default.css

```css
/*!
  Theme: Default
  Description: Original highlight.js style
  Author: (c) Ivan Sagalaev <maniac@softwaremaniacs.org>
  Maintainer: @highlightjs/core-team
  Website: https://highlightjs.org/
  License: see project LICENSE
  Touched: 2021
*/
    pre code.hljs {
  display: block;
  overflow-x: auto;
  padding: 1em;
}

    code.hljs {
  padding: 3px 5px;
}

    .hljs {
  background: #f3f3f3;
  color: #444;
}

    .hljs-comment {
  color: #697070;
}

    .hljs-punctuation,
    .hljs-tag {
  color: #444a;
}

    .hljs-tag .hljs-attr,
    .hljs-tag .hljs-name {
  color: #444;
}

    .hljs-attribute,
    .hljs-doctag,
    .hljs-keyword,
    .hljs-meta .hljs-keyword,
    .hljs-name,
    .hljs-selector-tag {
  font-weight: 700;
}

    .hljs-deletion,
    .hljs-number,
    .hljs-quote,
    .hljs-selector-class,
    .hljs-selector-id,
    .hljs-string,
    .hljs-template-tag,
    .hljs-type {
  color: #800;
}

```

```css
    .hljs-section,
    .hljs-title {
  color: #800;
  font-weight: 700;
}

    .hljs-link,
    .hljs-operator,
    .hljs-regexp,
    .hljs-selector-attr,
    .hljs-selector-pseudo,
    .hljs-symbol,
    .hljs-template-variable,
    .hljs-variable {
  color: #ab5656;
}

    .hljs-literal {
  color: #695;
}

    .hljs-addition,
    .hljs-built_in,
    .hljs-bullet,
    .hljs-code {
  color: #397300;
}

    .hljs-meta {
  color: #1f7199;
}

    .hljs-meta .hljs-string {
  color: #38a;
}

    .hljs-emphasis {
  font-style: italic;
}

    .hljs-strong {
  font-weight: 700;
}

```

LICENSE.md

package.json

```json
{
    "name": "repo2pdf",
    "version": "1.2.1",
    "description": "A Node.js utility for generating a PDF document from a GitHub repository",
    "main": "dist/clone.js",
    "bin": {
      "repo2pdf": "dist/clone.js"
    },
    "scripts": {
      "start": "node dist/clone.js",
      "watch": "tsc -w",
      "test": "echo \"Error: no test specified\" && exit 1"
    },
    "repository": {
      "type": "git",
      "url": "https://github.com/malpou/Repo-to-PDF.git"
    },
    "keywords": [
      "github",
      "repository",
      "pdf",
      "clone",
      "nodejs",
      "convert",
      "document",
      "langchain",
      "openai",
      "chatgpt",
      "utility",
      "tool"
    ],
    "author": "BankkRoll",
    "license": "MIT",
    "bugs": {
      "url": "https://github.com/BankkRoll/Repo-to-PDF/issues"
    },
    "homepage": "https://github.com/BankkRoll/Repo-to-PDF#readme",
    "documentation": "https://github.com/BankkRoll/Repo-to-PDF#readme",
    "dependencies": {
      "@types/inquirer": "^9.0.3",
      "@types/pdfkit": "^0.12.10",
      "chalk": "^5.2.0",
      "highlight.js": "^11.8.0",
      "inquirer": "^9.2.6",
      "isbinaryfile": "^5.0.0",
      "ora": "^6.3.1",
      "pdfkit": "^0.13.0",
      "puppeteer": "^20.7.3",
      "simple-git": "^3.18.0",
      "typescript": "^5.1.3"
    },
    "engines": {
      "node": ">=14.0.0"
    }
}
```

