codexer

Table of Contents

1. README.md 1
2. package.json 3
3. src
    1. cli.js 4
    2. fileProcessing.js 6
    3. header.html 8
    4. highlighter.js 11
    5. index.js 13
    6. makeEntries.js 16
    7. makeTOC.js 18
    8. makeTitlePage.js 21
    9. tocFooter.html 22
    10. utils.js 24
4. test.html 25

README.md

codexer

make a book out of a (code) directory

The de-facto way of looking at code is as a dry, machine-like, monospaced, almost monotonous form of giving instructions to the computer. But I think code is just as creative, colorful, vivid, and *textual* as all of the other forms of information we consume. Our unconscious perception of a divide between computer programs and other forms of text prevents us from critiquing the actual language of code, down to its individual variables and functions.

This (small) project tries to push back on that by providing a method to make folders of code into books -- or really, just printable PDFs. It features syntax highlighting with highlight.js, automagical formatting with prettier, and a table of contents with page numbers from my own hacky coding.

The actual PDF generation is done with html-pdf with some processing done in pdf-parse.

Usage (CLI)

Install with `npm install --global codexer`.

If you're new to Node, feel free to check out this guide.

```
codexer [options]


Options:
  -V, --version               output the version number
  -t, --target <path>         The path to a directory to be made into a PD
  -a, --author <name>         will be filled in on the title page and ever
  -o, --outPath <path>        output location. Tilde notation currently no
  -d, --dry                   Add -d to enable a dry run that produces onl
      representing the order in which files will be assembled.
  -j, --json <path>           Pass in the location of a JSON file to speci
      Most useful after trying out the -d option to see what a config shou
  -dh --dryHTML               Like the dry run option, but produces just H
  -s --stylePath <path>       Path to a HTML file with configurations that
  -q --quietly                Suppress all debugging messages
  -e --exclude <patterns...>  Specify regex patterns for files to exclude.
                              .git, yarn.lock and package-lock.json, and .
                              Codexer also excludes any non-text encoded f
  -h, --help                  display help for command
```

Usage (as a Node module)

You should probably stick to using it as a CLI tool or for just messing around, as it's not really ready for production. But if you want, you can install with `npm install codexer` in your Node project. Then use it as follows:

```
const codexer = require('codexer')
// or
import codexer from 'codexer'


codexer('.')
// Finished! PDF is located at /tmp/codexer/[your directory basename].pdf



codexer('.', {outPath: 'output.pdf'})
// Finished! PDF is located at [path to your directory].pdf
```

All of the CLI options are available for use in the npm version.

npx?

You can also call it as a single-use tool with npx if you have Node installed by prefixing npx to the codexer commands, e.g. by running `npx codexer .`.

The problem is that phantomJS and pdf-parse, which this tool depends on, are incredibly large (the local node_modules/ folder on my machine comes out to 184Mb). Download, load, and install times are quite long because of this.

If you intend on using this tool more than once, I'd recommend just installing it globally.

If this doesn't fit your use case

All in all, it probably doesn't! Please, please feel free to download and alter it if you want an adjustment.

package.json

```
{
  "name": "codexer",
  "author": {
    "name": "Nathan Kim",
    "email": "nathanckim18@gmail.com"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/18kimn/codexer.git"
  },
  "version": "0.2.5",
  "description": "",
  "main": "src/index.js",
  "type": "module",
  "bin": {
    "codexer": "src/cli.js"
  },
  "keywords": [],
  "license": "Hippocratic-2.1",
  "dependencies": {
    "commander": "^8.2.0",
    "highlight.js": "^11.3.0",
    "html-pdf": "^3.0.1",
    "istextorbinary": "^6.0.0",
    "json2md": "^1.12.0",
    "marked": "^3.0.7",
    "merge-pdf-buffers": "^1.0.3",
    "pdf-parse": "18kimn/pdf-parse-patch#cf9752dacc8dcdc31d1caf59c75e2fd82
    "prettier": "^2.4.1"
  }
}
```

