The Introduction of CodeSlide

CodeSlide

npm:codeslide-cli v0.11.1

Features

- CodeSlide makes a slideshow for code snippets
- Its applications:
  - CodeSlide CLI

Dependencies

- It uses esbuild as module bundler
- It uses Commander.js as CLI framework
- It uses Eta as HTML template engine
- It uses Highlight.js as syntax highlighter
- It uses Node Fetch as resource fetcher
- 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** form
2. Render the HTML template and CSS with the built renderer
3. Print thet slideshow to the output
```

Build a Renderer form

```
export * from './format';
export * from './layout';
export * from './pagesize';
export * from './renderer';
```

Renderer

```
import { z } from 'zod';
import { isFormat } from './format';
import { isLayout } from './layout';
import { isPagesize } from './pagesize';
import { render as renderEta } from 'eta';
import { Stylesheets, Template } from './slides';

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

export namespace Renderer {
  export const parse = (
    raw: object
  ): Renderer => _Renderer.parse(raw);

  export const render = (
    renderer: Renderer
  ): string => renderEta(
    Template,
    {
      layout: renderer.layout,
      slides: renderer.slides,
      style: `\
<style>
${
  [
    Stylesheets['github'],
    Stylesheets[renderer.layout],
    ...renderer.styles,
    `.hljs, code { font-family: ${renderer.fontFamily}; }`,
    `#slides { font-size: ${renderer.fontSize}; }`,
    `#slides { font-weight: ${renderer.fontWeight}; }`,
  ].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;
});
```

Build a Renderer form

Enumerations of fields

- 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="_{%~ index %}">
    {%_ if (index !== 0 && it.layout === 'vertical') { %}
      <hr>
    {%_ } %}
      {%~ slide %}
    </div>
  {%_ } %}
    </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;
}
body::-webkit-scrollbar {
  display: none;
}
pre {
  margin: 0;
  white-space: pre-wrap;
  word-break: break-word;
}
p:empty {
  display: none;
}
#slides {
  display: flex;
  flex-direction: row;
  position: fixed; /* fix height on mobile */
  width: 100vw;
  height: 100vh;
  overflow-x: scroll;
  scroll-behavior: smooth;
  scroll-snap-type: x mandatory;
}
.slide {
  display: flex;
  flex-direction: column;
  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;
}
@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;
}
pre {
  margin: 0;
  white-space: pre-wrap;
  word-break: break-word;
}
p:empty {
  display: none;
}
#slides {
  display: flex;
  flex-direction: column;
  position: fixed; /* fix height on mobile */
  width: 100vw;
  height: 100vh;
  overflow-y: scroll;
  scroll-behavior: smooth;
}
.slide {
  padding: 1em 2em;
}
@page {
  margin: 0;
  size: auto;
}
@media print {
  #slides {
    width: auto;
    height: auto;
  }
}
```

The HTML template and CSS are imported as text in the program

```
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

Applications

The print process runs in an application, the list of all applications is here:

1. CodeSlide CLI

CodeSlide CLI

A Node.js Command Line Interface

```
import { program } from 'commander';
import { readFileSync } from 'fs';
import { stdin, stdout } from 'process';
import { version, homepage, name } from '../package.json';
import { CLIOptions } from './options';
import { parse } from './parse';
import { render } 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) {
      manifest = readFileSync(manifest, 'utf8');
      await render(output ?? stdout.fd, await parse(manifest));
    } else {
      let data = Buffer.alloc(0);
      stdin
        .on('data', (d) => {
          data = Buffer.concat([data, d]);
        })
        .once('end', async () => {
          manifest = data.toString('utf8');
          await render(output ?? stdout.fd, await
parse(manifest));
        });
    }
  })
  .parseAsync();
```

Validate CLI options

