

Behind the Scenes of CodeSlide

```
<code id="slide">
```

CodeSlide

npm:codeslide-cli v0.12.4

Features

- CodeSlide makes a slideshow for code snippets
- Its applications:
 - [CodeSlide CLI](#)

Dependencies

- It uses [cross-fetch](#) as resource fetcher
- It uses [esbuild](#) as module bundler
- It uses [gray-matter](#) as YAML Front Matter parser
- It uses [Commander.js](#) as CLI framework
- It uses [Eta](#) as HTML template engine
- It uses [Highlight.js](#) as syntax highlighter
- It uses [Marked](#) as Markdown renderer
- It uses [Puppeteer](#) as PDF printer
- It uses [TypeScript](#) as the main programming language
- It uses [Zod](#) as JSON schema validator

Documents

- See [Reference](#) for more usage information
- See [Change Log](#) for more version information

Creator

- [AsherJingkongChen](#)
-

The abstract process

1. Build a **Renderer**
 2. Render the HTML template and CSS with the built renderer
 3. Print the slideshow to the output
-

Build a Renderer

```
export * from './Format';
export * from './Layout';
export * from './PageSize';
export * from './Renderer';
```

Renderer

```
import fetch from 'cross-fetch';
import hljs from 'highlight.js';
import matter from 'gray-matter';
import { Stylesheets, Template } from './slides';
import { isNode } from 'browser-or-node';
import { marked } from 'marked';
import { pathToFileURL } from 'url';
import { readFileSync } from 'fs';
import { render as renderEta } from 'eta';
import { z } from 'zod';
import { isFormat } from './Format';
import { isLayout } from './Layout';
import { isPageSize } from './PageSize';

export type Renderer = z.infer<typeof _Renderer>;

export namespace Renderer {
  export const parse = async (
    manifest: string
  ): Promise<Renderer> => {
    manifest = manifest.replace(
      /^[\u200B\u200C\u200D\u200E\u200F\uFEFF]/, ''
    );
    const { content, data } = matter(manifest);
    if (!data.codeslide) {
      throw new Error(
        'Cannot find "codeslide" scalar in the Front Matter section'
      );
    }

    const renderer = _Renderer.parse(data.codeslide);
    renderer.slides = await _parse(content)
      .then((html) => html.split('<hr>').map((s) => s.trim()));
    renderer.styles = await Promise.all(
      renderer.styles.map((path) => _getContent(path))
    );
  }
}
```

```

    );
    return renderer;
};

export const render = (
  renderer: Renderer
): string => renderEta(
  Template,
  {
    layout: renderer.layout,
    slides: renderer.slides,
    style: `
<style>
${
  [
    Stylesheets['github'],
    Stylesheets[renderer.layout],
    ...renderer.styles,
  ]
code {
  font-family: ${renderer.fontFamily};
  font-size: smaller;
}`,
  '\n
#slides {
  font-size: ${renderer.fontSize};
  font-weight: ${renderer.fontWeight};
  line-height: 1.5;
}`,
  ].join('\n')
}
</style>`,
  {
    autoTrim: false,
    tags: ['{%', '%}'],
  }
);
}

export const _Renderer = z.object({
  fontFamily: z
    .string()
    .default('')
    .transform((arg) => `
${arg ? `${arg}`, ` : ''}ui-monospace, SFMono-Regular, \
SF Mono, Menlo, Consolas, Liberation Mono, monospace`
    ),
  fontSize: z
    .string()
    .default('large'),
  fontWeight: z

```

```

        .string()
        .default('normal'),
format: z
    .string()
    .refine(isFormat)
    .default('html'),
layout: z
    .string()
    .refine(isLayout)
    .default('horizontal'),
pageSize: z
    .string()
    .refine(isPageSize)
    .default('A4'),
slides: z
    .array(z.string())
    .default([]),
styles: z
    .array(z.string())
    .default([]),
})
.transform((arg) => {
    if (
        arg.layout === 'horizontal' &&
        arg.format === 'pdf'
    ) {
        arg.layout = 'vertical';
    }
    return arg;
});

const _parse = async (manifest: string) => (
    marked.parse(manifest, {
        async: true,
        highlight: (code, language) => (
            hljs.highlight(code, { language }).value
        ),
        walkTokens: async (token: marked.Token) => {
            if (token.type === 'link') {
                const { href, text, raw } = token;
                if (!text.startsWith(':')) {
                    return;
                }
                const [prefix, suffix] = <[string, string | undefined]>
                    text.split('.');
                if (prefix === ':slide') {
                    token = _toHTMLToken(token);
                    token.raw = raw;
                    token.text = await _getContent(href)
                        .then((content) => _parse(content));
                } else if (prefix === ':code') {
                    token = _toHTMLToken(token);

