The Introduction of CodeSlide

README

```
# CodeSlide
[![](https://img.shields.io/npm/v/codeslide-cli?
color=%230647D4&label=npm%3Acodeslide-cli&style=flat-square)]
(https://www.npmjs.com/package/codeslide-cli?activeTab=readme)

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

## Dependencies
- It uses [esbuild](https://github.com/evanw/esbuild) as
module bundler
- It uses [Commander.js](https://github.com/tj/commander.js)
as CLI framework
- It uses [Eta](https://github.com/eta-dev/eta) as HTML
template engine
- It uses [Highlight.js]
(https://github.com/highlightjs/highlight.js) as syntax
highlighter
- It uses [Node Fetch](https://github.com/node-fetch/node-
fetch) as resource fetcher
- It uses [Puppeteer](https://github.com/puppeteer/puppeteer)
as PDF printer
- It uses [Zod](https://github.com/colinhacks/zod) as JSON
schema validator

## Documents
- See [**Reference**](./docs/REFERENCE.md) for more
information

## Creator
- [AsherJingkongChen](https://github.com/AsherJingkongChen)
```

The essentials

Render the HTML template and CSS to a slideshow

```
404: Not Found
```

```
import { z } from 'zod';
import { isFormat } from './format';
import { isLang } from './lang';
import { isLayout } from './layout';
import { isPagesize } from './pagesize';

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

export const Printer = 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.object({
          code: z
            .string()
            .default(''),
          lang: z
            .string()
            .refine(isLang)
            .optional(),
          title: z
            .string()
            .default(''),
        })
        .strict()
      )
      .default([]),
    styles: z
      .array(z.string())
      .default([])
      .transform((arg) => [
        'https://cdnjs.cloudflare.com/ajax/libs/highlight.js/'
+
        '11.8.0/styles/github-dark-dimmed.min.css',
        ...arg,
      ]),
  })
  .transform((arg) => {
    if (
      arg.layout === 'horizontal' &&
      arg.format === 'pdf'
    ) {
      arg.layout = 'vertical';
    }
    return arg;
  });
```

```
import { render as renderEta } from 'eta';
import { Stylesheet, Template } from './app';
import { Printer } from './printer';

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

export * from './format';
export * from './lang';
export * from './layout';
export * from './pagesize';
export * from './printer';
```

The HTML template

```
404: Not Found
```

The CSS

```
404: Not Found
```

Let's see some applications!

CodeSlide CLI: A Node.js Command Line Interface

```
import { program } from 'commander';
import { version, homepage, name } from '../package.json';
import { run } from './run';

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

Make a HTML or PDF slideshow for code snippets
with a JSON configuration.

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 file path of "slideshow output". ' +
    'If not set, it writes the output to stdout.'
  )
  .option('--font-family [string]',
```

```
    'CSS Property'
  )
  .option('--font-size [string]',
    'CSS Property'
  )
  .option('--font-weight [string]',
    'CSS Property'
  )
  .option('--format [enum]',
    'enum = html | pdf'
  )
  .option('--layout [enum]',
    'enum = horizontal | vertical\n' +
    'Note: Cannot set --layout=horizontal when --format=pdf '
  )
  .option('--pagesize [enum]',
    'enum = letter | legal | tabloid | ledger | a0 | a1 | a2 |
a3 | a4 | a5 | a6\n' +
    'Is only needed when --format=pdf'
  )
  .option('--slides [slide...]',
    'slide = title path'
  )
  .option('--styles [path...]',
    ''
  )
  .action(run)
  .parseAsync();
```

CLI options validator

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

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

export const CLIOptions = z.object(
  {
    output: z.string().optional(),
    fontFamily: z.string().optional(),
```

```
    fontSize: z.string().optional(),
    fontWeight: z.string().optional(),
    format: z.string().optional(),
    layout: z.string().optional(),
    pagesize: z.string().optional(),
    slides: z.array(z.string()).optional(),
    styles: z.array(z.string()).optional(),
  })
  .strict()
  .superRefine((ref, ctx) => {
    if ((ref.slides?.length ?? 0) % 2 !== 0) {
      ctx.addIssue({
        code: z.ZodIssueCode.custom,
        message: 'The option --slides should has even number
of arguments',
      });
    }
  });
```

Parse CLI options -> Print to output

```
import { Printer } from '../../../src';
import { CLIOptions } from './options';
import { mayfail } from './tool';

export const parse = (
  options: CLIOptions,
): Printer => {
  options = mayfail(() => CLIOptions.parse(options));

  let slides: Printer['slides'] = [];
  options.slides?.forEach((arg, index) => {
    if (index % 2 === 0) {
      slides.push({ title: arg, code: '' });
    } else {
      slides[slides.length - 1].code = arg;
    }
  });
```

```
  return mayfail(() => Printer.parse({
    ...options,
    slides,
  }));
};



import { PathOrFileDescriptor, writeFileSync } from 'fs';
import { launch } from 'puppeteer';
import { render, Printer } from '../../../src';
import { mayfailAsync } from './tool';

export const print = async (
  output: PathOrFileDescriptor,
  printer: Printer,
): Promise<void> => mayfailAsync(async () => {
  switch (printer.format) {
    case 'html':
      writeFileSync(output, render(printer), 'utf8');
      break;
    case 'pdf':
      const browser = await mayfailAsync(launch());
      const page = await mayfailAsync(browser.newPage());
      await mayfailAsync(page.setContent(render(printer)));
      const result = await mayfailAsync(
        page.pdf({
          printBackground: true,
          format: printer.pagesize, // is it redundant?
        }
      ));
      const closeBrowser = mayfailAsync(browser.close());
      writeFileSync(output, result, 'base64');
      await closeBrowser;
      break;
    default: throw new Error('Undefined Printer.format');
  }
});
```

```ts
import { stdout } from 'process';
import { guessLangFromURL } from '../../../src';
import { CLIOptions } from './options';
import { parse } from './parse';
import { print } from './print';
import { getContent, parseURL } from './tool';

export const run = async (
  options: CLIOptions,
): Promise<void> => {
  const printer = parse(options);

  printer.slides = await Promise.all(
    printer.slides.map(async (slide) => {
      if (slide.code) {
        const codeURL = parseURL(slide.code);
        return {
          code: await getContent(codeURL),
          lang: guessLangFromURL(codeURL),
          title: slide.title,
        };
      }
      return slide;
    })
  );

  printer.styles = await Promise.all(
    printer.styles.map((path) => getContent(path))
  );

  return print(options.output ?? stdout.fd, printer);

  // // Not paralleled
  // for (const slide of printer.slides) {
  //   if (slide.code) {
  //     const codeURL = parseURL(slide.code);
  //     slide.code = await getContent(codeURL);
  //     slide.lang = guessLangFromURL(codeURL);
  //   }
  // }
```

```
  // for (const [index, path] of printer.styles.entries()) {
  //   printer.styles[index] = await getContent(path);
  // }
};
```

The End