README.md

~~~markdown
# Repo-to-PDF

   Repo-to-PDF is a tool that allows you to convert a GitHub repository into a PDF file. It clones the repository, processes the files, and then creates a PDF.

   ## Example PDF

   [FreeCodeCamp](https://github.com/freeCodeCamp/freeCodeCamp) repository was converted into a PDF from 42,998 files to 186,453 pages in under 2 minutes. This conversion is purely for example and stress testing purposes. All content belongs to the original authors at FreeCodeCamp. You can view the PDF [here](https://freecodecamppdf.bankkroll.repl.co).
![Screenshot 2023-05-24 212226](https://github.com/BankkRoll/Repo-to-PDF/assets/106103625/9ceb176f-37f6-40d9-ab95-080942d2d7c0)


   ## Installation

   To use Repo-to-PDF, you have two options: cloning the repository from GitHub or installing it directly using NPX. Choose the method that suits you best.

   ### Cloning the Repository

   1. Clone the repository:
   ```shell
git clone https://github.com/BankkRoll/Repo-to-PDF
```

   2. Navigate to the Repo-to-PDF directory:
   ```shell
cd Repo-to-PDF
```

   3. Install the dependencies:
   ```shell
npm install
```

   4. Run the script:
   ```shell
npm start
```

   ### Installing with NPX
   This will download and install the latest version of Repo-to-PDF from the NPM registry.

   1. Install Repo-to-PDF using NPX:
   ```shell
npx repo2pdf
```

   2. Run Repo-to-PDF:
   ```shell
repo2pdf
```
~~~

~~~markdown

## Usage

Once you have installed Repo-to-PDF, you can use it to generate PDF files from GitHub repositories.

1. The script will install and start running. You will just follow the prompt:

You will be prompted to provide the following information:
- The URL of the GitHub repository
- The name of the output PDF file
- Whether or not you wish to keep the cloned repository after generating the PDF

The script will then clone the repository, process the files, and generate a PDF document based on the provided information.

Please note that you need to have Node.js installed on your system in order to run Repo-to-PDF.


## Configuration

Repo-to-PDF automatically ignores certain file types and directories (e.g., `.png`, `.git`). To customize the files and directories to ignore, edit the `excludedNames` and `excludedExtensions` variables in `clone.cjs`.


## Troubleshooting / FAQ

**Q: I'm getting an error "Failed to install [package-name]". What should I do?**
A: Make sure you have Node.js and npm installed on your system. Try running the following command to install the required package manually:
```shell
npm install [package-name]
~~~

Q: How can I customize the styling of the generated PDF?

A: You can modify the code in `clone.cjs` to change the font, font size, colors, and other styling options for the PDF document.
- Edit the `excludedExtensions` variable in `clone.cjs` to exclude certain file types from the PDF conversion.

Contributing

We welcome contributions! Here's how you can help:

- **Report bugs:** If you find a bug, please create an issue on GitHub describing the problem.
- **Suggest enhancements:** If you think of a way to improve Repo-to-PDF, we'd love to hear about it! Create an issue on GitHub to share your ideas.
- **Write code:** If you'd like to contribute code to fix a bug or implement a new feature, please fork the repository, make your changes, and submit a pull request.

License
```
```

```

   Repo-to-PDF is open source software, licensed under the MIT License. See the `LICENSE` file for more information.

```

src\clone.ts

```
#!/usr/bin/env node
  import fs from "fs"
  const fsPromises = fs.promises
  import path from "path"
  import { execSync } from "child_process"

  import git from "simple-git"
  import PDFDocument from "pdfkit"
  import { default as hljs } from "highlight.js"
  import { htmlToJson } from "./syntax"
  import { isBinaryFileSync } from "isbinaryfile"

  //@ts-ignore
  import type chalkType from "chalk";
  //@ts-ignore
  import type inquirerType from "inquirer";
  //@ts-ignore
  import type oraType from "ora";

  // TODO IDEAS
  // TODO add option to condiotionaly remove comments from code

  // TODO add option to condiotionaly remove empty lines from code

  // TODO add option to condiotionaly add line numbers to code

  // TODO add option to condiotionaly add linting to code

  // TODO add option to make one pdf per file


  let chalk: typeof chalkType;
  let inquirer: typeof inquirerType;
  let ora: typeof oraType;

  const spinnerPromise = import("ora").then((oraModule) =>     {
 ora = oraModule.default
    return ora("Setting everything up...").start()
})

  Promise.all([
 import("chalk").then((chalkModule) =>     chalkModule.default),
 import("inquirer").then((inquirerModule) =>     inquirerModule.default),
 spinnerPromise])
 .then(([chalkModule, inquirerModule, spinner]) =>     {
   chalk = chalkModule
   inquirer = inquirerModule
   spinner.succeed("Setup complete")
   askForRepoUrl()
 })
 .catch((err) =>     {
   spinnerPromise.then((spinner) =>     {
     spinner.fail("An error occurred during setup")
   })
   console.error(err)
 })

  async function askForRepoUrl( ) {
```

```
const questions: {
  type?: string,
  name: [
    "repoUrl",
    "optionalExcludedNames",
    "optionalExcludedExtensions",
    "addLineNumbers",
    "addLinting",
    "removeComments",
    "removeEmptyLines",
    "onePdfPerFile",
    "outputFileName",
    "outputFolderName",
    "keepRepo"
      ][number],
  message: string,
  validate?: (value: string      ) =>     boolean | string,
  filter?: (value: string      ) =>     boolean | string | string[],
  choices?: string[],
  default?: string | string[],
  when?: (answers: any   ) =>     boolean,
}[] = [
    {
      name: "repoUrl",
      message: "Please provide a GitHub repository URL:",
      validate: function (value: string      ) {
        var pass = value.match(
          /^https:\/\/github.com\/[A-Za-z0-9_.-]+\/[A-Za-z0-9_.-]+$/

            )
        if (pass) {
          return true
            }
        return "Please enter a valid GitHub repository URL."

