Lesson 09 Demo 02

Using Middleware in Express.js App

Objective: To use middleware in the Express.js app for enhanced request handling and functionality

Tools Required: Visual Studio, Node.js, and Express.js

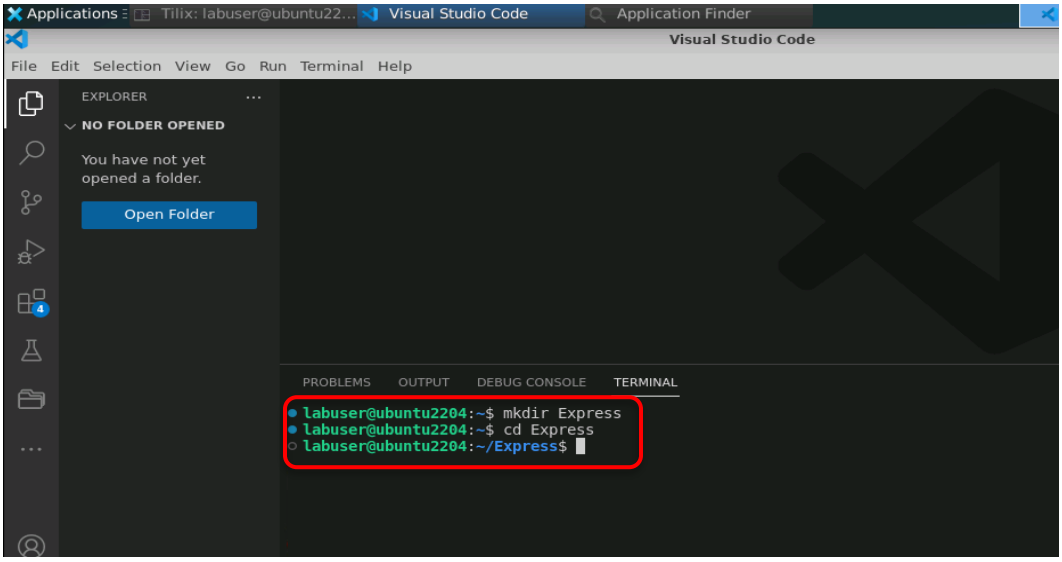
Prerequisites: Knowledge of JavaScript and Node.js

Steps to be followed:

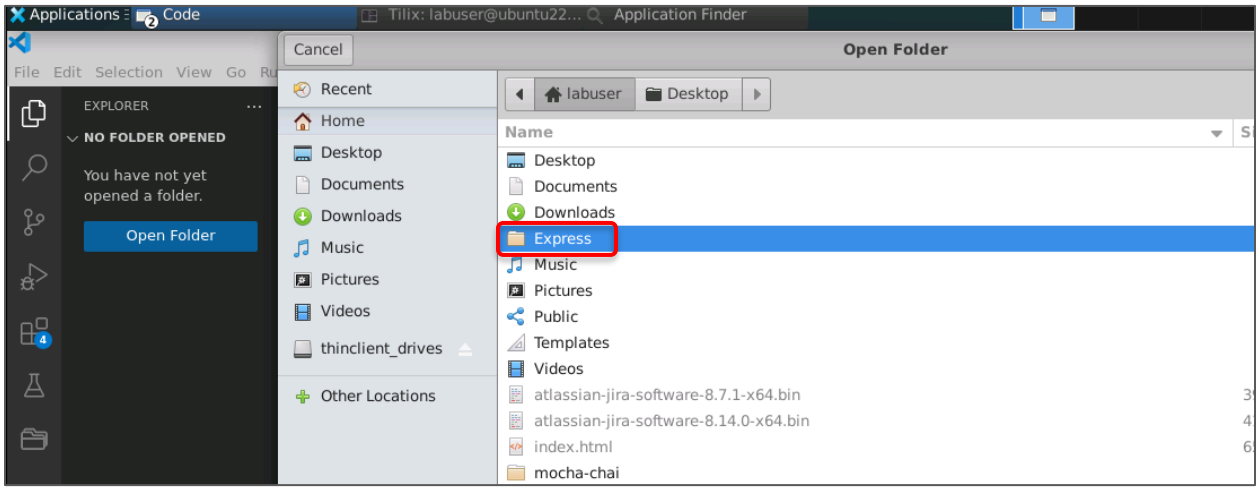
1. Set up Express.js
2. Demonstrate application-level middleware in Express.js
3. Demonstrate router-level middleware in Express.js
4. Demonstrate error-handling middleware in Express.js
5. Demonstrate third-party middleware in Express.js