```

```

        token.raw = raw;
        const code = await _getContent(href).then((content) => (
            hljs.highlight(content, {
                language: suffix ?? 'plaintext'
            })
        ));
        token.text = `

```

```

<pre><code${
    code.language ? ` class="language-${code.language}"` : ''
}>${
    code.value
}</code></pre>`;
    }
    },
}
));

```

```

const _toHTMLToken = (
    token: marked.Token
): marked.Tokens.HTML => {
    for (const p in token) {
        if (token.hasOwnProperty(p)){
            delete token[p as keyof marked.Token];
        }
    }
    token = token as marked.Token;
    token.type = 'html';
    token = token as marked.Tokens.HTML;
    token.pre = false;
    return token;
};

```

```

const _getContent = async (
    path: string | URL,
): Promise<string> => {
    if (typeof path === 'string') {
        path = _parseURL(path);
    }
    if (path.protocol === 'file:') {
        return readFileSync(path).toString();
    } else {
        return fetch(path).then(async (r) => {
            if (r.ok) { return r.text(); }
            throw new Error(await r.text());
        });
    }
};

```

```

const _parseURL = (path: string): URL => {
    try {
        return new URL(path);
    }

```

```
} catch (err) {  
  if (isNode) {  
    return pathToFileURL(path);  
  }  
  throw err;  
}  
};
```

Build a Renderer

Options

- Export slideshow file as HTML or PDF format

```
export type Format = keyof typeof Format;
```

```
export const Format = {  
  html: true,  
  pdf: true,  
} as const;
```

```
export const isFormat = (  
  raw: string  
): raw is Format => raw in Format;
```

- Specify page size for PDF format

```
export type PageSize = keyof typeof PageSize;
```

```
export const PageSize = {  
  letter: true,  
  legal: true,  
  tabloid: true,  
  ledger: true,  
  A0: true,  
  A1: true,  
  A2: true,  
  A3: true,  
  A4: true,  
  A5: true,  
  A6: true,  
} as const;
```

```
export const isPageSize = (  
  raw: string  
): raw is PageSize => raw in PageSize;
```

- Present the slideshow in horizontal or vertical layout

```
export type Layout = keyof typeof Layout;

export const Layout = {
  'horizontal': true,
  'vertical': true,
} as const;

export const isLayout = (
  raw: string
): raw is Layout => raw in Layout;
```

The HTML template

CodeSlide uses [Eta](#) to render {% and %} are interpolation symbols

```
<!DOCTYPE HTML>
<html class="hljs">
<head>
  <meta
    name="viewport"
    charset="utf-8"
    content="width=device-width, initial-scale=1, user-scalable=no">
    {%~ it.style %}
  </head>
<body class="hljs">
  <div id="slides">
    {%_ for (const [index, slide] of it.slides.entries()) { %}
      <div class="slide" id="slide_{{~ index %}}">
        {%_ if (index !== 0 && it.layout === 'vertical') { %}
          <hr>
        {%_ } %}
      </div>
    {%~ slide %}
  </div>
  {%_ } %}
</body>
</html>
```

CSS (Horizontal layout)

```
/*! CodeSlide slides.horizontal.css */
html, body {
  margin: 0;
  -webkit-print-color-adjust: exact;
  print-color-adjust: exact;
```

```
    overflow: hidden;
    overscroll-behavior: none;
    scrollbar-width: none;
}
a{
    color: dodgerblue;
}
body::-webkit-scrollbar {
    display: none;
}
li {
    margin-top: 0.3em;
}
p:empty {
    display: none;
}
pre {
    white-space: pre-wrap;
    word-wrap: break-word;
}
.slide {
    min-width: calc(100vw - 4em);
    height: calc(100vh - 2em);
    overflow-y: scroll;
    scroll-snap-align: start;
    scroll-snap-stop: always;
    scrollbar-width: none;
    padding: 1em 2em;
}
#slides {
    display: flex;
    flex-direction: row;
    position: absolute; /* fix height on mobile */
    width: 100vw;
    height: 100vh;
    overflow-x: scroll;
    scroll-behavior: smooth;
    scroll-snap-type: x mandatory;
}
@media only screen and (max-width: 768px) {
    .slide {
        height: calc(100dvh - 2em);
    }
}
.slide::-webkit-scrollbar {
    display: none;
}
@page {
    margin: 0;
    size: auto;
}
@media print {
```



```
#slides {  
  width: auto;  
  height: auto;  
}  
}
```

