Behind the Scenes of CodeSlide

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

CodeSlide

npm:codeslide-cli v0.12.6

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 general process

1. Build a `Renderer`
2. Render HTML and CSS
3. Print the slideshow to the output

Build a `Renderer`

`Renderer`

- `Renderer.parse`: Parse the manifest into renderer
- `Renderer.render`: Render the slideshow

```
import { Stylesheets, Template } from './slides';
import { render as renderEta } from 'eta';
import { z } from 'zod';
import { ManifestParser } from './parsers';

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

export namespace Renderer {
  export const parse = (
    manifest: string
  ): Promise<Renderer> => (
    ManifestParser.parseAsync(manifest)
  );

  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: 85%;
}`,
`#slides {
  font-family: system-ui;
  font-size: ${renderer.fontSize};
  font-weight: ${renderer.fontWeight};
  line-height: 1.5;
}`,
].join('\n')}
</style>`,
  }, { autoTrim: false, tags: ['{%', '%}'] });
}
```

Build a Renderer

Parsers

`Renderer` = `ManifestParser` (Parsed)

`ManifestParser` = `FrontMatterParser` + `SlideShowParser`

Build a Renderer

`ManifestParser`

```
import matter from 'gray-matter';
import { z } from 'zod';
import { FrontMatterParser } from './FrontMatterParser';
import { _getContent } from './_getContent';
import { SlideShowParser } from './SlideShowParser';

export const ManifestParser = z.string().transform(
  async (manifest: string) => {
    manifest = manifest.replace(
      /^[\u200B\u200C\u200D\u200E\u200F\uFEFF]/, ''
    );
    const { content, data } = matter(manifest);
    if (data.codeslide === undefined) {
      throw new Error(
        'Cannot find the key "codeslide" in the Front Matter section'
      );
    }
    const codeslide = FrontMatterParser.parse(data.codeslide);
    const slides = await SlideShowParser.parseAsync(content);
    codeslide.styles = await Promise.all(
      codeslide.styles.map((path) => _getContent(path))
    );
    return { slides, ...codeslide };
  }
);
```

Build a Renderer

FrontMatterParser

```
import { z } from 'zod';
import { isFormat } from '../Format';
import { isLayout } from '../Layout';
import { isPageSize } from '../PageSize';
import { version } from '../../package.json';

export const FrontMatterParser = 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'),
  styles: z.array(z.string()).default([]),
  version: z.string().default(version),
})
.strict()
.transform((fm) => {
  if (
    fm.layout === 'horizontal' &&
    fm.format === 'pdf'
  ) {
    fm.layout = 'vertical';
  }
  return fm;
});
```

Build a Renderer

SlideShowParser

```
import hljs from 'highlight.js';
import { marked } from 'marked';
import { z } from 'zod';
import { _getContent } from './_getContent';

export const SlideShowParser = z.string().transform((markdown) => (
  _parseSlideShow(markdown).then((html) => (
    html.split('<hr>').map((s) => s.trim())
  ))
));

const _parseSlideShow = (
  markdown: string
): Promise<string> => marked.parse(markdown, {
  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) => _parseSlideShow(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 = true;
  return token;
};
```

Build a `Renderer`

The utility to acquire resources

```
import { isNode } from 'browser-or-node';
import { fetch } from 'cross-fetch';
import { readFileSync } from 'fs';
import { pathToFileURL } from 'url';

export const _getContent = async (
  path: string | URL,
): Promise<string> => {
  if (isNode) {
    if (typeof path === 'string') {
      try {
        path = new URL(path);
      } catch (err) {
        path = pathToFileURL(path.toString());
      }
    }
    if (path.protocol === 'file:') {
      return readFileSync(path, 'utf8');
    } else {
      return fetch(path).then(async (r) => {
        if (r.ok) { return r.text(); }
        throw new Error(await r.text());
      });
    }
  }
  throw new Error(
    '_getContent is not implemented yet for the current platform'
  );
};
```

Build a `Renderer`

Options

- Export a slideshow as a HTML or PDF file

```
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 the page size in 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;
```

Render HTML and CSS

HTML template

CodeSlide depends on Eta to render HTML template.

{% and %} are interpolation characters.

```html
<!DOCTYPE HTML>
<html class="hljs">
<head>
<meta charset="utf-8">
<meta
  name="description"
  content="CodeSlide makes a slideshow for code snippets">
<meta
  name="viewport"
  content="width=device-width, initial-scale=1">
{%~ 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>
  {%_ } %}
{%~ slide %}
</div>
{%_ } %}
</div>
</body>
</html>
```

Render HTML and CSS

CSS (Horizontal layout)

```css
/*! CodeSlide slides.horizontal.css */
html, body {
  margin: 0;
  -webkit-print-color-adjust: exact;
  print-color-adjust: exact;
  overflow: hidden;
  overscroll-behavior: none;
}
a {
  color: dodgerblue;
}
```

```css
li {
  margin-top: 0.25em;
}
p:empty {
  display: none;
}
pre {
  white-space: pre-wrap;
  overflow-wrap: break-word;
}
pre > code {
  display: block;
  padding: 1em;
}
.slide {
  min-width: calc(100vw - 4em);
  height: calc(100vh - 2em);
  padding: 1em 2em;
  overflow-y: scroll;
  scroll-snap-align: start;
  scroll-snap-stop: always;
  scrollbar-width: none;
}
.slide::-webkit-scrollbar {
  display: none;
}
@media only screen and (max-width: 768px) {
  .slide {
    height: calc(90% - 2em);
    height: calc(100svh - 2em);
  }
}
#slides {
  display: flex;
  flex-direction: row;
  width: 100vw;
  height: 100vh;
  overflow-x: scroll;
  scroll-behavior: smooth;
  scroll-snap-type: x mandatory;
}
@media print {
  @page {
    margin: 0;
    size: auto;
  }
  #slides {
    width: auto;
    height: auto;
  }
}
```