src/cli.js

```
#!/usr/bin/env node
import {program} from 'commander'
import main from './index.js'

program.version('0.1.0')

program
  .option(
    '-t, --target <path>',
    'The path to a directory to be made into a PDF. The flag can be omitte
  )
  .option(
    '-a, --author <name>',
    'will be filled in on the title page and every header',
  )
  .option(
    '-o, --outPath <path>',
    'output location. Tilde notation currently not accepted :(',
  )
  .option(
    '-d, --dry',
    `Add -d to enable a dry run that produces only a JSON
    representing the order in which files will be assembled.`,
  )
  .option(
    '-j, --json <path>',
    `Pass in the location of a JSON file to specify your own order of file
    Most useful after trying out the -d option to see what a config should
  )
  .option(
    '-dh --dryHTML',
    'Like the dry run option, but produces just HTML output. No page numbe
  )
  .option(
    '-s --stylePath <path>',
    'Path to a HTML file with configurations that will be used to style th
  )
  .option('-q --quietly', 'Suppress all debugging messages')
  .option('-w --width <length>', 'Page width')
  .option('-h --height <length>', 'Page height')
  .option(
    '-e --exclude <patterns...>',
    `Specify regex patterns for files to exclude. Default excludes node_mo
.git, yarn.lock and package-lock.json, and .env files.
Codexer also excludes any non-text encoded files; this cannot be altered.`
  )
```

```
program.parse()

const opts = program.opts()
const target = opts.target || program.args[0]
delete opts.target

main(target, program.opts())
```

src/fileProcessing.js

```
import fs from 'fs'
import path from 'path'
import {isBinary} from 'istextorbinary'

/*
  getAllFiles(dirPath, exclusions, container)
    - dirPath: path of directory to make into a book
    - exclusions: regular expressions to exclude directories and files.
    - container: Used in file recursion, do not touch yourself
  Returns an array, with strings for filenames and objects for directories
    Directory objects are just one item long, with that one item being nam
*/

const getAllFiles = (dirPath, exclusions, container = []) => {
  if (exclusions.some((exclude) => exclude.test(dirPath))) return

  const files = fs.readdirSync(dirPath)
  files.forEach((file) => {
    const inExclusions = exclusions.some((exclude) =>
      exclude.test(file.toLowerCase()),
    )
    const notText = isBinary(file.toLowerCase())
    if (inExclusions || notText) return
    fs.statSync(dirPath + '/' + file).isDirectory()
      ? container.push({
          [file]: getAllFiles(path.join(dirPath, '/', file), exclusions, [
        })
      : container.push(file)
  })
  return container
}

/*
  Converts the nested array returned by getAllFiles into a flat array.
*/

const fileObjToArray = (nested, currentPath = '') => {
  const prefix = currentPath ? currentPath + '/' : currentPath
  // nested = array of items (strings and objects), represents one level o
  const flattened = nested.reduce((prev, curr) => {
    const toAppend =
      typeof curr === 'string'
        ? [path.join(prefix, curr)]
        : fileObjToArray(Object.values(curr)[0], prefix + Object.keys(curr
    return [...prev, ...toAppend]
  }, [])
```

```
  return flattened
}

export {getAllFiles, fileObjToArray}
// // usage examples
// const obj = getAllFiles('../yale-detour', [], [/node_modules/, /\.git/]
// const arr = fileObjToArray(obj, '../yale-detour')
// fs.writeFileSync('obj.json', JSON.stringify(obj))

// fs.writeFileSync('arr.json', JSON.stringify(arr))
```

src/header.html

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<style>
  pre code.hljs {
    display: block;
    overflow-x: auto;
    padding: 1em;
  }
  code.hljs {
    padding: 3px 5px;
  } /*!
  Theme: GitHub
  Description: Light theme as seen on github.com
  Author: github.com
  Maintainer: @Hirse
  Updated: 2021-05-15

