FILE: codeExporter.js

```
#!/usr/bin/env node

const fs = require("fs");
const path = require("path");
const { exec } = require("child_process");
const puppeteer = require("puppeteer");
require("dotenv").config();

const rootDir = path.resolve(process.env.ROOT_DIR || "./");
const projectDirName = path.basename(rootDir);
const excludeDirs = (
  process.env.EXCLUDE_DIRS || "node_modules,.git,dist"
).split(",");
const allowedExtensions = (
  process.env.ALLOWED_EXTENSIONS || ".js,.ts,.jsx,.tsx"
).split(",");
const maxFileSize = parseInt(process.env.MAX_FILE_SIZE_MB || "1") * 1024 * 1024;
const outputDir = path.join(process.cwd(), "pdf");

if (!fs.existsSync(outputDir)) {
  fs.mkdirSync(outputDir);
}

let outputFileIndex = 1;
let currentSize = 0;
let fileContents = [];

function shouldExclude(filePath) {
  return excludeDirs.some((exclude) =>
    filePath.includes(path.sep + exclude + path.sep)
  );
}

function isHiddenOrInvalid(filePath) {
  const baseName = path.basename(filePath);
  const ext = path.extname(filePath).toLowerCase();
  return baseName.startsWith(".") || !allowedExtensions.includes(ext);
}

function escapeHtml(str) {
  return str.replace(/&/g, "&amp;").replace(/</g, "&lt;").replace(/>/g, "&gt;");
}

function addToContent(relativePath, content) {
  const ext = path.extname(relativePath).toLowerCase();
  let lang = "javascript";
  if (ext === ".ts" || ext === ".tsx") lang = "typescript";

  const htmlSection = `
    <section style="margin-bottom: 40px; page-break-inside: avoid;">
      <h2 style="font-family: monospace; font-size: 14px; background: #eee; padding:
8px;">
        FILE: ${relativePath}
      </h2>
      <pre><code class="language-${lang}">${escapeHtml(content)}</code></pre>
    </section>
  `;
  fileContents.push(htmlSection);
```

```
}

function processFile(filePath) {
  if (isHiddenOrInvalid(filePath)) return;

  const relativePath = path.relative(rootDir, filePath);
  const content = fs.readFileSync(filePath, "utf-8");

  const buffer = Buffer.from(content, "utf-8");
  if (currentSize + buffer.length > maxFileSize) {
    saveAsPDF(outputFileIndex, fileContents.join(""));
    outputFileIndex++;
    fileContents = [];
    currentSize = 0;
  }

  currentSize += buffer.length;
  addToContent(relativePath, content);
}

function traverseDirectory(dir) {
  const entries = fs.readdirSync(dir, { withFileTypes: true });
  for (const entry of entries) {
    const fullPath = path.join(dir, entry.name);
    if (entry.name.startsWith(".")) continue;

    if (entry.isDirectory()) {
      if (!shouldExclude(fullPath)) {
        traverseDirectory(fullPath);
      }
    } else {
      processFile(fullPath);
    }
  }
}

async function saveAsPDF(index, htmlContent) {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();

  const html = `
    <html>
    <head>
      <meta charset="UTF-8">
      <title>${projectDirName}-${index}</title>
      <link href="https://fonts.googleapis.com/css2?family=Roboto+Mono&display=swap"
rel="stylesheet">
      <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.11.0/styles/arduino-
light.min.css">
      <style>
        body {
          font-family: 'Roboto Mono', monospace;
          padding: 0 20px;
          font-size: 12px;
          color: #333;
        }
        h2 {
          color: #222;
          font-size: 14px;
          margin-top: 30px;
          font-weight: bold;
        }
        section {
```

```
          page-break-inside: avoid;
        }
        pre {
          white-space: pre-wrap;
          word-break: break-word;
          overflow-wrap: break-word;
          // padding: 4px;
          // border-radius: 6px;
          // background:rgb(241, 239, 239);
        }
        code {
          font-family: 'Roboto Mono', monospace;
          font-size: 12px;
        }
        @page {
          margin: 0.5in;
        }
      </style>
    </head>
    <body>
      ${htmlContent}
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.11.0/highlight.min.js">
</script>
      <script>hljs.highlightAll();</script>
    </body>
    </html>
  `;