CSS (Vertical layout)

```
/*! CodeSlide slides.vertical.css */  
html, body {  
  margin: 0;  
  -webkit-print-color-adjust: exact;  
  print-color-adjust: exact;  
  overflow: hidden;  
  overscroll-behavior: none;  
}  
a {  
  color: dodgerblue;  
}  
li {  
  margin-top: 0.3em;  
}  
p:empty {  
  display: none;  
}  
pre {  
  white-space: pre-wrap;  
  word-wrap: break-word;  
}  
.slide {  
  padding: 1em 2em;  
}  
#slides {  
  display: flex;  
  flex-direction: column;  
  position: absolute; /* fix height on mobile */  
  width: 100vw;  
  height: 100vh;  
  overflow-y: scroll;  
  scroll-behavior: smooth;  
}  
@page {  
  margin: 0;  
  size: auto;  
}  
@media print {  
  #slides {  
    width: auto;  
    height: auto;
```

```
}  
}
```

Reference the HTML template and CSS as text

```
declare module '*.css' {  
  const _: string;  
  export default _;  
}  
  
declare module '*.html' {  
  const _: string;  
  export default _;  
}  
  
import GithubDarkDimmed from './github-dark-dimmed.css';  
import HorizontalStylesheet from './slides.horizontal.css';  
import VerticalStylesheet from './slides.vertical.css';  
import Template from './slides.html';  
  
const Stylesheets = {  
  horizontal: HorizontalStylesheet,  
  vertical: VerticalStylesheet,  
  github: GithubDarkDimmed,  
};  
  
export { Stylesheets, Template };
```

Print the slideshow to the output

The print process is implemented in an application ...

Applications of CodeSlide

- 1. CodeSlide CLI

CodeSlide CLI

Usage demo

```
<yilan time=11:11:24 dir="cli/examples/rustlings" /> █
```

See also [Example usages](#)

Installation

1. Prepare Node.js runtime and NPM package manager
2. Run `npm install -g codeslide-cli` on the command line

Features

- It is an application of [CodeSlide](#)
- It allows you to easily make awesome slideshows for code snippets on command lines
- It is a Node.js Command Line Interface (CLI)

Documents

- See [Reference](#) for more information

Creator

- [AsherJingkongChen](#)

CLI entryptpoint

```

import { program } from 'commander';
import { readFile } from 'fs';
import { stdin, stdout } from 'process';
import { version, homepage, name } from '../package.json';
import { CLIOptions } from './CLIOptions';
import { print } from './print';

```

```

program
  .name(name)
  .description(`\
Example: ${name} -m ./manifest.md -o ./output.html

```

Make a slideshow (HTML/PDF file) for code snippets with a manifest (Markdown file).

Go to home page for more information: [\\${homepage}](#)

```

  .version(version, '-v, --version', `
Check the version number.`

```

```

  )
  .helpOption('-h, --help', `
Check all options and their description.`

```

```

  )
  .option('-o, --output [local_path]', `
The "output file path" of slideshow.
By default it writes the output to stdout.`

```

```

  )
  .option('-m, --manifest [local_path]', `
The "manifest file path" of slideshow.
By default it reads manifest from stdin.`

```

```

  )
  .action(async (options: CLIOptions) => {
    let { output, manifest } = CLIOptions.parse(options);
    if (manifest) {
      readFile(manifest, (err, data) => {
        if (err) { throw err; }
        print(output ?? stdout.fd, data.toString('utf8'));
      });
    } else {
      let data = Buffer.alloc(0);
      stdin
        .on('data', (d) => {
          data = Buffer.concat([data, d]);
        })
        .once('end', () => {
          print(output ?? stdout.fd, data.toString('utf8'));
        });
    }
  })
  .parseAsync();

```

CLI options validation

1. Manifest path

2. Output path

```
import { z } from 'zod';

export type CLIOptions = z.infer<typeof CLIOptions>;

export const CLIOptions = z.object({
  manifest: z.string().optional(),
  output: z.string().optional(),
})
.strict();
```

Build a Renderer and Print to the output

```
import { PathOrFileDescriptor, writeFile } from 'fs';
import { launch } from 'puppeteer';
import { Renderer } from '../../../src';

export const print = async (
  output: PathOrFileDescriptor,
  manifest: string,
): Promise<void> => {
  const renderer = await Renderer.parse(manifest);
  if (renderer.format === 'html') {
    writeFile(output, Renderer.render(renderer), 'utf8', (err) => {
      if (err) { throw err; }
    });
  } else if (renderer.format === 'pdf') {
    const browser = await launch();
    const page = await browser.newPage();
    await page.setContent(Renderer.render(renderer));
    const result = await page.pdf({
      printBackground: true,
      format: renderer.pageSize,
    });
    const closeBrowser = browser.close();
    writeFile(output, result, 'base64', (err) => {
      if (err) { throw err; }
    });
    await closeBrowser;
  }
};
```

Thanks for your watching!

See other CodeSlide CLI examples [here](#)

The installation guide [here](#)