Step 1: Set up Express.js

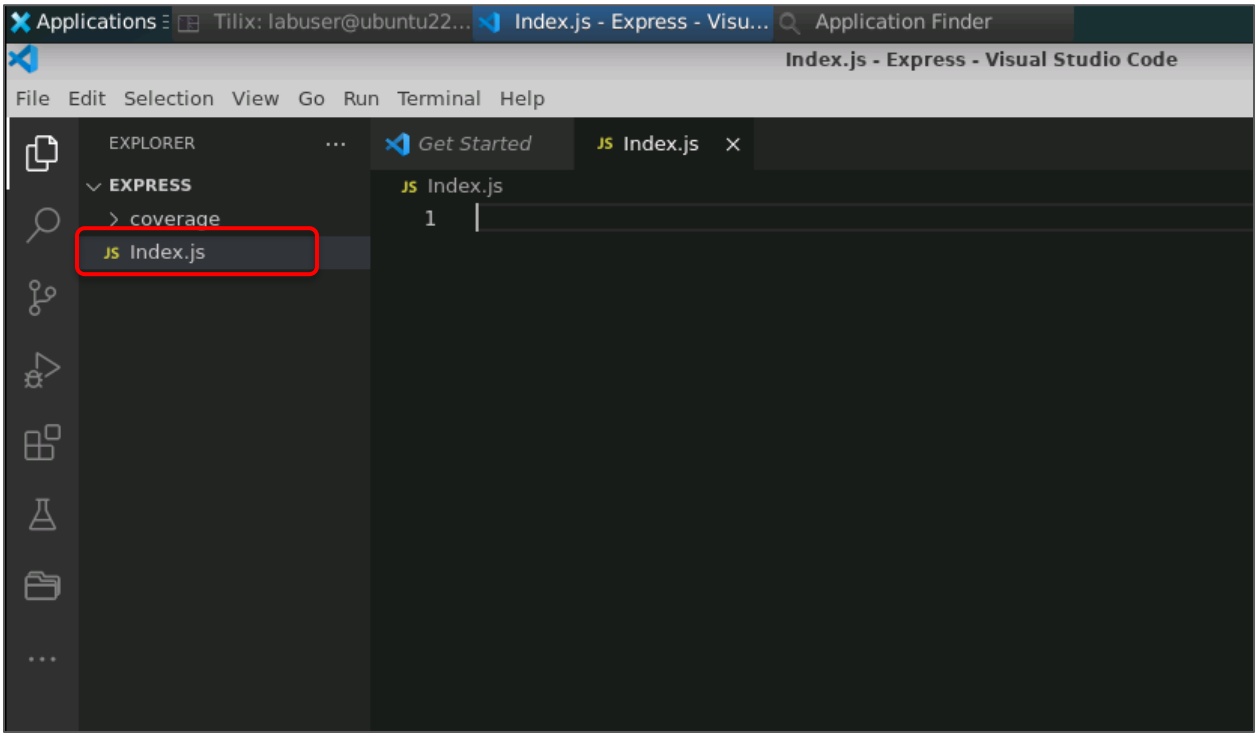
1.1 Open VS Code, create a folder named Express.js with the command **mkdir Express**, and change the current working directory using cd Express



