package.json

```
{
  "name": "stupid-usco",
  "version": "0.0.1",
  "description": "",
  "main": "dist/main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "npm run cleanDist && tsc && npm run copyAssets",
    "cleanDist": "ts-node scripts/cleanDir --project tsconfig.build.json",
    "copyAssets": "ts-node scripts/copyAssets --project tsconfig.build.json"
  },
  "author": "Joshua Sosso (joshmossas)",
  "license": "MIT",
  "dependencies": {
    "fast-glob": "^3.2.11",
    "fs-extra": "^10.1.0",
    "istextorbinary": "^6.0.0",
    "pdf-merger-js": "^3.4.0",
    "wkhtmltopdf": "^0.4.0"
  },
  "devDependencies": {
    "@types/fs-extra": "^9.0.13",
    "@types/wkhtmltopdf": "^0.3.4",
    "typescript": "^4.7.4"
  }
}
```

src/logger.ts

```
class Logger {
  logEnabled: boolean;

  constructor(enabled: boolean) {
    this.logEnabled = enabled;
  }

  log(msg: any) {
    if (this.logEnabled) {
      console.log(msg);
    }
  }

  error(msg: any) {
    if (this.logEnabled) {
      console.error(msg);
    }
  }
}

export default Logger;
```

src/main.ts

```
import { ensureDir, remove } from "fs-extra";
import glob from "fast-glob";
import { isText } from "istextorbinary";
import FileConverter from "./fileConverter";
import Logger from "./logger";
import path from "path";

const TEMP_DIR = ".stupid-usco";

const removeLeadingAndTrailingSlashes = (string: string): string =>
  string.trim().replace(/^\/+|\/+$/g, "");

const getFilePaths = async (directory: string, ignore: string[]) => {
  const formattedDir = removeLeadingAndTrailingSlashes(directory);
  console.log(formattedDir);
  const files = await glob(`${formattedDir}/**/*`, {
    ignore,
  });
  return files.filter((file) => isText(file));
};

export interface Config {
  dir: string;
  ignore: string[];
  output: string;
  disableLogs?: boolean;
}

export const defineConfig = (config: Config) => config;

export const generatePdf = async (config: Config) => {
  const logger = new Logger(!config.disableLogs);
  await ensureDir(TEMP_DIR);
  const files = await getFilePaths(config.dir, config.ignore);
  const manager = new FileConverter();
  logger.log(`converting ${files.length} files to PDF`);
  for (const file of files) {
    await manager.convertToPdf(file);
    logger.log(`converted ${file}`);
  }
  logger.log(`merging PDFs`);
  await manager.mergePdfs(config.output);
  await remove(TEMP_DIR);
  logger.log("finished!");
};
```

src/template.html

```
{{heading}}

  {{body}}
```

test_files/hello.js

```
console.log("hello world");

const sayHello = () => {
  console.log("hello world");
};

const saySomething = (input) => {
  console.log(input);
};

sayHello();
saySomething("Hello Again!");
```

test_files/hello.py

```
print("hello world")

def sayHello():
    print("Hello World!")

def saySomething(str):
    print(str)

sayHello()

saySomething("Hello again")
```

test_files/somemodule/hello.rs

```
fn main() {
    println!("Hello world")
}
```

test_files/somemodule/hello_again.rs

```
fn main() {
    println!("Hello Again")
}
```

README.md

````
# Stupid USCO

The united states copyright office is stupid. They want to recieve source code as PDF files. This is a library will convert all text files in a directory to a single PDF.

## Installation

```
npm install stupid-usco
```

## Usage

```ts
import { defineConfig, generatePdf } from "stupid-usco";

const config = defineConfig({
  dir: "",
  output: "",
  ignore: ["node_modules", "dist", "compiled-libs"],
});
```
````

test.js

```
const { generatePdf } = require("./dist/main");

generatePdf({
  dir: ".",
  output: "example.pdf",
  ignore: ["node_modules", "dist", "package-lock.json"],
}).catch((err) => console.log(err));
```

tsconfig.build.json

```
{
  "compilerOptions": {
    "composite": true,
    "rootDir": "./",
    "esModuleInterop": true
  }
}
```