```
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 form and render with it

```
import hljs from 'highlight.js/lib/core';
import matter from 'gray-matter';
import { marked } from 'marked';
import fetch from 'node-fetch';
import { Renderer } from '../../../src';
import { readFileSync } from 'fs';
import { pathToFileURL } from 'url';

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 Font 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))
  );

  // Is raw stylesheet needed?
  // const stylesheet: string | undefined =
  //   data.codeslide.stylesheet;
  // if (stylesheet) {
  //   renderer.styles.push(stylesheet);
  // }

  return renderer;
};

const _parse = async (manifest: string) => (
  marked.parse(manifest, {
    async: true,
    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 class="${
  code.language ? `language-${code.language} ` : ''
}hljs">${
  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 _parseURL = (path: string): URL => {
  try { return new URL(path); }
  catch (_) { return pathToFileURL(path); }
};

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());
    });
  }
};

import armasm from 'highlight.js/lib/languages/armasm';
import c from 'highlight.js/lib/languages/c';
import clojure from 'highlight.js/lib/languages/clojure';
import cmake from 'highlight.js/lib/languages/cmake';
import coffeescript from
'highlight.js/lib/languages/coffeescript';
import cpp from 'highlight.js/lib/languages/cpp';
import csharp from 'highlight.js/lib/languages/csharp';
import css from 'highlight.js/lib/languages/css';
import dart from 'highlight.js/lib/languages/dart';
import diff from 'highlight.js/lib/languages/diff';
import elixir from 'highlight.js/lib/languages/elixir';
import erlang from 'highlight.js/lib/languages/erlang';
import go from 'highlight.js/lib/languages/go';
import graphql from 'highlight.js/lib/languages/graphql';
import groovy from 'highlight.js/lib/languages/groovy';
import haskell from 'highlight.js/lib/languages/haskell';
import ini from 'highlight.js/lib/languages/ini';
import java from 'highlight.js/lib/languages/java';
import javascript from 'highlight.js/lib/languages/javascript';
import json from 'highlight.js/lib/languages/json';
import julia from 'highlight.js/lib/languages/julia';
import kotlin from 'highlight.js/lib/languages/kotlin';
import less from 'highlight.js/lib/languages/less';
import lisp from 'highlight.js/lib/languages/lisp';
import lua from 'highlight.js/lib/languages/lua';
import makefile from 'highlight.js/lib/languages/makefile';
import markdown from 'highlight.js/lib/languages/markdown';
import objectivec from 'highlight.js/lib/languages/objectivec';
import perl from 'highlight.js/lib/languages/perl';
import php from 'highlight.js/lib/languages/php';
import plaintext from 'highlight.js/lib/languages/plaintext';
import python from 'highlight.js/lib/languages/python';
import r from 'highlight.js/lib/languages/r';
import ruby from 'highlight.js/lib/languages/ruby';
import rust from 'highlight.js/lib/languages/rust';
import scala from 'highlight.js/lib/languages/scala';
import scss from 'highlight.js/lib/languages/scss';
import shell from 'highlight.js/lib/languages/shell';
import sql from 'highlight.js/lib/languages/sql';
import swift from 'highlight.js/lib/languages/swift';
import typescript from 'highlight.js/lib/languages/typescript';
import vbnet from 'highlight.js/lib/languages/vbnet';
import xml from 'highlight.js/lib/languages/xml';
import yaml from 'highlight.js/lib/languages/yaml';

hljs.registerLanguage('armasm', armasm);
hljs.registerLanguage('c', c);
```

```
  hljs.registerLanguage('clojure', clojure);
  hljs.registerLanguage('cmake', cmake);
  hljs.registerLanguage('coffeescript', coffeescript);
  hljs.registerLanguage('cpp', cpp);
  hljs.registerLanguage('csharp', csharp);
  hljs.registerLanguage('css', css);
  hljs.registerLanguage('dart', dart);
  hljs.registerLanguage('diff', diff);
  hljs.registerLanguage('elixir', elixir);
  hljs.registerLanguage('erlang', erlang);
  hljs.registerLanguage('go', go);
  hljs.registerLanguage('graphql', graphql);
  hljs.registerLanguage('groovy', groovy);
  hljs.registerLanguage('haskell', haskell);
  hljs.registerLanguage('ini', ini);
  hljs.registerLanguage('java', java);
  hljs.registerLanguage('javascript', javascript);
  hljs.registerLanguage('json', json);
  hljs.registerLanguage('julia', julia);
  hljs.registerLanguage('kotlin', kotlin);
  hljs.registerLanguage('less', less);
  hljs.registerLanguage('lisp', lisp);
  hljs.registerLanguage('lua', lua);
  hljs.registerLanguage('makefile', makefile);
  hljs.registerLanguage('markdown', markdown);
  hljs.registerLanguage('objectivec', objectivec);
  hljs.registerLanguage('perl', perl);
  hljs.registerLanguage('php', php);
  hljs.registerLanguage('plaintext', plaintext);
  hljs.registerLanguage('python', python);
  hljs.registerLanguage('r', r);
  hljs.registerLanguage('ruby', ruby);
  hljs.registerLanguage('rust', rust);
  hljs.registerLanguage('scala', scala);
  hljs.registerLanguage('scss', scss);
  hljs.registerLanguage('shell', shell);
  hljs.registerLanguage('sql', sql);
  hljs.registerLanguage('swift', swift);
  hljs.registerLanguage('typescript', typescript);
  hljs.registerLanguage('vbnet', vbnet);
  hljs.registerLanguage('xml', xml);
  hljs.registerLanguage('yaml', yaml);
```

Print the slideshow to the output

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

export const render = async (
  output: PathOrFileDescriptor,
  renderer: Renderer,
): Promise<void> => {
  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