  await page.setContent(html, { waitUntil: "domcontentloaded" });

  const pdfPath = path.join(outputDir, `${projectDirName}-${index}.pdf`);

  await page.pdf({
    path: pdfPath,
    format: "A4",
    printBackground: true,
    margin: {
      top: "1in",
      bottom: "1in",
      left: "1in",
      right: "1in",
    },
    displayHeaderFooter: true,
    headerTemplate: `
      <div style="font-family: Roboto Mono, monospace; font-size: 10px; padding: 0
20px; width: 100%; text-align: center;">
        ${projectDirName}-${index}.pdf
      </div>
    `,
    footerTemplate: `
      <div style="font-family: Roboto Mono, monospace; font-size: 10px; padding: 0
20px; width: 100%; text-align: center;">
        <span class="pageNumber"></span> of <span class="totalPages"></span>
        <span style="float: right;">${new Date().toLocaleDateString()}  ${new
Date().toLocaleTimeString()}</span>
      </div>
    `,
  });

  await browser.close();
  console.log(`📄 PDF saved: ${pdfPath}`);
}
```

```
function openOutputFolder() {
  const platform = process.platform;

  if (platform === "win32") {
    exec(`start "" "${outputDir}"`);
  } else if (platform === "darwin") {
    exec(`open "${outputDir}"`);
  } else if (platform === "linux") {
    exec(`xdg-open "${outputDir}"`);
  } else {
    console.log("🚫 Cannot auto-open folder: unsupported platform.");
  }
}

function main() {
  traverseDirectory(rootDir);
  if (fileContents.length > 0) {
    saveAsPDF(outputFileIndex, fileContents.join("")).then(() => {
      console.log("✅ All done!");
      openOutputFolder();
    });
  } else {
    console.log("❗ No content to write.");
  }
}

main();
```

FILE: copyDirectoryCLI (to text).js

```
#!/usr/bin/env node

const fs = require("fs");
const path = require("path");
require("dotenv").config();

const rootDir = path.resolve(process.env.ROOT_DIR || "./");
const excludeDirs = (
  process.env.EXCLUDE_DIRS || "node_modules,.git,dist"
).split(",");
const allowedExtensions = (process.env.ALLOWED_EXTENSIONS || ".js,.ts").split(
  ","
);
const maxFileSize = parseInt(process.env.MAX_FILE_SIZE_MB || "1") * 1024 * 1024;
const outputPrefix = process.env.OUTPUT_PREFIX || "output";

let outputFileIndex = 1;
let currentSize = 0;
let writeStream = null;

function getOutputFilePath() {
  return path.join(process.cwd(), `${outputPrefix}-${outputFileIndex}.txt`);
}

function openNewWriteStream() {
  if (writeStream) {
    writeStream.end();
  }
  const filePath = getOutputFilePath();
  writeStream = fs.createWriteStream(filePath, { flags: "w" });
  currentSize = 0;
  console.log(`📄 Writing to ${filePath}`);
}

function shouldExclude(filePath) {
  return excludeDirs.some((exclude) =>
    filePath.includes(path.sep + exclude + path.sep)
  );
}

function isHiddenOrInvalid(filePath) {
  const baseName = path.basename(filePath);
  const ext = path.extname(filePath);
  return baseName.startsWith(".") || !allowedExtensions.includes(ext);
}

function writeWithSizeCheck(data) {
  const buffer = Buffer.from(data, "utf-8");

  if (currentSize + buffer.length > maxFileSize) {
    outputFileIndex++;
    openNewWriteStream();
  }

  writeStream.write(buffer);
  currentSize += buffer.length;
}

function processFile(filePath) {
  if (isHiddenOrInvalid(filePath)) return;
```

```
  const relativePath = path.relative(rootDir, filePath);
  const content = fs.readFileSync(filePath, "utf-8");

  const header = `--- FILE: ${relativePath} ---\n`;
  const fullContent = `${header}${content}\n\n`;

  writeWithSizeCheck(fullContent);
}

function traverseDirectory(dir) {
  const entries = fs.readdirSync(dir, { withFileTypes: true });

  for (const entry of entries) {
    const fullPath = path.join(dir, entry.name);

    if (entry.name.startsWith(".")) continue;

    if (entry.isDirectory()) {
      if (!shouldExclude(fullPath)) {
        traverseDirectory(fullPath);
      }
    } else {
      processFile(fullPath);
    }
  }
}

function main() {
  openNewWriteStream();
  traverseDirectory(rootDir);
  if (writeStream) writeStream.end(() => console.log("✅ All done!"));
}

main();
```