Render HTML and CSS

CSS (Vertical layout)

```css
/*! 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: 0.25em;
}
p:empty {
  display: none;
}
pre {
  white-space: pre-wrap;
  overflow-wrap: break-word;
}
pre > code {
  display: block;
  padding: 1em;
}
.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;
}
@media print {
  @page {
    margin: 0;
    size: auto;
  }
  #slides {
    width: auto;
    height: auto;
  }
}
```

Render HTML and CSS

Referenced as text modules

```
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 by application ...

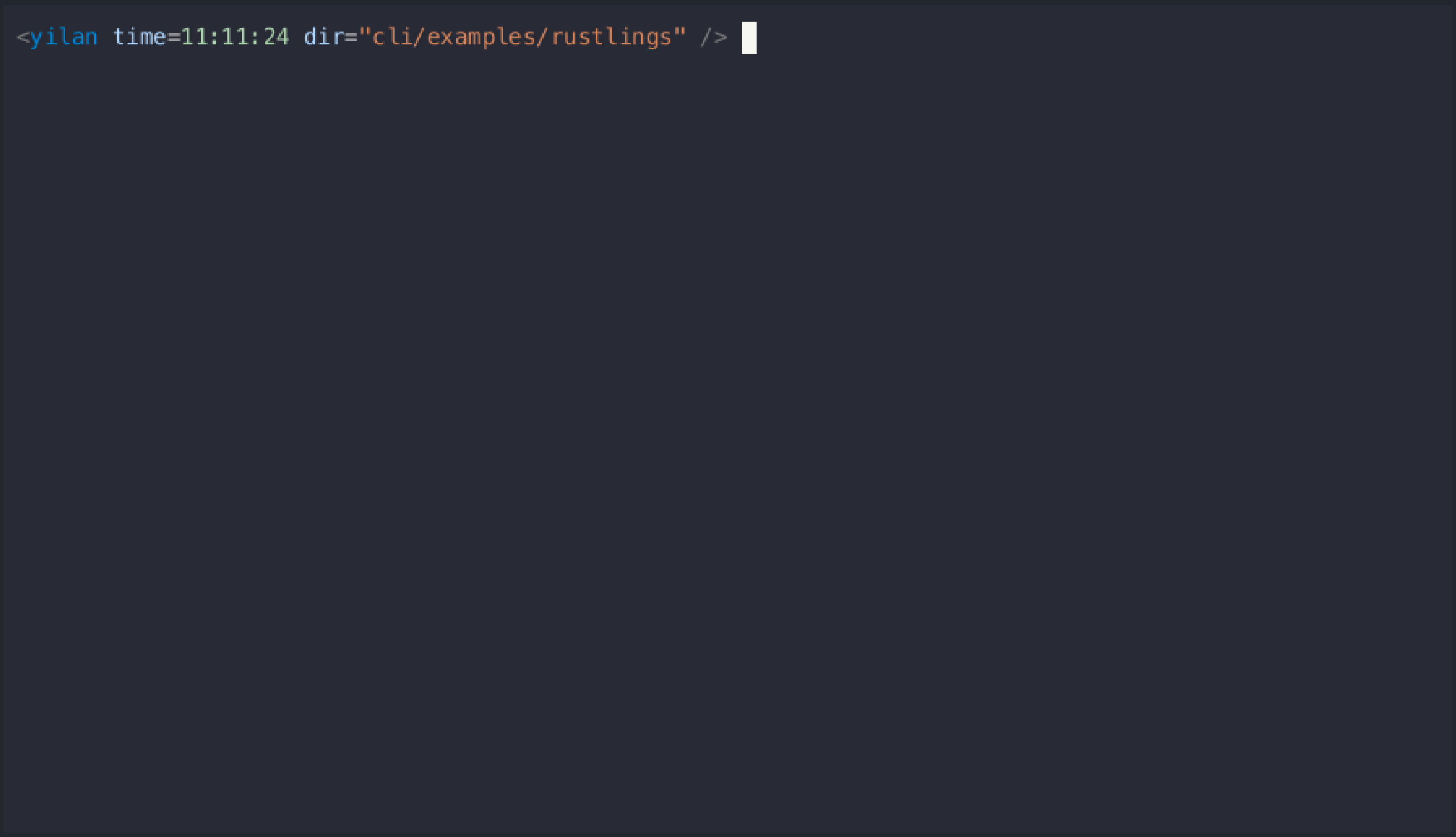
Applications of CodeSlide

1. CodeSlide CLI

CodeSlide CLI

npm v0.12.6

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 entry point

```
import { program } from 'commander';
import { readFileSync } 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) {
      print(output ?? stdout.fd, readFileSync(manifest, '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()
  .catch((err) => { throw err; });
```

CLI options validation

1. `-m, --manifest`: Manifest file path
2. `-o, --output`: Output file 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

Make use of `Renderer.parse` and `Renderer.render`

```
import { PathOrFileDescriptor, writeFileSync } 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') {
    writeFileSync(output, Renderer.render(renderer), 'utf8');
  } 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();
    writeFileSync(output, result, 'base64');
    await closeBrowser;
  }
};
```

Thanks for your watching!

See other CodeSlide CLI examples here

See the installation guide of CodeSlide CLI here