# Practical – 6

GitHub Link: https://github.com/Devsharma511/NodeJs.git

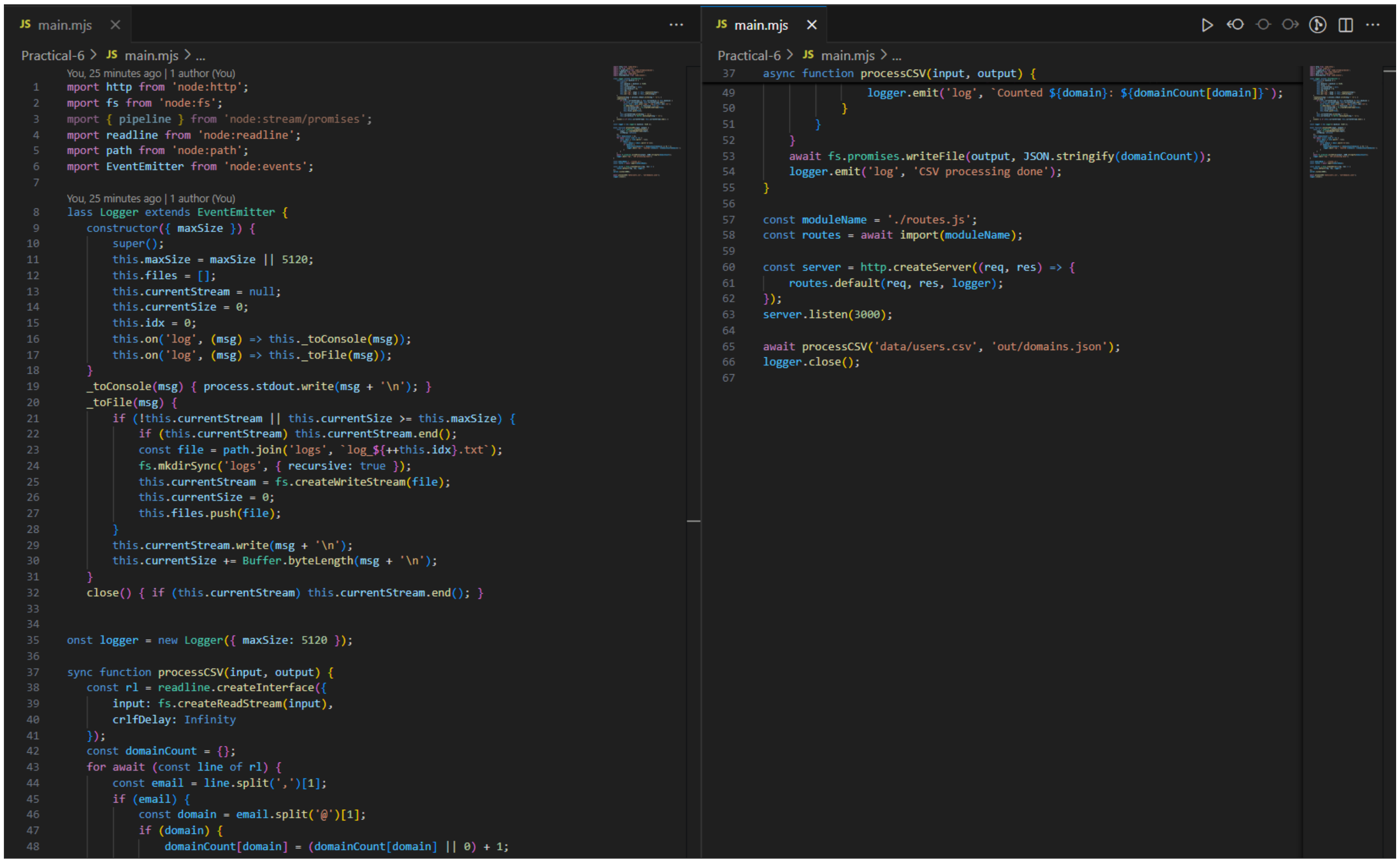
1. Write a program to create the server with dynamic imports with top level await.

2. Goal: Use stream, readline, pipeline, backpressure.
Tasks: Given data/users.csv (~1M rows), count users per domain from email. Write results to out/domains.json without loading all into memory.