  Outdated base version: https://github.com/primer/github-syntax-light
  Current colors taken from GitHub's CSS
*/
  .hljs {
    color: #24292e;
    background: #fff;
  }
  .hljs-doctag,
  .hljs-keyword,
  .hljs-meta .hljs-keyword,
  .hljs-template-tag,
  .hljs-template-variable,
  .hljs-type,
  .hljs-variable.language_ {
    color: #d73a49;
  }
  .hljs-title,
  .hljs-title.class_,
  .hljs-title.class_.inherited__,
  .hljs-title.function_ {
    color: #6f42c1;
  }
  .hljs-attr,
  .hljs-attribute,
  .hljs-literal,
  .hljs-meta,
  .hljs-number,
  .hljs-operator,
  .hljs-selector-attr,
  .hljs-selector-class,
```

```
.hljs-selector-id,
.hljs-variable {
  color: #005cc5;
}
.hljs-meta .hljs-string,
.hljs-regexp,
.hljs-string {
  color: #032f62;
}
.hljs-built_in,
.hljs-symbol {
  color: #e36209;
}
.hljs-code,
.hljs-comment,
.hljs-formula {
  color: #6a737d;
}
.hljs-name,
.hljs-quote,
.hljs-selector-pseudo,
.hljs-selector-tag {
  color: #22863a;
}
.hljs-subst {
  color: #24292e;
}
.hljs-section {
  color: #005cc5;
  font-weight: 700;
}
.hljs-bullet {
  color: #735c0f;
}
.hljs-emphasis {
  color: #24292e;
  font-style: italic;
}
.hljs-strong {
  color: #24292e;
  font-weight: 700;
}
.hljs-addition {
  color: #22863a;
  background-color: #f0fff4;
}
.hljs-deletion {
  color: #b31d28;
  background-color: #ffeef0;
}
```

```
</style>

<style>
  @import url('https://fonts.googleapis.com/css2?family=Fira+Code:wght@300
  @import url('https://fonts.googleapis.com/css2?family=Crimson+Text:ital,
  @page {
    @bottom-center {
      content: counter(page);
    }
  }
  html,
  body {
    font-family: 'Crimson Text';
  }
  code {
    font-family: 'Fira Code';
  }
  code,
  span,
  div,
  p {
    page-break-before: avoid !important;
    font-size: 8pt;
  }

  tr {
    page-break-before: avoid;
    display: 'inline-block';
  }

  body {
    page-break-before: avoid;
  }

  pre {
    -ms-overflow-style: none; /* IE and Edge */
    scrollbar-width: none; /* Firefox */
  }

  pre::-webkit-scrollbar {
    display: none;
  }
</style>
```

src/highlighter.js

```
import hljs from 'highlight.js'
import {promises as fs} from 'fs'
import {extname, join} from 'path'
import {makeLink, langs, updateConsole} from './utils.js'
import prettier from 'prettier'
import marked from 'marked'

Object.keys(langs).map((lang) => hljs.registerLanguage(lang, langs[lang]))

const prettierSupported = prettier
  .getSupportInfo()
  .languages.reduce((prev, curr) => [...prev, ...curr?.extensions], [])

const highlight = async (filename, dirname, opts) => {
  const {quietly, index, totalLength} = opts

  const basefilename = filename
  filename = join(dirname, filename)
  const text = await fs.readFile(filename, 'utf-8')
  const ext = extname(filename).substring(1)

  const prettified = prettierSupported.includes(extname(filename))
    ? prettier.format(text, {
        // see https://prettier.io/docs/en/options.html
        semi: false,
        filepath: filename,
        singleQuote: true,
        quoteProps: 'preserve',
        bracketSpacing: false,
        trailingComma: 'all',
        printWidth: 80,
        proseWrap: 'always',
      })
    : text

  let formatted
  if (ext === 'md' || !ext) {
    // we want to render markdown text, not just format it
    formatted = marked(prettified)
  } else if (ext.replace('.', '') in Object.keys(langs)) {
    // for a few cases where hljs gets weird we need to manually specify l
    formatted = hljs.highlight(prettified, {language: ext}).value
  } else {
    // but for most cases highlightAuto works fine
    formatted = hljs.highlightAuto(prettified).value
  }
```

