Interactive Code Playgrounds - Bundle

Interactive Code Playgrounds Bundle is a plugin for embedding interactive code playgrounds in HTML pages.

The editor used in these playgrounds is CodeMirror6, an in-browser editor distributed as a collection of modules.

The project is based on Svelte, a tool for building web applications. The actual build step is performed through Vite and the gulp tool.

The most computationally expensive tasks the bundle does are performed using Web Workers and SharedWorkers.

Examples

Some examples of this plugin can be found in the repository icp-slides. In particular, a presentation of the main elements this plugin exposes is hosted as a Github Pages website in https://lucademenego99.github.io/icp-slides/editors.html

Building from source

Install all the packages:

```
npm install
```

Run a demo:

```
npm run dev
```

Note: In Windows, the following codes must be executed in Bash and not Powershell (e.g in a GIT Bash)

Build the entire module:

```
npm run build-all
```

Build for a specific language;

```
npm run build -- --language python
```

Note: before building for c/c++/python/java, you need to prepare the webworker:

```
npm run prepare-[cpp/python/java]
```

Each language built is zipped inside the `dist/base/export-languages.zip` file, which can be copied and distribted for use in the mobile app.

The build-all command will generate a `dist/base` folder with the following files:

- `*.iife.js`: script to be included inside the HTML page using ICP; to reduce the size of the bundle to be downloaded, please use the file corresponding to the language you want to use;
- `redbean.com`: a redbean web-server ready to be customized to serve ICP slides. redbean is a simple HTTP server that exposes the current folder. It can be executed on Linux, MacOS, Windows, FreeBSD, OpenBSD and NetBSD thanks to the Cosmopolitan project. Please see icp-create-server for more information on how to use it;
- `example.html`: an example showcasing the main elements exported by the library;
- various css files: styles to make the slides work.

Command structure

The command used to build support for languages follows this structure:

```
npm run build -- --language lang [--incremental] [--mobile y/n]
```

- language: the language to be built
- incremental: if the new language should overwrite the ones already built or not
- mobile: if the .zip file to export langauge support to mobile devices should be built

Redbean file

The resulting Redbean file contains the dependencies to compile and run all programming languages supported by ICPs, so it is quite heavy.
If you need to prepare a redbean file containing only the dependencies for a specific programming language, you can:

- take the `redbean.com` file contained in the `public` directory;
- use the `zip` command line program available in Linux to insert the dependencies you need and the HTML slides;
- distribute it!

As an example, here are the commands to prepare a Redbean file compatible with Python:

- `zip redbean.com custom-style.css reveal.js python-offline.iife.js reveal.css white.css # Common dependencies`
- `zip redbean.com -r utils # Utils folder available in the public directory, containing only the "python" subfolder`
- `zip redbean.com index.html # Your HTML slides`

Exported Web Components

The build phase creates the following web components:

- `<javascript-editor></javascript-editor>`;
- `<typescript-editor></typescript-editor>`;
- `<python-editor></python-editor>`;
- `<java-editor></java-editor>`;
- `<cpp-editor></cpp-editor>`;
- `<sql-editor></sql-editor>`;
- `<p5-editor></p5-editor>`;
- `<processing-editor></processing-editor>`;
- `<standardml-editor></standardml-editor>`.

Every component has a set of properties that can be passed inside the tags:

- `code`: the initial code snippet the playground should contain;
- `theme`: *light* or *dark* (but if the user has selected a default theme, the one chosen by the user will be used);
- `type`: *normal* or *vertical*;
- `downloadable`: if set to true, a download button will appear allowing to download the code snippet;
- `id` and `save`: if save is set to true, the code will be saved in local storage with key=id.

To access these components you need to import a bundle in your HTML page. Which bundle to add depends on the language you want to use:

- all the languages: `base/full.iife.js`;
- specific language: `base/{language}.iife.js`.

Since the package is published on npm, you can access it using the CDN you prefer. However, since CDNs like `unpkg.com` are quite slow with not famous scripts, the latest version of the package is also exposed in Github Pages. So you can include it with:

```
<script src="https://lucademenego99.github.io/icp-
bundle/base/{language}.iife.js"></script>
```

The available languages that can be used in the editors are the following:

	Syntax Highlighting	Auto Completion	Lint Checks	Run Code
javascript	☑	☑		☑
typescript	☑	☑	☑	☑
python	☑	☑		☑
java	☑			☑

	Syntax Highlighting	Auto Completion	Lint Checks	Run Code
c/c++	☑			☑
sql	☑	☑		☑
p5	☑			☑
processing	☑			☑
standard-ml	☑			☑

Read-only components with predefined editable parts

Sometimes you may want to embed editor components in which only some parts of them are editable. In order to do it, you can use the specially crafted tokens `<EDITABLE>` and `</EDITABLE>`. For example:

```
<python-editor code="print('<EDITABLE>Hello World!</EDITABLE>')" />
```

Mentions

- the typescript plugin included in `src/modules/` has been taken from prisma/text-editors
- run java code in the browser: teavm-javac based on frankbauer's fork (but modified to work as a SharedWorker)
- run c++ code in the browser: wasm-clang and playcode
- run python code in the browser: pyodide
- run sql code in the browser: sql.js
- run standard ml code in the browser: SOSML