3. Create an EventEmitter-based logger with transports and rotation: Logger emits log events; transports subscribe (console, file). Implement size-based rotation at ~5-0KB per file.

**Project Structure:**

1. main.mjs: Main program file implementing the server, CSV processing, and logger.



```
mport http from 'node:http';
mport fs from 'node:fs';
mport { pipeline } from 'node:stream/promises';
mport readline from 'node:readline';
mport path from 'node:path';
mport EventEmitter from 'node:events';

lass Logger extends EventEmitter {
   constructor({ maxSize }) {
      super();
      this.maxSize = maxSize || 5120;
      this.files = [];
      this.currentStream = null;
      this.currentSize = 0;
      this.idx = 0;
      this.on('log', (msg) => this._toConsole(msg));
      this.on('log', (msg) => this._toFile(msg));
   }
   _toConsole(msg) { process.stdout.write(msg + '\n'); }
   _toFile(msg) {
      if (!this.currentStream || this.currentSize >= this.maxSize) {
         if (this.currentStream) this.currentStream.end();
         const file = path.join('logs', `log_${++this.idx}.txt`);
         fs.mkdirSync('logs', { recursive: true });
         this.currentStream = fs.createWriteStream(file);
         this.currentSize = 0;
         this.files.push(file);
      }
      this.currentStream.write(msg + '\n');
      this.currentSize += Buffer.byteLength(msg + '\n');
   }
   close() { if (this.currentStream) this.currentStream.end(); }


onst logger = new Logger({ maxSize: 5120 });

sync function processCSV(input, output) {
   const rl = readline.createInterface({
      input: fs.createReadStream(input),
      crlfDelay: Infinity
   });
   const domainCount = {};
   for await (const line of rl) {
      const email = line.split(',')[1];
      if (email) {
         const domain = email.split('@')[1];
         if (domain) {
            domainCount[domain] = (domainCount[domain] || 0) + 1;
```

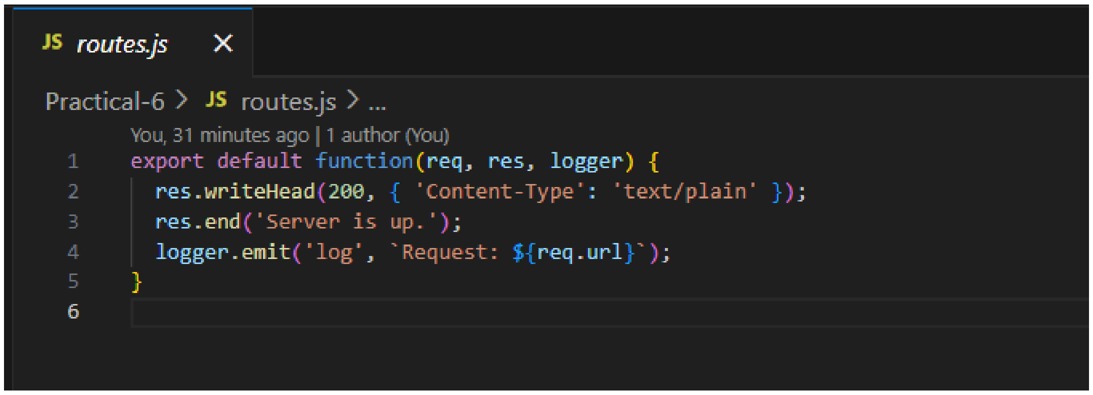
```
async function processCSV(input, output) {
               logger.emit('log', `Counted ${domain}: ${domainCount[domain]}`);
            }
         }
      }
      await fs.promises.writeFile(output, JSON.stringify(domainCount));
      logger.emit('log', 'CSV processing done');
}

const moduleName = './routes.js';
const routes = await import(moduleName);

const server = http.createServer((req, res) => {
      routes.default(req, res, logger);
});
server.listen(3000);

await processCSV('data/users.csv', 'out/domains.json');
logger.close();
```

2. routes.js: Module handling HTTP request responses.


routes.js
Practical-6 > routes.js > ...
You, 31 minutes ago | 1 author (You)

