
So managing it is very important, like dividing based on common preferences.
Create a routes folder in the backend, and routes having notesRoutes.js file:



Code in server.js:

```
import express from "express"
```

```
import notesRoutes from "./routes/notesRoutes.js"
const app = express()

app.use("/api/notes", notesRoutes)

app.listen(5001, () => {
    console.log("Server started on PORT: 5001...")
})
```

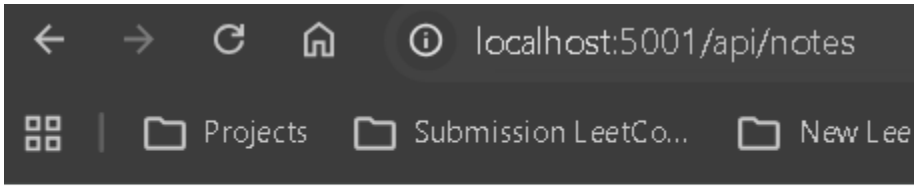
Code in notesRoutes.js:

```
import express from "express"
const router = express.Router()

// "/api/notes" exists in -> app.use("/api/notes", notesRoutes)
router.get("/", (req, res) => {
    res.status(200).send("You just fetched the notes")
})
// http get
// app.get("/api/notes", (req, res) => {
//     res.status(200).send("You got 5 notes")
// })

router.post("/", (req, res) => {
    // Gets data in form of json
    res.status(201).json({message: "Note created successfully!"})
})
// // http post
// app.post("/api/notes", (req, res) => {
//     // Gets data in form of json
//     res.status(201).json({message: "Note created successfully!"})
// })

router.put("/:id", (req, res) => {
    res.status(200).json({message: "Note updated successfully!"})
})
// // :id <- is dynamic id
// // Eg: http://localhost:5001/api/notes/2312 <- where id = 2312
// // http put
// app.put("/api/notes/:id", (req, res) => {
//     res.status(200).json({message: "Note updated successfully!"})
// })

router.delete("/:id", (req, res) => {
```

```
    res.status(200).json({message: "Note deleted successfully!"})
})
// // http delete
// app.delete("/api/notes/:id", (req, res) => {
//     res.status(200).json({message: "Note deleted successfully!"})
// })

export default router;
```

Run the code:

```
PS C:\Users\kiran\OneDrive\Desktop\mern-thinkboard\backend> npm run dev

> backend@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server started on PORT: 5001...
```



You just fetched the notes <-Still works the same!

If each of the route has larger code:
Eg:

```
router.put("/:id", (req, res) => {
    res.status(200).json({message: "Note updated successfully!"})
    // having codes here
    // having codes here
    // having codes here
    // having codes here
    // having codes here
    // having codes here
    // ends here
})
```

We can also manage this complexity by controllers.

Code in notesRoutes.js:

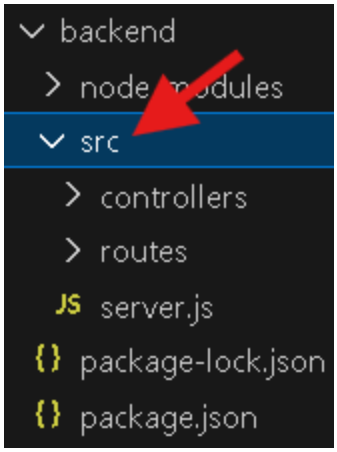
```
import express from "express"
import { getAllNotes, createNote, updateNote, deleteNote } from
"../controllers/notesController.js"
const router = express.Router()

router.get("/", getAllNotes)
router.post("/", createNote)
router.put("/:id", updateNote)
router.delete("/:id", deleteNote)

export default router;
```

Code in notesController.js:

```
export const getAllNotes = (req, res) => {
   res.status(200).send("You just fetched the notes")
}
/*// Other way of writing above
export function getAllNotes(req, res) {
   res.status(200).send("You just fetched the notes")
}*/

export function createNote(req, res) {
   // Gets data in form of json
   res.status(201).json({message: "Note created successfully!"})
}

export function updateNote(req, res) {
   res.status(200).json({message: "Note updated successfully!"})
}

export function deleteNote(req, res) {
   res.status(200).json({message: "Note deleted successfully!"})
}
```

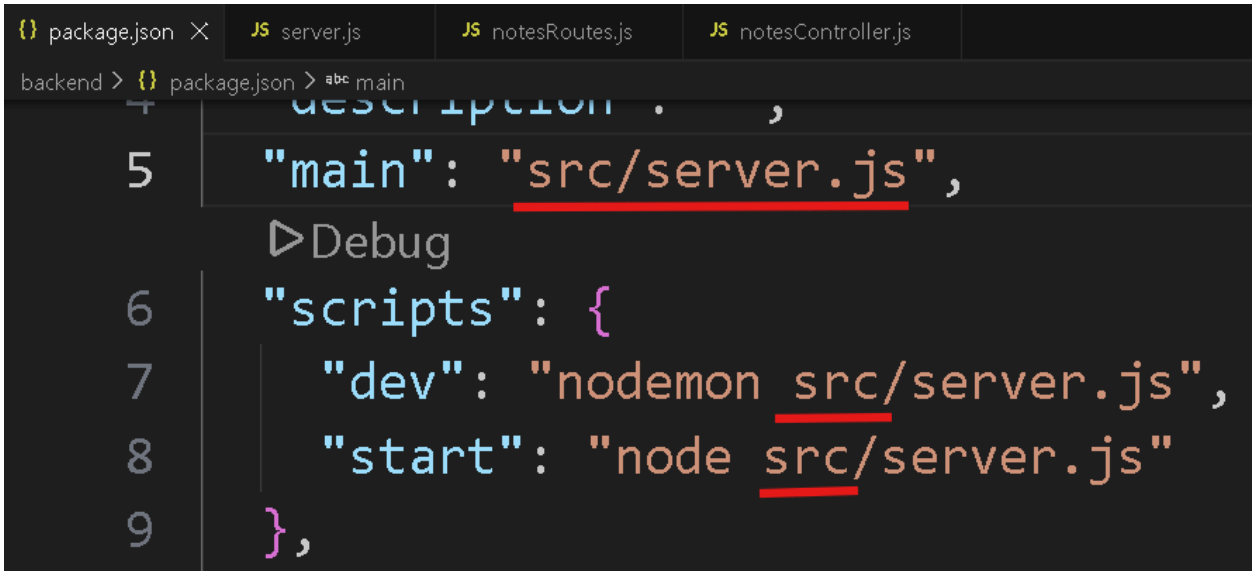
Best practice for folder structure:

Create src folder within backend, drag and drop **controllers, routes and server.js** in the src folder which is created in backend folder.




```
Node.js v20.12.2
[nodemon] app crashed - waiting for file changes before starting...
```

To remove the error:



Stop the server.
And rerun the server.

```
[nodemon] app crashed - waiting for file changes before starting...
Terminate batch job (Y/N)? y
PS C:\Users\kiran\OneDrive\Desktop\mern-thinkboard\backend> npm run dev

> backend@1.0.0 dev
> nodemon src/server.js

[nodemon] 3.1.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/server.js`
Server started on PORT: 5001...
```