1.2 Open the **Express** folder in the VS Code



1.3 Create an **index.html** file

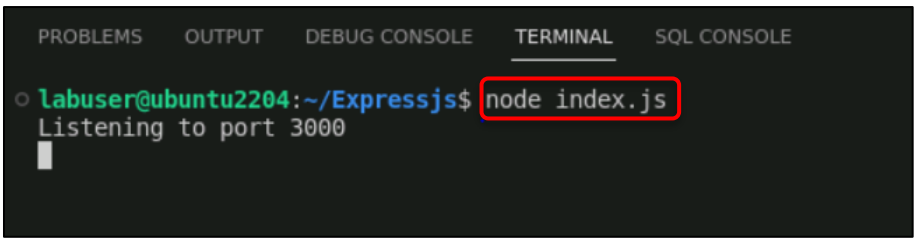


Step 2: Demonstrate application-level middleware in Express.js

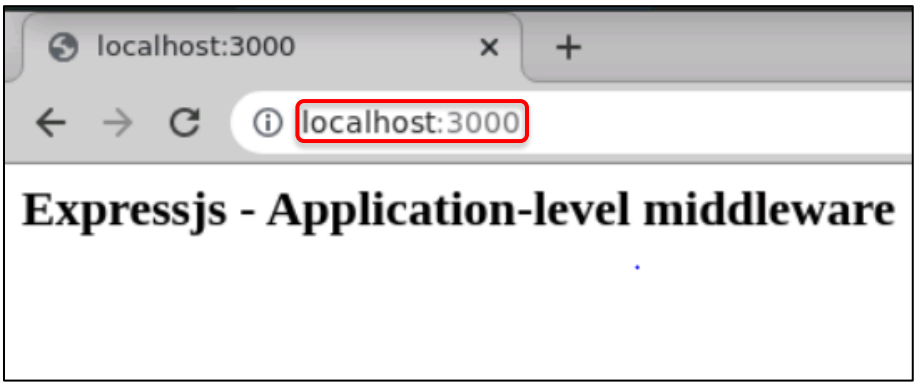
2.1 Add the following code in the **index.js** file:

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

const port = process.env.port || 3000;
app.get(
  "/",
  (req, res, next) => {
    console.log("hello");
    next();
  },
  (req, res) => {
    res.send(`<div>
    <h2>Expressjs - Application-level middleware</h2>
  </div>`);
  }
 );
app.listen(port, () => {
console.log(`Listening to port ${port}`);
});
```

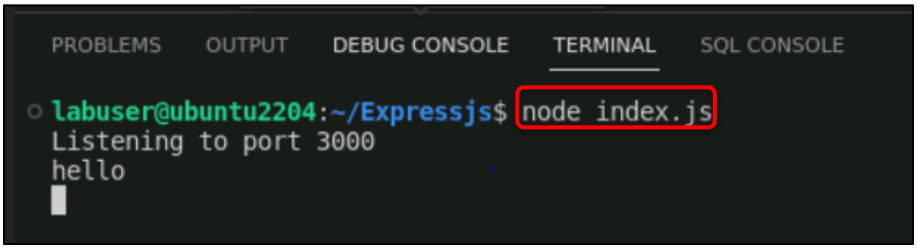
2.2 Run the **node index.js** command in the terminal and get the output



2.3 Open the browser and browse **http://localhost:3000**



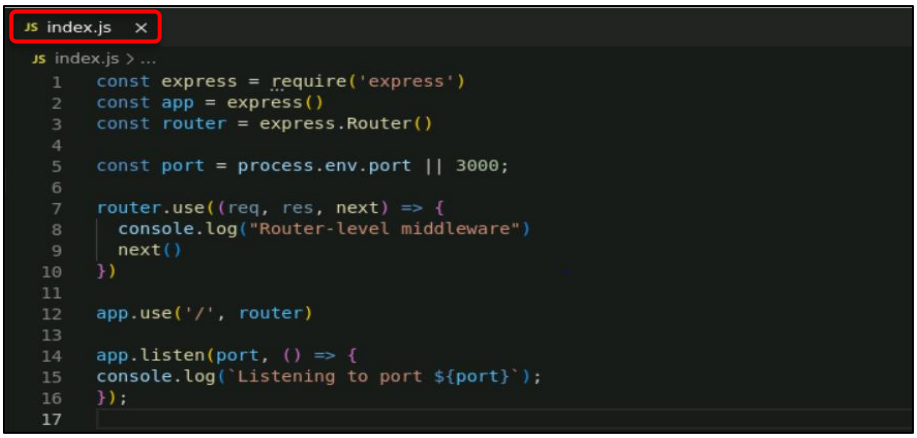
2.4 Check the terminal



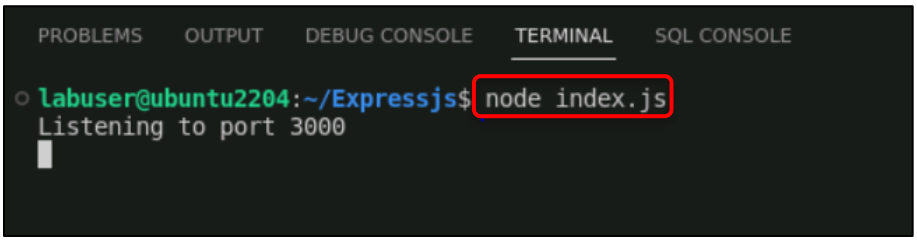
Step 3: Demonstrate router-level middleware in Express.js

3.1 Write the following code in the **index.js** file:

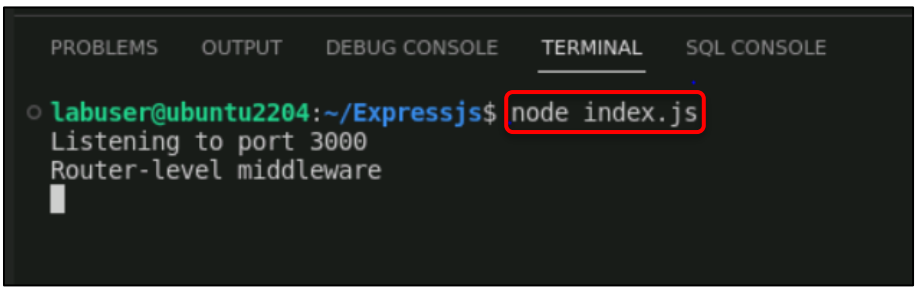
```
const express = require('express')
const app = express()
const router = express.Router()