```
  const linkTag = makeLink(basefilename)

  const codewrapper =
    ext === 'md' ? formatted : `<pre><code>${formatted}</pre></code>`
  updateConsole(quietly, `Highlighting entries... ${index + 1}/${totalLeng

  return `

## <a name="${linkTag}">${basefilename}</a>

${codewrapper}

  `
}

export default highlight
```

src/index.js

```
import {getAllFiles, fileObjToArray} from './fileProcessing.js'
import highlight from './highlighter.js'
import makeTOC from './makeTOC.js'
import {promises as fs} from 'fs'
import {join, resolve, basename, dirname} from 'path'
import {platform, tmpdir} from 'os'
import makeTitlePage from './makeTitlePage.js'
import makeEntries from './makeEntries.js'
import {merge} from 'merge-pdf-buffers'
import {fileURLToPath} from 'url'
import {updateConsole} from './utils.js'
import marked from 'marked'
const __dirname = dirname(fileURLToPath(import.meta.url))

const main = async (target, options) => {
  const {
    author,
    outPath,
    dry,
    json,
    stylePath: headerPath,
    dryHTML: html,
    quietly,
    exclude,
  } = options || {}

  const {width, height} = options || {}
  const dims = {width: width || '152mm', height: height || '228mm'}
  if (!target) return
  const baseHeader = await fs.readFile(
    headerPath || join(__dirname, './header.html'),
  )

  // on linux, the default dpi messes phantomjs up.
  // See https://github.com/marcbachmann/node-html-pdf/issues/525
  const zoom = `
<style>
html {
  zoom: ${platform() === 'linux' ? 0.753 : 1};
}
</style>
`
  const header = zoom + baseHeader

  // we want to safely save to a temp directory
  //  if a specific path is not given
```

```
const outDir = outPath ? dirname(target) : join(tmpdir(), 'codexer')
fs.mkdir(outDir, {recursive: true}, (err) => {
  if (err && err !== 'EEXIST') throw err
})
const resolvedOutPath =
  outPath || join(outDir, `${basename(resolve(target))}.pdf`)

// handle user-specified exclusions
const defaultExclusions = [/node_modules/, /\.git/, /lock/, /\.env/]
const exclusions = exclude?.map((str) => new RegExp(str)) || defaultExcl

let fileObj
if (dry) {
  const json = getAllFiles(resolve(target), exclusions)
  const jsonPath = join(outDir, `${basename(resolve(target))}.json`)
  await fs.writeFile(jsonPath, JSON.stringify(json))
  console.log(`dry run finished; json written to ${jsonPath}`)
  return
} else if (typeof json === 'string') {
  fileObj = JSON.parse(await fs.readFile(resolve(json), {encoding: 'utf-
} else {
  fileObj = getAllFiles(resolve(target), exclusions)
}

const fileArray = fileObjToArray(fileObj)

// creating buffers for each section
const entries = await Promise.all(
  fileArray.map((filename, index) => {
    const opts = {quietly, index, totalLength: fileArray.length}
    return highlight(filename, target, opts)
  }),
)

if (html) {
  const htmlPath = join(outDir, `${basename(resolve(target))}.html`)
  await fs.writeFile(
    htmlPath,
    header +
      marked(entries.join('<div style="page-break-after: always;"></div>
  )
  updateConsole(quietly, `HTML file written to ${htmlPath}\n`)
  return
}

updateConsole(quietly, 'creating title page and table of contents buffer
const titleBuffer = makeTitlePage(target, author || '', dims)
const entryBuffers = await makeEntries(entries, header, quietly, dims)
const tocBuffer = makeTOC(fileObj, header, entryBuffers, dims)
```

```
  updateConsole(quietly, 'writing file to disk...')
  // assembling buffers and writing
  return Promise.all([titleBuffer, tocBuffer, ...entryBuffers])
    .then((buffers) => merge(buffers))
    .then((merged) => fs.writeFile(resolvedOutPath, merged))
    .then(() =>
      updateConsole(
        quietly,
        `Finished! PDF is located at ${resolvedOutPath}\n`,
      ),
    )
}

// main('../../work/aemp/evictorbook', 'The Anti-Eviction Mapping Project'

export default main
```

src/makeEntries.js

```
import pdf from 'html-pdf'
import marked from 'marked'
import {baseOpts, getPageCount, updateConsole} from './utils.js'

