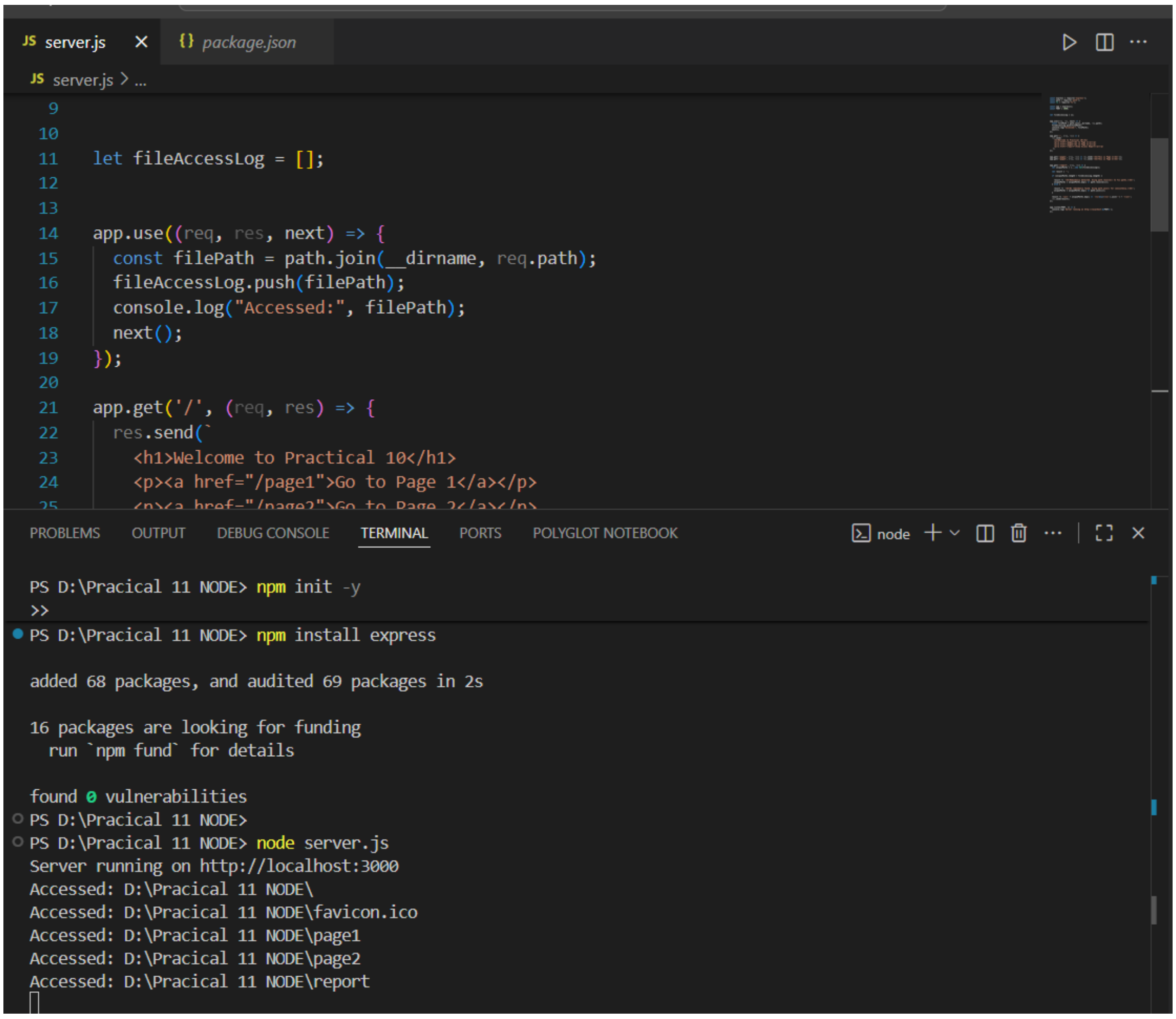
Practical 11-



server.js   package.json

server.js > ...

```
let fileAccessLog = [];


app.use((req, res, next) => {
  const filePath = path.join(__dirname, req.path);
  fileAccessLog.push(filePath);
  console.log("Accessed:", filePath);
  next();
});

app.get('/', (req, res) => {
  res.send(`
    <h1>Welcome to Practical 10</h1>
    <p><a href="/page1">Go to Page 1</a></p>
```

```
PS D:\Pracical 11 NODE> npm init -y
>>
PS D:\Pracical 11 NODE> npm install express

added 68 packages, and audited 69 packages in 2s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\Pracical 11 NODE>
PS D:\Pracical 11 NODE> node server.js
Server running on http://localhost:3000
Accessed: D:\Pracical 11 NODE\
Accessed: D:\Pracical 11 NODE\favicon.ico
Accessed: D:\Pracical 11 NODE\page1
Accessed: D:\Pracical 11 NODE\page2
Accessed: D:\Pracical 11 NODE\report
```

In Node.js, the **path module** is used to handle and manage file and folder paths efficiently. Sometimes, while tracking file access or navigation patterns in a website, multiple paths may point to the same file — this creates **redundancy**. To resolve this, we use specific functions from the path module depending on whether redundancy exists or not.

If redundancy exists, we use **path.resolve()**.
It converts a sequence of paths into an **absolute path** and removes unnecessary parts such as "../" or "./". This ensures that the final path is clean and free of duplication.

If there is no redundancy, we use **path.join()**.
It simply joins multiple path segments together using the correct operating system separator (like "/" or "\"). This is ideal for clean paths that don't contain redundant elements.

Code Snippet:

const path = require('path');

```
let redundancyFound = true; // assume we checked and found redundancy


if (redundancyFound) {
  console.log("Redundancy detected → Using path.resolve()");
  const fixedPath = path.resolve('folder', 'subfolder', '../file.txt');
  console.log("Resolved Path:", fixedPath);
} else {
  console.log("No redundancy → Using path.join()");
  const joinedPath = path.join('folder', 'subfolder', 'file.txt');
  console.log("Joined Path:", joinedPath);
}
```

Output Example:

If redundancy exists →
C:\Users\Akshat\folder\file.txt *(Clean absolute path using path.resolve())*

If no redundancy →
folder/subfolder/file.txt *(Simple joined path using path.join())*

Opinion / Conclusion:

If redundancy is found in the file paths, **path.resolve()** should be used because it cleans the path by removing unnecessary segments and provides a single absolute path. This helps prevent duplication and ensures accuracy in the website's path tracking.

If there is no redundancy, **path.join()** is used because it efficiently combines path segments without altering them. It is faster, simpler, and ideal for already organized file structures.

Therefore, both methods are used according to the situation:

- Use path.resolve() → when redundancy exists.
- Use path.join() → when there is no redundancy.

Github- https://github.com/Akshat-280205/Practical-11-NODE-JS.git