const port = process.env.port || 3000;

router.use((req, res, next) => {
  console.log("Router-level middleware")
  next()
})

app.use('/', router)

app.listen(port, () => {
console.log(`Listening to port ${port}`);
});
```



3.2 Run the **node index.js** command in the terminal and verify the output



3.3 Open the browser, browse **http://localhost:3000**, and check the terminal again

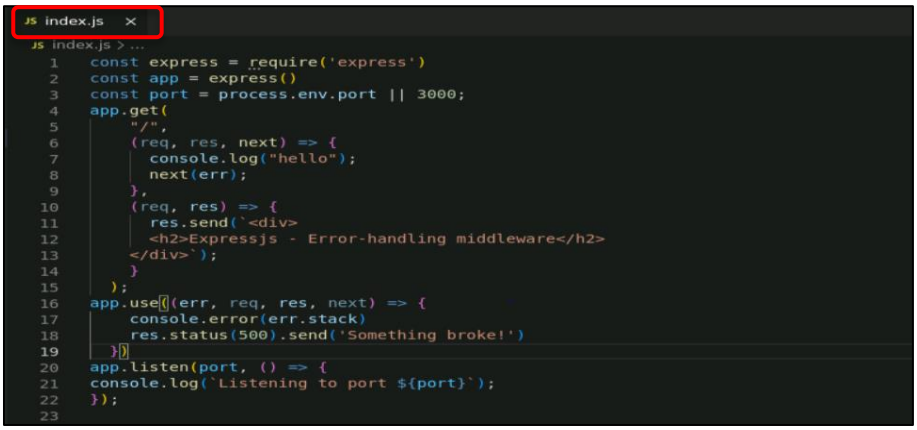


Step 4: Demonstrate error-handling middleware in Express.js

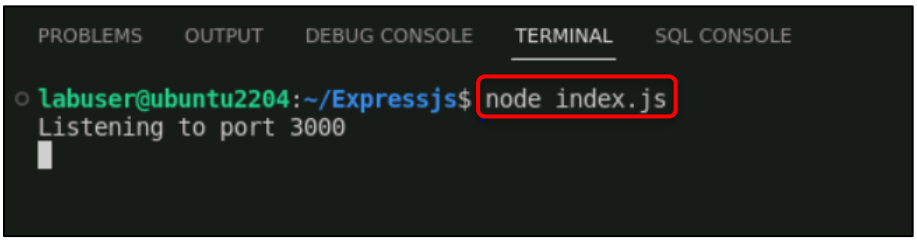
4.1 Add the following code in the **index.js** file:

```
const express = require('express')
const app = express()

const port = process.env.port || 3000;

app.get(
  "/",
  (req, res, next) => {
    console.log("hello");
```

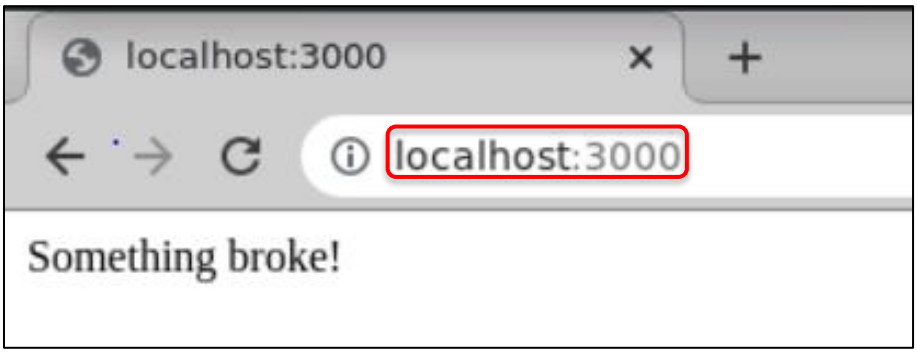
```
    next(err);
  },
  (req, res) => {
    res.send(`<div>
    <h2>Expressjs - Error-handling middleware</h2>
  </div>`);
  }
 );