// first iterate through each document and make one pdf for each of them.
// this unfortunately cannot be done async since we need to know the lengt

const getBuffer = async (entry, pageStartOffset = 0, opts) => {
  const {header, index, totalLength, quietly, dims} = opts
  const footer = {
    height: '20mm',
    contents: {
      default: `<div id="pageNum" style="width: 100%; height:100%; text-al
          display: flex; place-items: center; place-content: center;">{{pa
          <script>
            const div = document.getElementById("pageNum")
            div.innerText = Number(div.innerText) + ${pageStartOffset}
          </script>`,
    },
  }

  const html = marked(header + entry)
  return new Promise((resolve) => {
    pdf
      .create(html, {...baseOpts, footer, ...dims})
      .toBuffer(async (_, buffer) => {
        updateConsole(
          quietly,
          `creating buffers for entries...${index + 1}/${totalLength}`,
        )
        resolve(buffer)
      })
  })
}

const makeEntries = async (entries, header, quietly, dims) => {
  let pageOffset = 0
  const buffers = entries.reduce(async (prev, curr, index) => {
    const opts = {header, index, totalLength: entries.length, quietly, din
    const prevArray = await prev
    const lastBuffer = await prevArray[prevArray.length - 1]
    pageOffset = lastBuffer
      ? pageOffset + (await getPageCount(lastBuffer))
      : pageOffset
    const currentBuffer = await getBuffer(curr, pageOffset, opts)
```

```
    return [...prevArray, currentBuffer]
  }, new Promise((resolve) => resolve([])))

  return buffers
}

export default makeEntries
```

src/makeTOC.js

```
/*
  give the file object from fileProcessing/getAllFiles as input,
  produce a markdown nested numbered list

  This is mostly a recursive solution that adds indentation levels with ea
    recursion. It's wrapped in a function that adds `.join('\n')` on it
    to make the array into a file-writable string
*/
import marked from 'marked'
import {fileObjToArray} from './fileProcessing.js'
import {baseOpts, getPageCount} from './utils.js'
import prettier from 'prettier'
import pdf from 'html-pdf'
import {promises as fs} from 'fs'
import {fileURLToPath} from 'url'
import {join} from 'path'
import {dirname} from 'path'

const makeRow = (filename, pageNum, indent, index) => {
  const spaces = Array(indent).fill(' ').join('')
  const item = `<span>${index + 1}. ${filename}</span>`
  return `|<span>${spaces}<span/>${item}| ${pageNum}  |`
}

const makeTOCArray = async (
  fileObj,
  fileArray,
  pageCounts,
  indent = 0,
  parent = '',
) => {
  parent = parent ? parent + '/' : ''

  return fileObj.reduce(async (prev, curr, index) => {
    prev = await prev
    const filename = typeof curr === 'string' ? curr : Object.keys(curr)[0
    if (!Object.values(curr)[0].length) return prev
    const fileIndex = fileArray.findIndex((node) => node === parent + file
    const pageNum = fileIndex === 0 ? 1 : (await pageCounts[fileIndex - 1]
    const row = makeRow(filename, pageNum || '', indent, index)
    const toAdd =
      typeof curr === 'string'
        ? [row]
        : [
            row,
            ...(await makeTOCArray(
```

```
              Object.values(curr)[0],
              fileArray,
              pageCounts,
              indent + 4,
              parent + filename,
            )),
          ]
    return [...prev, ...toAdd]
  }, new Promise((resolve) => resolve([])))
}

const makeTOC = async (fileObj, header, entryBuffers, dims) => {
  const footer = {
    height: '20mm',
    contents: {
      default: await fs.readFile(
        join(dirname(fileURLToPath(import.meta.url)), 'tocFooter.html'),
        'utf-8',
      ),
    },
  }

  const pageCountPromises = Promise.all(entryBuffers.map(getPageCount))
  const pageCounts = (await pageCountPromises)
    .reduce((prev, curr) => [...prev, curr + prev[prev.length - 1]], [0])
    .slice(1)

  const fileArray = fileObjToArray(fileObj)
  const tocPromises = await makeTOCArray(fileObj, fileArray, pageCounts)
  const tocArray = await Promise.all(tocPromises)
  const tocMd = `

