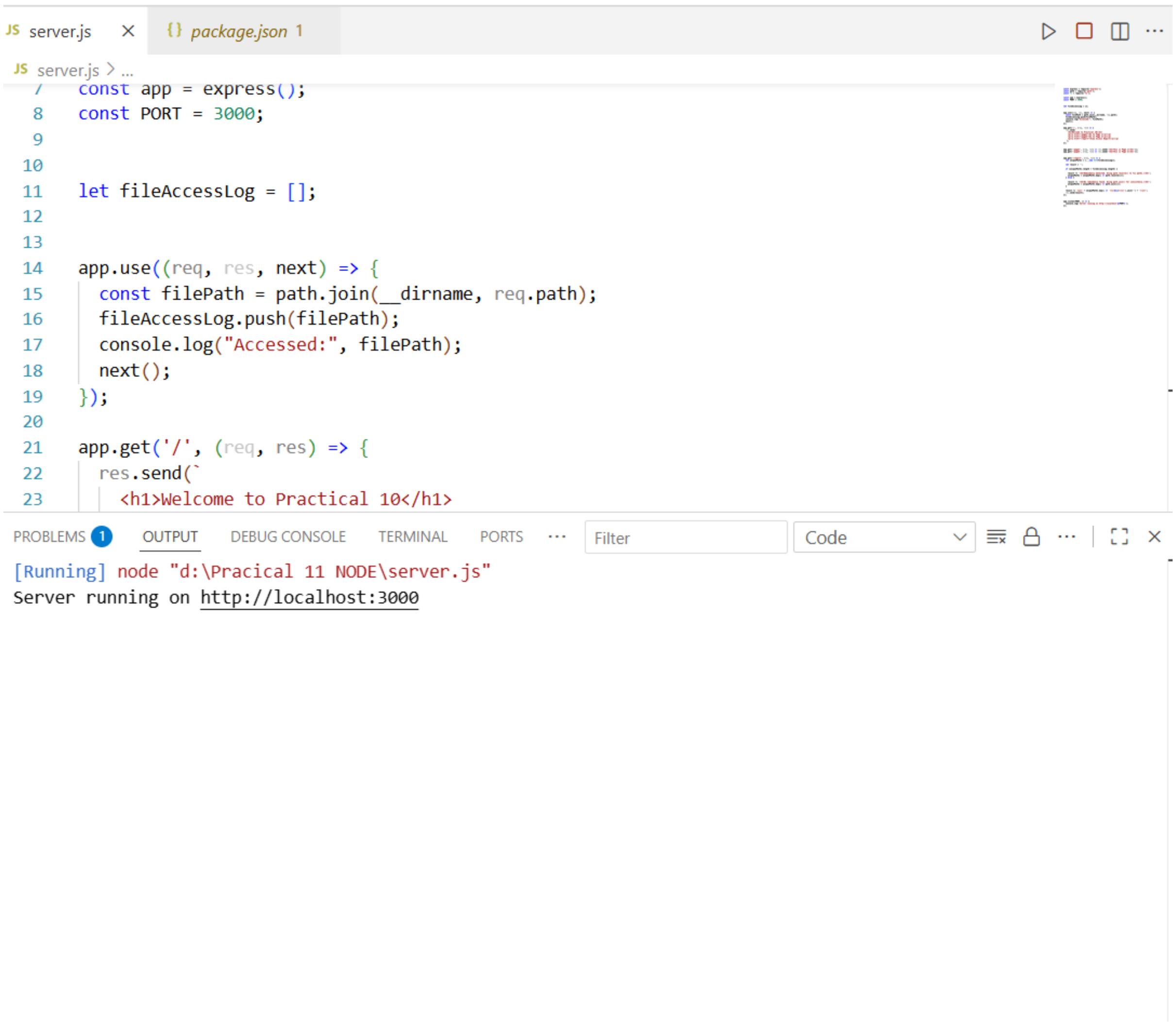


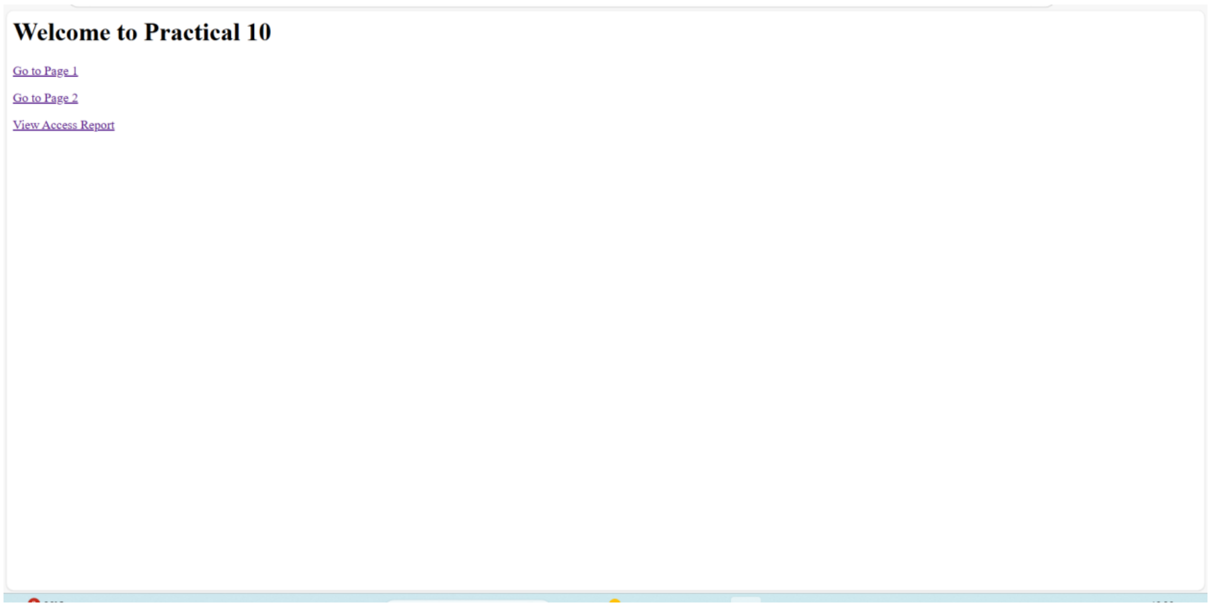
server.js × {} package.json 1

server.js > ...

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


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>
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS ··· Filter Code

```
[Running] node "d:\Pracical 11 NODE\server.js"
Server running on http://localhost:3000
```



Welcome to Practical 10

Go to Page 1

Go to Page 2

View Access Report

In Node.js, the *path module* is used to handle and manipulate file and directory paths. It helps ensure that file paths work correctly across all operating systems. When creating a website that tracks file access patterns, sometimes the same file may be accessed more than once using slightly different paths. This creates redundancy in file paths. To handle such redundancy, different path module methods are used depending on the situation.

If redundancy exists, we use the **path.resolve()** method. This method converts a sequence of paths into an absolute path and automatically removes any redundant segments like "../" or "./". It ensures that each file path is unique, clean, and standardized. For example, if the same file is accessed from different subfolders, path.resolve() will simplify the path and point to the correct file location without duplication. Hence, when redundancy is detected in the website's file access logs, path.resolve() is executed to fix and normalize the paths.

However, if there is no redundancy, we use the **path.join()** method. This method simply joins different parts of a file path together using the correct system separator (either "/" on Linux or "\" on Windows). It does not attempt to check or remove redundant segments, making it faster and simpler for clean and organized paths. Therefore, when all paths are already structured properly and there is no duplication, path.join() is the preferred choice because it provides neat and efficient path concatenation.

In conclusion, path.resolve() is executed when redundancy exists, as it removes repeated or unnecessary path segments and gives a clear absolute path. On the other hand, path.join() is executed when there is no redundancy, since it efficiently combines the path segments without modification. Both methods ensure proper and reliable path handling in Node.js applications depending on whether redundancy is present or not.

```
const path = require('path');


if (redundancyFound) {
  // Redundancy exists
  console.log("Using path.resolve() to clean redundant paths");
  const fixedPath = path.resolve('folder', 'subfolder', '../file.txt');
  console.log(fixedPath);
} else {
  // No redundancy
  console.log("Using path.join() for simple path creation");
  const cleanPath = path.join('folder', 'subfolder', 'file.txt');
  console.log(cleanPath);
}
```

Github- https://github.com/RituSingh0204/Practical-11-Node-Js.git