app.use((err, req, res, next) => {
  console.error(err.stack)
  res.status(500).send('Something broke!')
 })

app.listen(port, () => {
console.log(`Listening to port ${port}`);
});
```

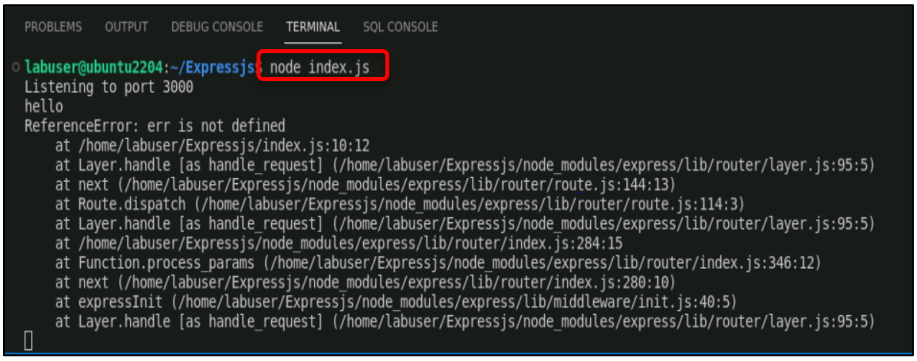


4.2 Run the **node index.js** command in the terminal and get the output



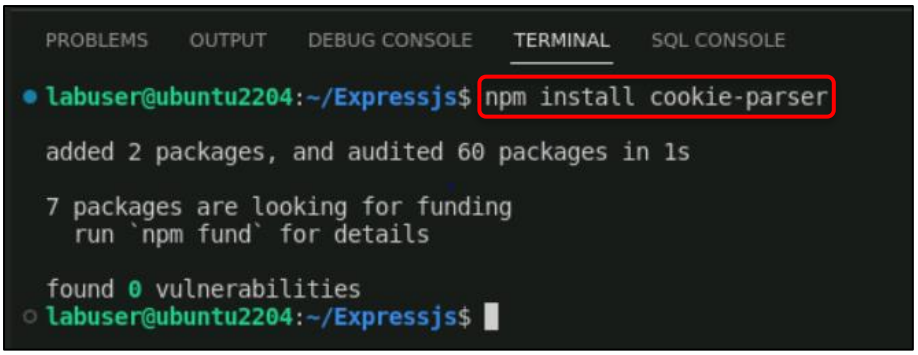
4.3 Open the browser, browse **http://localhost:3000**, and check the terminal again





Step 5: Demonstrate third-party middleware in Express.js

5.1 Install third-party middleware, use the following command, and run it in the terminal:
npm install cookie-parser

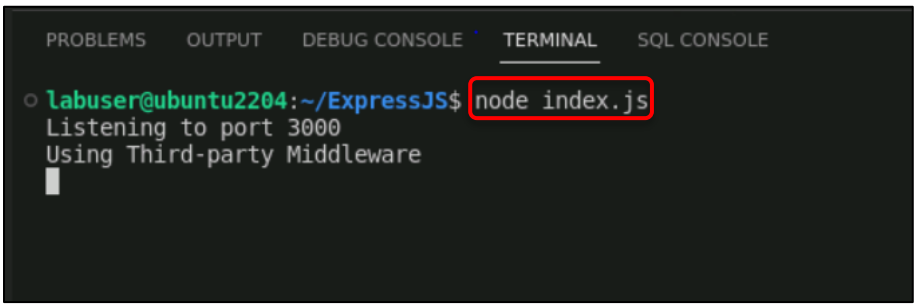


5.2 Add the following code in the **index.js** file, which uses third-party middleware:

```
const express = require('express')
const app = express()
const cookieParser = require('cookie-parser')

const port = process.env.port || 3000;
```

```
// load the cookie-parsing middleware
app.use(cookieParser())

app.listen(port, () => {
   console.log(`Listening to port ${port}`);
   console.log('Using Third-party Middleware.')
  });
```

5.3 Run the **node index.js** command in the terminal and verify the output



By following these steps, you have effectively utilized diverse middleware in an **Express.js** app to enhance request handling and functionality.