  <div style="width:100%;text-align:center;">
  <a name="toc"><h1>Table of Contents</h1></a>
  </div>

  |  |  |
  ---------| -------------
  ${tocArray.join('\n')}
  `

  const prettiered = prettier.format(tocMd, {
    filepath: 'placeholder.md',
  })
  const html = marked(`
${header}
<style>
table{width: 100%;}
span, tr{font-size: 10pt;}
</style>
```

```
${prettiered}`)

  return new Promise((resolve) => {
    pdf.create(html, {...baseOpts, footer, ...dims}).toBuffer((_, buffer)
      resolve(buffer)
    })
  })
}

export default makeTOC
```

src/makeTitlePage.js

```
import {basename, resolve} from 'path'
import {baseOpts} from './utils.js'
import pdf from 'html-pdf'
const makeTitlePage = (target, author, dims) => {
  const titlePage = `
<div style="width:100%; text-align: center;">
  <div style="height: calc((${dims.height} - 1in) / 4);"></div>
  <h1 style="width: 100%;">${basename(resolve(target))}</h1>
  <h3 style="width: 100%;">${author}</h3>
  <div style="height: calc((${dims.height} - 1in) / 4);"></div>
</div>
`

  return new Promise((resolve) => {
    pdf.create(titlePage, {...baseOpts, ...dims}).toBuffer((_, buffer) =>
      resolve(buffer)
    })
  })
}

export default makeTitlePage
```

src/tocFooter.html

```
<div
  id="pageNum"
  style="
    width: 100%;
    height: 100%;
    text-align: center;
    display: flex;
    place-items: center;
    place-content: center;
  "
>
  {{page}}
</div>
<script>
  function romanize(num) {
    if (isNaN(num)) return NaN
    var digits = String(+num).split(''),
      key = [
        '',
        'C',
        'CC',
        'CCC',
        'CD',
        'D',
        'DC',
        'DCC',
        'DCCC',
        'CM',
        '',
        'X',
        'XX',
        'XXX',
        'XL',
        'L',
        'LX',
        'LXX',
        'LXXX',
        'XC',
        '',
        'I',
        'II',
        'III',
        'IV',
        'V',
        'VI',
        'VII',
```

```
        'VIII',
        'IX',
      ],
      roman = '',
      i = 3
    while (i--) roman = (key[+digits.pop() + i * 10] || '') + roman
    return Array(+digits.join('') + 1).join('M') + roman
  }
  const div = document.getElementById('pageNum')
  div.innerText = romanize(Number(div.innerText)).toLowerCase()
</script>
```

src/utils.js

```
import json from 'highlight.js/lib/languages/json'
import html from 'highlight.js/lib/languages/xml'
import yaml from 'highlight.js/lib/languages/yaml'
import pdfparse from 'pdf-parse'
// some languages are not included by default in hljs or have weird aliase

const langs = {
  json,
  html,
  yaml,
}

const makeLink = (filename) => {
  return filename.toLowerCase().replace(/\/|\./g, '-')
}

const getPageCount = async (buffer) =>
  pdfparse(buffer).then((data) => data.numpages)

const baseOpts = {
  timeout: '120000',
  border: {
    top: '0.5in',
    right: '0.5in',
    left: '0.5in',
  },
  zoomFactor: '0.753',
}

const updateConsole = (quietly, str) => {
  if (quietly) return
  process.stdout.clearLine()
  process.stdout.cursorTo(0)
  process.stdout.write(str)
}

export {makeLink, langs, getPageCount, baseOpts, updateConsole}
```

test.html

```
<body>
  <div style="width: 100%; text-align: center">
    <div style="height: 5cm"></div>
    <div style="height: calc((200mm - 1in) / 2 - 1in)"></div>
    <h1 style="width: 100%">test</h1>
    <h3 style="width: 100%"></h3>
  </div>
</body>
```