**index copy 2.js** – *js code*

const fs = require('fs');
  
const path = require('path');
  
const PizZip = require('pizzip');
  
const Docxtemplater = require('docxtemplater');
  
const hljs = require('highlight.js');
  
const readline = require('readline');
  
const { exec } = require('child\_process');
  
  
const rl = readline.createInterface({
  
 input: process.stdin,
  
 output: process.stdout
  
});
  
  
const defaultInputDir = path.relative(process.cwd(), path.resolve(\_\_dirname, 'input'));
  
  
const askForInputDir = () => {
  
 return new Promise((resolve) => {
  
 rl.question(`\x1b[36mEnter the path to the input directory (leave blank for ${defaultInputDir}): \x1b[0m`, (inputDir) => {
  
 const resolvedInputDir = inputDir.trim() || path.resolve(\_\_dirname, 'input');
  
 resolve(resolvedInputDir);
  
 });
  
 });
  
};
  
  
const askForIgnoredPaths = () => {
  
 return new Promise((resolve) => {
  
 rl.question('\x1b[36mEnter the paths to ignore (separated by commas, leave blank for none): \x1b[0m', (input) => {
  
 const ignoredPaths = input.trim().split(',').map((path) => path.trim()).filter((path) => path !== '');
  
 resolve(ignoredPaths);
  
 });
  
 });
  
};
  
  
const askToOpenFile = () => {
  
 return new Promise((resolve) => {
  
 rl.question('\x1b[36mDo you want to open the converted file? (y/n): \x1b[0m', (answer) => {
  
 resolve(answer.trim().toLowerCase() === 'y');
  
 });
  
 });
  
};
  
  
const openFileInDefaultSoftware = (filePath) => {
  
 try {
  
 exec(`start ${filePath}`, (error, stdout, stderr) => {
  
 if (error) {
  
 if (error.message.includes('The process cannot access the file because it is being used by another process')) {
  
 console.log(`\x1b[31mFile ${filePath} is already open or some other software is currently using it.\nPlease close the file and try again.\x1b[0m`);
  
 } else {
  
 console.error(`\x1b[31mExec error: ${error}\x1b[0m`);
  
 }
  
 return;
  
 }
  
 console.log(`🚀 Opened ${filePath} in default software.`);
  
 });
  
 } catch (error) {
  
 console.error(`\x1b[31mError: ${error.message}\x1b[0m`);
  
 }
  
 };
  
  
const startConversion = async () => {
  
 const inputDir = await askForInputDir();
  
 if (!fs.existsSync(inputDir)) {
  
 console.log(`\x1b[31mSorry can't find input directory. Please enter a valid folder path!\x1b[0m`);
  
 process.exit()
  
 }
  
  
 const ignoredPaths = await askForIgnoredPaths();
  
  
 // Load the docx file as binary content
  
 let content
  
 try{
  
 content = fs.readFileSync(path.resolve(\_\_dirname, 'template.docx'), 'binary');
  
 }catch{
  
 console.log(`\x1b[31mSorry can't find template.docx file. Please enter a valid template path!\x1b[0m`);
  
 process.exit()
  
 }
  
  
 // Unzip the content of the file
  
 const zip = new PizZip(content);
  
  
 // Create a new instance of Docxtemplater
  
 const doc = new Docxtemplater(zip, { paragraphLoop: true, linebreaks: true });
  
  
 // Function to recursively read code files from a directory
  
 function readCodeFiles(dir) {
  
 const snippets = [];
  
 const files = fs.readdirSync(dir);
  
  
 for (const file of files) {
  
 const filePath = path.join(dir, file);
  
 const relativeFilePath = path.relative(inputDir, filePath).replace(/\\/g, '/');
  
  
 if (ignoredPaths.includes(relativeFilePath)) {
  
 continue; // Skip ignored files/folders
  
 }
  
  
 const stats = fs.statSync(filePath);
  
  
 if (stats.isDirectory()) {
  
 snippets.push(...readCodeFiles(filePath));
  
 } else if (/\.(js|py|java|cpp|cs|rb)$/.test(filePath)) {
  
 const code = fs.readFileSync(filePath, 'utf8');
  
 const language = path.extname(filePath).slice(1);
  
 const highlighted = hljs.highlight(code, { language }).value;
  
  
 // Remove all styles except font-family
  
 const formattedHighlighted = highlighted.replace(/<\/?span[^>]\*>/g, '').replace(/style=".\*?"/g, '');
  
  
 snippets.push({
  
 language,
  
 code,
  
 highlighted: `<span style="font-family:Consolas,Courier New,monospace;">${formattedHighlighted}</span>`,
  
 filePath: relativeFilePath
  
 });
  
 }
  
 }
  
  
 return snippets;
  
 }
  
  
 const snippets = readCodeFiles(inputDir);
  
  
 // Set the data for the document
  
 const data = {
  
 title: 'Code Snippets',
  
 snippets
  
 };
  
  
 // Render the document with the data
  
 doc.render(data);
  
  
 // Get the zip document and generate it as a nodebuffer
  
 const buf = doc.getZip().generate({
  
 type: 'nodebuffer',
  
 compression: 'DEFLATE'
  
 });
  
  
 // Write the buffer to a file
  
 const outputFilePath = path.resolve(\_\_dirname, 'output.docx');
  
 fs.writeFileSync(outputFilePath, buf);
  
  
 console.log('\n---------\n✨ Conversion completed! The output.docx file has been generated.\n---------\n');
  
  
 const openFile = await askToOpenFile();
  
 if (openFile) {
  
 openFileInDefaultSoftware(outputFilePath);
  
 }
  
  
 rl.close();
  
};
  
  
startConversion();