          },
    },
    {
      name: "optionalExcludedNames",
      message:
        "Please provide a list of file names to exclude, separated by commas:",
      filter: function (value: string      ) {
        return value.split(",").map((v) =>     v.trim())
      },
    },
    {
      name: "optionalExcludedExtensions",
      message:
        "Please provide a list of file extensions to exclude, separated by commas:",
      filter: function (value: string      ) {
        return value.split(",").map((v: string      ) =>     v.trim())
      },
    },
    {
      name: "addLineNumbers",
      message: "Do you want to add line numbers to the PDF?",
      choices: ["Yes", "No"],
      filter: function (val: string      ) {
        return val.toLowerCase() === "yes"
```

```
          },
      },
      {
        name: "addLinting",
        message: "Do you want to add linting to the PDF?",
        choices: [/*"Yes",*/ "No"],
        filter: function (val: string      ) {
          return val.toLowerCase() === "yes"
          },
      },
      {
        name: "removeComments",
        message: "Do you want to remove comments from the PDF?",
        choices: [/*"Yes",*/ "No"],
        filter: function (val: string      ) {
          return val.toLowerCase() === "yes"
          },
      },
      {
        name: "removeEmptyLines",
        message: "Do you want to remove empty lines from the PDF?",
        choices: [/*"Yes",*/ "No"],
        filter: function (val: string      ) {
          return val.toLowerCase() === "yes"
          },
      },
      {
        name: "onePdfPerFile",
        message: "Do you want to make one PDF per file?",
        choices: [/*"Yes",*/ "No"],
        filter: function (val: string      ) {
          return val.toLowerCase() === "yes"
          },
      },
      {
        name: "outputFileName",
        message: "Please provide an output file name:",
        default: "output.pdf",
        when(answers: { onePdfPerFile: any }  ) {
          return !answers.onePdfPerFile
          },
      },
      {
        name: "outputFolderName",
        message: "Please provide an output folder name:",
        default: "./output",
        when(answers: { onePdfPerFile: any }  ) {
          return answers.onePdfPerFile
          },
      },
      {
        type: "list",
        name: "keepRepo",
        message: "Do you want to keep the cloned repository?",
        choices: ["Yes", "No"],
        filter: function (val: string      ) {
          return val.toLowerCase() === "yes"
          },
      },
    ]

```

```
  console.log(
    chalk.cyanBright(`

%^%^%^%^%^%^%W %^%^%^%^%^%^%^%W%^%^%^%^%^%^%W  %^%^%^%^%^%^%W         %^%^%^%^%^%^%W
%^%^%T%P%P%^%^%W%^%^%T%P%P%P%P%]%^%^%T%P%P%^%^%W%^%^%T%P%P%P%^%^%W        %Z%P%P%P%P%^
%^%^%^%^%^%^%T%]%^%^%^%^%^%W  %^%^%^%^%^%^%T%]%^%^%Q   %^%^%Q         %^%^%^%^%^%^%T%]
%^%^%T%P%P%^%^%W%^%^%T%P%P%]  %^%^%T%P%P%P%] %^%^%Q   %^%^%Q        %^%^%T%P%P%P%]
%^%^%Q  %^%^%Q%^%^%^%^%^%^%^%W%^%^%Q     %Z%^%^%^%^%^%^%T%]        %^%^%^%^%^%^%^%W
%Z%P%]  %Z%P%]%Z%P%P%P%P%P%P%]%Z%P%]      %Z%P%P%P%P%P%]         %Z%P%P%P%P%P%P%]