tsconfig.json

```
{
  "compilerOptions": {
    /* Visit https://aka.ms/tsconfig.json to read more about this file */

    /* Projects */
    // "incremental": true,                              /* Enable incremental compilation */
    // "composite": true,                                /* Enable constraints that allow a TypeScript project to be used with project references. */
    // "tsBuildInfoFile": "./",                          /* Specify the folder for .tsbuildinfo incremental compilation files. */
    // "disableSourceOfProjectReferenceRedirect": true,  /* Disable preferring source files instead of declaration files when referencing composite projects */
    // "disableSolutionSearching": true,                 /* Opt a project out of multi-project reference checking when editing. */
    // "disableReferencedProjectLoad": true,             /* Reduce the number of projects loaded automatically by TypeScript. */

    /* Language and Environment */
    "target": "es2016" /* Set the JavaScript language version for emitted JavaScript and include compatible library declarations. */,
    // "lib": [],                                        /* Specify a set of bundled library declaration files that describe the target runtime environment. */
    // "jsx": "preserve",                                /* Specify what JSX code is generated. */
    // "experimentalDecorators": true,                   /* Enable experimental support for TC39 stage 2 draft decorators. */
    // "emitDecoratorMetadata": true,                    /* Emit design-type metadata for decorated declarations in source files. */
    // "jsxFactory": "",                                 /* Specify the JSX factory function used when targeting React JSX emit, e.g. 'React.createElement' or 'h' */
    // "jsxFragmentFactory": "",                         /* Specify the JSX Fragment reference used for fragments when targeting React JSX emit e.g. 'React.Fragment' or 'Fragme
    // "jsxImportSource": "",                            /* Specify module specifier used to import the JSX factory functions when using `jsx: react-jsx*`.` */
    // "reactNamespace": "",                             /* Specify the object invoked for `createElement`. This only applies when targeting `react` JSX emit. */
    // "noLib": true,                                    /* Disable including any library files, including the default lib.d.ts. */
    // "useDefineForClassFields": true,                  /* Emit ECMAScript-standard-compliant class fields. */

    /* Modules */
    "module": "commonjs" /* Specify what module code is generated. */,
    "rootDir": "./src" /* Specify the root folder within your source files. */,
    // "moduleResolution": "node",                       /* Specify how TypeScript looks up a file from a given module specifier. */
    "baseUrl": "./" /* Specify the base directory to resolve non-relative module names. */,
    // "paths": {},                                      /* Specify a set of entries that re-map imports to additional lookup locations. */
    // "rootDirs": [],                                   /* Allow multiple folders to be treated as one when resolving modules. */
    // "typeRoots": [],                                  /* Specify multiple folders that act like `./node_modules/@types`. */
    // "types": [],                                      /* Specify type package names to be included without being referenced in a source file. */
    // "allowUmdGlobalAccess": true,                     /* Allow accessing UMD globals from modules. */
    // "resolveJsonModule": true,                        /* Enable importing .json files */
    // "noResolve": true,                                /* Disallow `import`s, `require`s or ``s from expanding the number of files TypeScript should add to a project. */

    /* JavaScript Support */
    // "allowJs": true,                                  /* Allow JavaScript files to be a part of your program. Use the `checkJS` option to get errors from these files. */
    // "checkJs": true,                                  /* Enable error reporting in type-checked JavaScript files. */
    // "maxNodeModuleJsDepth": 1,                        /* Specify the maximum folder depth used for checking JavaScript files from `node_modules`. Only applicable with `allow

    /* Emit */
    "declaration": true /* Generate .d.ts files from TypeScript and JavaScript files in your project. */,
    // "declarationMap": true,                           /* Create sourcemaps for d.ts files. */
    // "emitDeclarationOnly": true,                      /* Only output d.ts files and not JavaScript files. */
    // "sourceMap": true,                                /* Create source map files for emitted JavaScript files. */
    // "outFile": "./",                                  /* Specify a file that bundles all outputs into one JavaScript file. If `declaration` is true, also designates a file t
    "outDir": "./dist" /* Specify an output folder for all emitted files. */,
    // "removeComments": true,                           /* Disable emitting comments. */
    // "noEmit": true,                                   /* Disable emitting files from a compilation. */
    // "importHelpers": true,                            /* Allow importing helper functions from tslib once per project, instead of including them per-file. */
    // "importsNotUsedAsValues": "remove",               /* Specify emit/checking behavior for imports that are only used for types */
    // "downlevelIteration": true,                       /* Emit more compliant, but verbose and less performant JavaScript for iteration. */
    // "sourceRoot": "",                                 /* Specify the root path for debuggers to find the reference source code. */
    // "mapRoot": "",                                    /* Specify the location where debugger should locate map files instead of generated locations. */
    // "inlineSourceMap": true,                          /* Include sourcemap files inside the emitted JavaScript. */
    // "inlineSources": true,                            /* Include source code in the sourcemaps inside the emitted JavaScript. */
    // "emitBOM": true,                                  /* Emit a UTF-8 Byte Order Mark (BOM) in the beginning of output files. */
    // "newLine": "crlf",                                /* Set the newline character for emitting files. */
    // "stripInternal": true,                            /* Disable emitting declarations that have `@internal` in their JSDoc comments. */
    // "noEmitHelpers": true,                            /* Disable generating custom helper functions like `__extends` in compiled output. */
    // "noEmitOnError": true,                            /* Disable emitting files if any type checking errors are reported. */
    // "preserveConstEnums": true,                       /* Disable erasing `const enum` declarations in generated code. */
    // "declarationDir": "./",                           /* Specify the output directory for generated declaration files. */
    // "preserveValueImports": true,                     /* Preserve unused imported values in the JavaScript output that would otherwise be removed. */

    /* Interop Constraints */
    // "isolatedModules": true,                          /* Ensure that each file can be safely transpiled without relying on other imports. */
    // "allowSyntheticDefaultImports": true,             /* Allow 'import x from y' when a module doesn't have a default export. */
    "esModuleInterop": true /* Emit additional JavaScript to ease support for importing CommonJS modules. This enables `allowSyntheticDefaultImports` for type compatibility. */
    // "preserveSymlinks": true,                         /* Disable resolving symlinks to their realpath. This correlates to the same flag in node. */
    "forceConsistentCasingInFileNames": true /* Ensure that casing is correct in imports. */,

    /* Type Checking */
    "strict": true /* Enable all strict type-checking options. */,
    // "noImplicitAny": true,                            /* Enable error reporting for expressions and declarations with an implied `any` type.. */
    // "strictNullChecks": true,                         /* When type checking, take into account `null` and `undefined`. */
    // "strictFunctionTypes": true,                      /* When assigning functions, check to ensure parameters and the return values are subtype-compatible. */
    // "strictBindCallApply": true,                      /* Check that the arguments for `bind`, `call`, and `apply` methods match the original function. */
    // "strictPropertyInitialization": true,             /* Check for class properties that are declared but not set in the constructor. */
    // "noImplicitThis": true,                           /* Enable error reporting when `this` is given the type `any`. */
    // "useUnknownInCatchVariables": true,               /* Type catch clause variables as 'unknown' instead of 'any'. */
    // "alwaysStrict": true,                             /* Ensure 'use strict' is always emitted. */
    // "noUnusedLocals": true,                           /* Enable error reporting when a local variables aren't read. */
    // "noUnusedParameters": true,                       /* Raise an error when a function parameter isn't read */
    // "exactOptionalPropertyTypes": true,               /* Interpret optional property types as written, rather than adding 'undefined'. */
    // "noImplicitReturns": true,                        /* Enable error reporting for codepaths that do not explicitly return in a function. */
    // "noFallthroughCasesInSwitch": true,               /* Enable error reporting for fallthrough cases in switch statements. */
    // "noUncheckedIndexedAccess": true,                 /* Include 'undefined' in index signature results */
    // "noImplicitOverride": true,                       /* Ensure overriding members in derived classes are marked with an override modifier. */
    // "noPropertyAccessFromIndexSignature": true,       /* Enforces using indexed accessors for keys declared using an indexed type */
    // "allowUnusedLabels": true,                        /* Disable error reporting for unused labels. */
    // "allowUnreachableCode": true,                     /* Disable error reporting for unreachable code. */

    /* Completeness */
    // "skipDefaultLibCheck": true,                      /* Skip type checking .d.ts files that are included with TypeScript. */
    "skipLibCheck": true /* Skip type checking all .d.ts files. */
  },
  "include": ["src/**/*"],
  "references": [
    {
      "path": "tsconfig.build.json"
    }
  ]
}
```