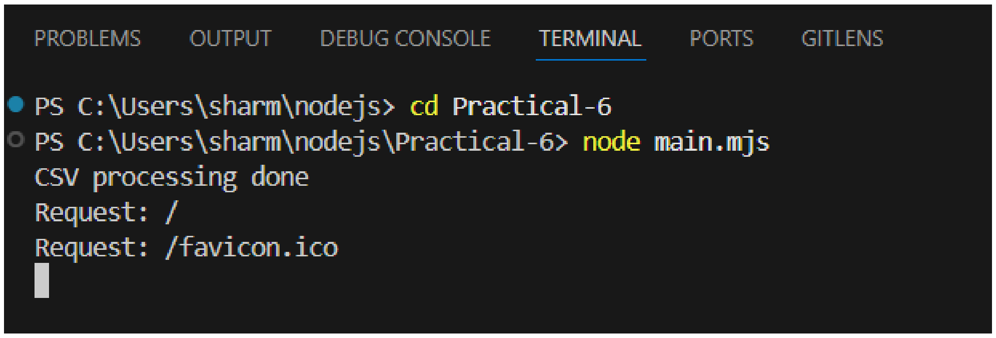
```javascript
export default function(req, res, logger) {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Server is up.');
  logger.emit('log', `Request: ${req.url}`);
}
```

**3.** data/users.csv: Input CSV with user data (id, name, email).

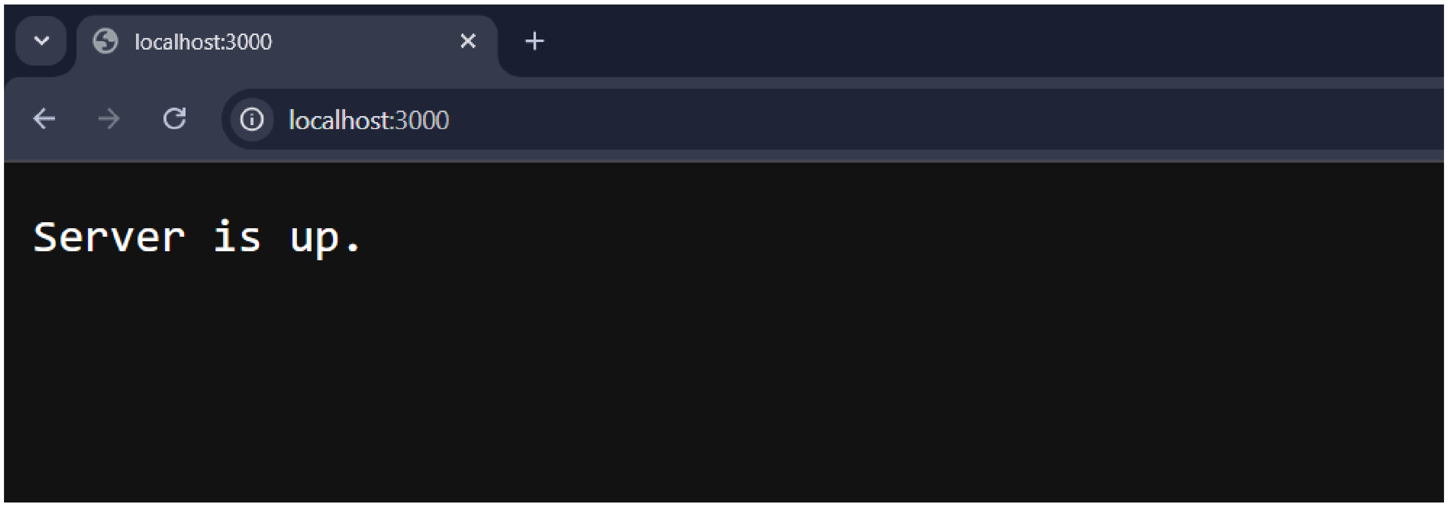
**4.** out/domains.json: Output JSON file to save domain counts (created after processing).

**5.** logs/: Directory where rotating log files are saved.

**Output:**



```
PS C:\Users\sharm\nodejs> cd Practical-6
PS C:\Users\sharm\nodejs\Practical-6> node main.mjs
CSV processing done
Request: /
Request: /favicon.ico
```



```
Server is up.
```