Welcome to Repo-to-PDF! Let's get started...
`)
  )

  const answers = await inquirer.prompt(questions)
  console.log(chalk.cyanBright("\nProcessing your request...\n"))
  main(
    answers.repoUrl,
    answers.optionalExcludedNames,
    answers.optionalExcludedExtensions,
    answers.addLineNumbers,
    answers.addLinting,
    answers.removeComments,
    answers.removeEmptyLines,
    answers.onePdfPerFile,
    answers.outputFileName,
    answers.outputFolderName,
    answers.keepRepo
      )
}

    async function main(
  repoUrl: string,
  optionalExcludedNames: any,
  optionalExcludedExtensions: any,
  addLineNumbers: any,
  addLinting: any,
  removeComments: any,
  removeEmptyLines: any,
  onePdfPerFile: any,
  outputFileName: fs.PathLike,
  outputFolderName: any,
  keepRepo: any
        ) {
  const gitP = git()
  const tempDir = "./tempRepo"
      const doc = new PDFDocument()
  doc.pipe(fs.createWriteStream(outputFileName))

  let fileCount = 0
      const spinner = ora(chalk.blueBright("Cloning repository...")).start
()

  gitP
    .clone(repoUrl, tempDir)
    .then(() => {
      spinner.succeed(chalk.greenBright("Repository cloned successfully"))
      spinner.start(chalk.blueBright("Processing files..."))
      appendFilesToPdf(
        tempDir,
        optionalExcludedNames,
        optionalExcludedExtensions
      ).then(() => {
        doc.end()
        spinner.succeed(
          chalk.greenBright(`PDF created with ${fileCount} files
processed.`          )
        )
        if (!keepRepo) {
          fs.rmSync(tempDir, { recursive: true, force: true })
          spinner.succeed(
            chalk.greenBright("Temporary repository has been deleted.")
          )
        }
      })
    })
    .catch((err) =>     {
      spinner.fail(chalk.redBright("An error occurred"))
      console.error(err)
    })

  async function appendFilesToPdf(
    directory: string,
    optionalExcludedNames: any,
    optionalExcludedExtensions: any
        ) {
    const files = await fsPromises.readdir(directory)
    for (let file of files) {
      const filePath = path.join(directory, file)
      const stat = await fsPromises.stat(filePath)

      const excludedNames = [
        ".gitignore",
        ".gitmodules",
        "package-lock.json",
        "yarn.lock",
        ".git",
      ]
      excludedNames.push(...optionalExcludedNames)

      const excludedExtensions = [
        ".png",
        ".yml",
        ".jpg",
        ".jpeg",
        ".gif",
        ".svg",
        ".bmp",
        ".webp",
        ".ico",
        ".mp4",
        ".mov",
        ".avi",
        ".wmv",
      ]
      excludedExtensions.push(...optionalExcludedExtensions)

      // Check if file or directory should be excluded

          if (
        excludedNames.includes(path.basename(filePath)) ||
        excludedExtensions.includes(path.extname(filePath))
```

```
   ) {
    continue
     }

   if (stat.isFile()) {
    fileCount++
    spinner.text = chalk.blueBright(
     `Processing files... (${fileCount} processed)`

      )
    let fileName = path.relative(tempDir, filePath)
    if (isBinaryFileSync(filePath)) {
     const data = fs.readFileSync(filePath).toString("base64")
     doc
      .addPage()
      .font("Courier")
      .fontSize(10)
      .text(`${fileName}\n\nBASE64:\n\n${data}` , { lineGap: 4 })
    } else {
     let data = await fsPromises.readFile(filePath, "utf8")
     data = data.replace(/
/g, "\n")
     data = data.replace(/\r\n/g, "\n")
     data = data.replace(/\r/g, "\n")

     doc
      .addPage()
      .font("Courier")
      .fontSize(10)
      .text(`${fileName}\n\n`     , { lineGap: 4 })

     const highlightedCode = hljs.highlight(data, { language: "ps1" }).value
       const hlData = htmlToJson(highlightedCode);
     let lineNum = 1;
     for (let i = 0; i < hlData.length; i++) {
      const { text, color } = hlData[i];
      if (i == 0 || hlData[i - 1]?.text === "\n")
       doc.text(String(lineNum++).padStart(3, " "), { continued: true });

      if (text !== "\n") doc.text(text, { continued: true });
      else doc.text(text);

      if (color) doc.fillColor(color);
      else doc.fillColor("black");
     }
    }
   } else if (stat.isDirectory()) {
    await appendFilesToPdf(
     filePath,
     optionalExcludedNames,
     optionalExcludedExtensions
    )
   }
  }
 }

 doc.on("finish", () => {
  spinner.succeed(
   chalk.greenBright(`PDF created with ${fileCount} files processed.`
)
    )
  })
}

```

src\hljstest.ts

```
const hljs = require("highlight.js")
   const { htmlToJson } = require("./syntax")

   // Here's a simple JavaScript code snippet

   const code = `
function helloWorld() {
  console.log("Hello, world!");
}
helloWorld();
`