scripts/cleanDir.ts

```
import { remove } from "fs-extra";

remove("dist");
```

scripts/copyAssets.ts

```
import { copyFile } from "fs-extra";

copyFile("src/template.html", "dist/template.html");
```

src/fileConverter.ts

```
import { createWriteStream, ensureFile, readFile, writeFile } from "fs-extra";
import path from "path";
import htmlToPdf from "wkhtmltopdf";
import glob from "fast-glob";
import PDFMerger from "pdf-merger-js";

class FileConverter {
  fileCount = 0;
  htmlTemplate = "";
  footerHtmlTemplate = `



      This pdf was generated using

        https://github.com/ModiiMedia/stupid-usco
      



`;
  tempDir = ".stupid-usco";

  async getHtml(heading: string, body: string) {
    if (!this.htmlTemplate) {
      this.htmlTemplate = (
        await readFile(path.resolve(__dirname, "template.html"))
      ).toString();
    }
    return this.htmlTemplate
      .replace("{{heading}}", heading)
      .replace("{{body}}", body);
  }

  getPdfFilename() {
    this.fileCount++;
    return `${this.tempDir}/${this.fileCount}.pdf`;
  }

  async convertToPdf(file: string): Promise {
    try {
      const text = (await readFile(file)).toString();
      const html = await this.getHtml(file, text);
      return new Promise((resolve, reject) => {
        const writeStream = createWriteStream(this.getPdfFilename());
        htmlToPdf(html, {
          footerLeft:
            "This PDF was generated using https://github.com/modiimedia/stupid-usco",
        }).pipe(writeStream);
        writeStream.on("finish", () => resolve());
        writeStream.on("error", (err) => reject(err));
      });
    } catch (_) {
      console.error(_);
      console.error(`file doesn't exists:`, file);
    }
  }

  async mergePdfs(filename: string) {
    const pdfFiles = await glob([`${this.tempDir}/**/*.pdf`]);
    const merger = new PDFMerger();
    for (const file of pdfFiles) {
      merger.add(file);
    }
    await ensureFile(filename);
    await merger.save(filename);
  }
}

export default FileConverter;
```

src/footer.html

```
      This pdf was generated using

_______ https://github.com/ModiiMedia/stupid-usco
______
```