**index copy.js** – *js code*

const fs = require('fs');
  
const path = require('path');
  
const PizZip = require('pizzip');
  
const Docxtemplater = require('docxtemplater');
  
const hljs = require('highlight.js');
  
const readline = require('readline');
  
const { exec } = require('child\_process');
  
  
const rl = readline.createInterface({
  
 input: process.stdin,
  
 output: process.stdout
  
});
  
  
const defaultInputDir = path.relative(process.cwd(), path.resolve(\_\_dirname, 'input'));
  
  
const askForInputDir = () => {
  
 return new Promise((resolve) => {
  
 rl.question(`\x1b[36mEnter the path to the input directory (leave blank for ${defaultInputDir}): \x1b[0m`, (inputDir) => {
  
 const resolvedInputDir = inputDir.trim() || path.resolve(\_\_dirname, 'input');
  
 resolve(resolvedInputDir);
  
 });
  
 });
  
};
  
  
const askForIgnoredPaths = () => {
  
 return new Promise((resolve) => {
  
 rl.question('\x1b[36mEnter the paths to ignore (separated by commas, leave blank for none): \x1b[0m', (input) => {
  
 const ignoredPaths = input.trim().split(',').map((path) => path.trim()).filter((path) => path !== '');
  
 resolve(ignoredPaths);
  
 });
  
 });
  
};
  
  
const askToOpenFile = () => {
  
 return new Promise((resolve) => {
  
 rl.question('\x1b[36mDo you want to open the converted file? (y/n): \x1b[0m', (answer) => {
  
 resolve(answer.trim().toLowerCase() === 'y');
  
 });
  
 });
  
};
  
  
const openFileInDefaultSoftware = (filePath) => {
  
 try {
  
 exec(`start ${filePath}`, (error, stdout, stderr) => {
  
 if (error) {
  
 if (error.message.includes('The process cannot access the file because it is being used by another process')) {
  
 console.log(`\x1b[31mFile ${filePath} is already open or some other software is currently using it.\nPlease close the file and try again.\x1b[0m`);
  
 } else {
  
 console.error(`\x1b[31mExec error: ${error}\x1b[0m`);
  
 }
  
 return;
  
 }
  
 console.log(`🚀 Opened ${filePath} in default software.`);
  
 });
  
 } catch (error) {
  
 console.error(`\x1b[31mError: ${error.message}\x1b[0m`);
  
 }
  
 };
  
  
const startConversion = async () => {
  
 const inputDir = await askForInputDir();
  
 if (!fs.existsSync(inputDir)) {
  
 console.log(`\x1b[31mSorry can't find input directory. Please enter a valid folder path!\x1b[0m`);
  
 process.exit()
  
 }
  
  
 const ignoredPaths = await askForIgnoredPaths();
  
  
 // Load the docx file as binary content
  
 let content
  
 try{
  
 content = fs.readFileSync(path.resolve(\_\_dirname, 'template.docx'), 'binary');
  
 }catch{
  
 console.log(`\x1b[31mSorry can't find template.docx file. Please enter a valid template path!\x1b[0m`);
  
 process.exit()
  
 }
  
  
 // Unzip the content of the file
  
 const zip = new PizZip(content);
  
  
 // Create a new instance of Docxtemplater
  
 const doc = new Docxtemplater(zip, { paragraphLoop: true, linebreaks: true });
  
  
 // Function to recursively read code files from a directory
  
 function readCodeFiles(dir) {
  
 const snippets = [];
  
 const files = fs.readdirSync(dir);
  
  
 for (const file of files) {
  
 const filePath = path.join(dir, file);
  
 const relativeFilePath = path.relative(inputDir, filePath).replace(/\\/g, '/');
  
  
 if (ignoredPaths.includes(relativeFilePath)) {
  
 continue; // Skip ignored files/folders
  
 }
  
  
 const stats = fs.statSync(filePath);
  
  
 if (stats.isDirectory()) {
  
 snippets.push(...readCodeFiles(filePath));
  
 } else if (/\.(js|py|java|cpp|cs|rb)$/.test(filePath)) {
  
 const code = fs.readFileSync(filePath, 'utf8');
  
 const language = path.extname(filePath).slice(1);
  
 const highlighted = hljs.highlight(code, { language }).value;
  
  
 // Remove all styles except font-family
  
 const formattedHighlighted = highlighted.replace(/<\/?span[^>]\*>/g, '').replace(/style=".\*?"/g, '');
  
  
 snippets.push({
  
 language,
  
 code,
  
 highlighted: `<span style="font-family:Consolas,Courier New,monospace;">${formattedHighlighted}</span>`,
  
 filePath: relativeFilePath
  
 });
  
 }
  
 }
  
  
 return snippets;
  
 }
  
  
 const snippets = readCodeFiles(inputDir);
  
  
 // Set the data for the document
  
 const data = {
  
 title: 'Code Snippets',
  
 snippets
  
 };
  
  
 // Render the document with the data
  
 doc.render(data);
  
  
 // Get the zip document and generate it as a nodebuffer
  
 const buf = doc.getZip().generate({
  
 type: 'nodebuffer',
  
 compression: 'DEFLATE'
  
 });
  
  
 // Write the buffer to a file
  
 const outputFilePath = path.resolve(\_\_dirname, 'output.docx');
  
 fs.writeFileSync(outputFilePath, buf);
  
  
 console.log('\n---------\n✨ Conversion completed! The output.docx file has been generated.\n---------\n');
  
  
 const openFile = await askToOpenFile();
  
 if (openFile) {
  
 openFileInDefaultSoftware(outputFilePath);
  
 }
  
  
 rl.close();
  
};
  
  
startConversion();