   // Here, we're using the 'javascript' language for highlighting

   const highlightedCode = hljs.highlight(code, { language: "js" }).value


   console.log(highlightedCode);
   const data = htmlToJson(highlightedCode);
   console.log(data);

```

src\syntax.ts

```
/**
 * @param {string} htmlCode
 */
    export function htmlToJson(htmlCode: string      ): { text: string, color?: string }[] {
  const originalCode = htmlCode;
  /**
   * @type {{text: string, color?: string}[]}
   */
      const data: { text: string, color?: string }[] = [];
  const elementRegex = /^<span\s+class="hljs-([^"]+)"[^>]*>([^<]*)(?:<\/span>)?/;
  const nonelementRegex = /[^<]*/;
  while (htmlCode) {
    const match = htmlCode.match(elementRegex);
    if (match) {
      const fullText = match[0];
      const cls = match[1];
      const text = match[2];
      let color = "black";
      // const color = cls;
          const type = cls.split(" ")[0].toLowerCase() ?? "unknown";
      switch (type) {
        case "comment":
          color = "#697070";
          break;
        case "punctuation":
        case "tag":
          color = "#444a";
          break;
        case "attribute":
        case "doctag":
        case "keyword":
        case "meta":
        case "keyword":
        case "name":
        case "selector-tag":
          color = "#7ddcfe";
          break;
        case "deletion":
        case "number":
        case "quote":
        case "selector-class":
        case "selector-id":
        case "string":
        case "template-tag":
        case "type":
        case "section":
        case "title":
          color = "#800";
          break;
        case "link":
        case "operator":
        case "regexp":
        case "selector-attr":
        case "selector-pseudo":
        case "symbol":
        case "template-variable":
```

```
        case "variable":
          color = "#ab5656";
          break;
        case "literal":
          color = "#695";
          break;
        case "addition":
        case "built_in":
        case "bullet":
        case "code":
          color = "#397300";
          break;
        case "meta":
          color = "#1f7199";
          break;
        case "string":
          color = "#38a";
          break;
      }
      console.log({ type, text, color, fullText });
      data.push({ text, color });
      htmlCode = htmlCode.slice(fullText.length);
    }
    else if (htmlCode.startsWith("</span>")) { // Failed ending from hljs

          const text = "</span>";
      data.push({ text: "" }); // Empty text on purpose

          htmlCode = htmlCode.slice(text.length);
    }
    else if (htmlCode.startsWith("\n")) {
      const text = "\n";
      htmlCode = htmlCode.slice(1);
      data.push({ text });
    }
    else {
      const match = htmlCode.match(nonelementRegex);
      const text = match![0];
      htmlCode = htmlCode.slice(text.length);
      data.push({ text });
    }
  }

  /**
   * @type {{text: string, color?: string}[]}
   */
      const fixedData: { text: string, color?: string }[] = [];
  // Fix newlines
      for (let i = 0; i < data.length; i++) {
    const { text, color } = data[i];
    const lines = text.split("\n");
    for (let j = 0; j < lines.length; j++) {
      const line = lines[j];
      if (j > 0) fixedData.push({ text: "\n" });
      fixedData.push({ text: line, color });
    }
  }

  return fixedData;
}
```

tsconfig.json

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

        /* Projects */
        // "incremental": true,                              /*
Save .tsbuildinfo files to allow for incremental compilation of projects. */

        // "composite": true,                                /* Enable
constraints that allow a TypeScript project to be used with project
references. */
        // "tsBuildInfoFile": "./.tsbuildinfo",              /* Specify
the path to .tsbuildinfo incremental compilation file. */

        // "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 'Fragment'. */
        // "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. */
```

```
    // "moduleDetection": "auto",                          /* Control what method is used to detect module-format JS files. */

    /* Modules */
    "module": "CommonJS",                                  /* Specify what module code is generated. */
    // "rootDir": "./",                                    /* Specify the root folder within your source files. */
    "moduleResolution": "Node16",                         /* 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. */
    // "moduleSuffixes": [],                               /* List of file name suffixes to search when resolving a module. */
    // "resolveJsonModule": true,                          /* Enable importing .json files. */
    // "noResolve": true,                                  /* Disallow 'import's, 'require's or '<reference>'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 'allowJs'. */

    /* 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 that bundles all .d.ts output. */

        "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,               /* Default catch clause variables as 'unknown' instead of 'any'. */
        // "alwaysStrict": true,                             /* Ensure 'use strict' is always emitted. */
        // "noUnusedLocals": true,                           /* Enable error reporting when 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,                 /* Add 'undefined' to a type when accessed using an index. */
```

```
    // "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"
  ],
  "exclude": [
    "node_modules",
    "dist"
  ]
}
```