project-calavera Documentation

Schalk Neethling

Apr 17, 2022

Contents

CHAPTER 1

Contributing

1.1 Contributing a config

At the time of writing all configs form part of the base Calavera package. Would a pluggable architecture make sense instead? It most definitely is something to consider but, until then, the information below guides you through the process of adding a new config to Calavera.

1.2 The skeletons

The first step is adding the relevant skeleton(s) to the closet. Generally, and for this example, you will add this to the root of `closet/skeletons`. For our current example, we will add a file called `tsconfig.json` to this folder.

1.3 The config source

The next step is to create the source file for our `config`. Inside `lib/config-helpers` create a new file called for example `typescript.js`. At some point we will want to write out our configurtion file content and so, the first thing we need to do is import `fileUtils`:

```
const { writeFiles } = require("../fileUtils");
```

Next, we set a couple of constants that hold the values for the config file(s) we will output, as well as the dependencies an end user will need to install. For example:

```
const typescriptConfigFiles = ["tsconfig.json"];
const typescriptDependencies = ["typescript", "@tsconfig/recommended"];
```

Next we add our main function:

```
const typeScriptTooling = {
    addTooling: (target) => {

    }
}
```

And lastly we export our constants and main function:

```
module.exports = {
  typescriptConfigFiles,
  typescriptDependencies,
  typeScriptTooling,
};
```

Depending on the complexity of the config you are adding, the main function here could be as simple as the following:

```
/**
 * Returns an array of tool dependencies
 * and writes the relevant config file to disk
 * @returns Array of strings of relevant dependencies
 */
() => {
    writeFiles(configFiles);
    return dependencies;
}
```

1.4 Adding config to the Catacomb

At this point our config is ready to use, but will not be avialable to end users until we add the relevant entry to `lib/fill-catacomb.js`. First, we need to import our config:

```
const { typescriptTooling } = require("./config-helpers/typescript");
```

Next add a `case` for your config, for example:

```
case "typescript":
  let typescriptDependencies = await typescriptTooling.addTooling();
  dependencies = dependencies.concat(typescriptDependencies);
  break;
```

With the above in place, you should be able to use your new config by adding an entry to `package.json`, for example:

```
"calavera": {
    "typescript": true
}
```

1.5 Writing tests

All new configs, or changes to existing configs, should have relevant tests to ensure the code will work as expected. We use Jest as our test framework and mock-fs for filesystem mocking suring tests.

Create your test file, for example, `typescript.test.js` next to the source file inside `lib/config-helpers/` First import the libraries for the filesystem as well as `mock-fs`:

```
const fs = require("fs");
const mock = require("mock-fs");
```

Next we import our constants and main function:

```
const {
  typescriptConfigFiles,
  typescriptDependencies,
  typescriptTooling,
} = require("./typescript");
```

Next we create an object that `mock-fs` will use to create a mock filesystem we can test against:

```
const defaultFiles = {
  "closet/skeletons/": {
    "tsconfig.json": "{}",
  },
};
```

Now we are ready to start writing our tests:

```
describe("typescriptTooling.addTooling", () => {
  afterEach(() => {
    mock.restore();
  });

  it("returns an array with the default dependencies as strings", async () => {
    mock(defaultFiles);

    let dependencies = await typescriptTooling.addTooling();
    expect(dependencies).toStrictEqual(typescriptDependencies);
  });
});
```

To run the above, enter the following in your terminal:

```
yarn test
```

> NOTE: The above will run the tests once and exit. While developing, you probably want to run Jest in watch mode. To do this, run the following command instead: `yarn test:dev`

The above is not an exhaustive test by any means. For more examples, have a look at existing tests inside the `lib/config-helpers` folder.

Once satisfied with your test coverage, there is one last step you need to complete.

1.6 Documentation

You will find the documentation for the project in the `docs` folder. We host out documentation on ReadTheDocs and use the Sphinx documentation generator. You can find further details on the Sphynx documentation site.

The easiest way to get a head start on writing relevant documentation for your config, is to copy and change one of the existing files inside `docs/source/configs`. Using one of these as a template, create a new file inside `docs/`

`source/configs` called for example `typescript.md`. Once your documentation is written, add a refence to it inside `docs/source/index.rst` under `toctree`

You are now ready to open your pull request. We follow the following pattern for commit messages:

```
(enhancement) short title description

More detailed body desription of the enhancement

fix #123
```

Thank you for using and contributing to Project Calavera.

CHAPTER 2

Configurations

2.1 Prettier config

From the Prettier docs:

> Prettier is an opinionated code formatter

To add `Prettier` to your project, add the following to your `package.json`

```
...
"calavera": {
    "prettier": true
}
```

2.1.1 Skeletons

This adds the following files to the root of your project:

- .prettierrc.json

Calavera uses the default Prettier config. If you need to customise the defaults, you can find relevant documentation on the Prettier website.

2.1.2 Dev dependencies

`Prettier` adds the following `devDependencies`:

- prettier

2.2 CSS config

The CSS config adds the required configurations files and configuration for Stylelint. By default it also enables the `stylelint-config-recommended`(https://github.com/stylelint/stylelint-config-recommended) and `stylelint-a11y`(https://github.com/YozhikM/stylelint-a11y) Stylelint extensions.

To add the `CSS` config to your project, add the following to `package.json`

```
...
"calavera": {
    "css": true
}
```

2.2.1 Skeletons

This adds the following files to the root of your project:

- .stylelintrc
- .stylelintignore

.stylelintrc

```
{
  "extends": [
    "stylelint-config-recommended",
    "stylelint-a11y/recommended"
  ],
  "rules": {
    "max-nesting-depth": 2,
    "declaration-no-important": true,
    "font-weight-notation": "named-where-possible"
  }
}
```

.stylelintignore

By default files inside `css/libs` will be ignored.

```
css/libs/
```

2.2.2 Dev dependencies

`CSS` config adds the following `devDependencies`:

- stylelint
- stylelint-a11y
- stylelint-config-recommended

Calavera will output the command you need to run to install the above dependencies inside your project.

2.3 SASS config

The SASS config adds everything you get with the CSS config as well as Stylelint extensions for SASS. To add the `SASS` config to your project, add the following to `package.json`

```
...
"calavera": {
    "sass": true
}
```

2.3.1 Skeletons

This adds the following files to the root of your project:

- .stylelintrc
- .stylelintignore

.stylelintrc

```
{
  "extends": [
    "stylelint-config-recommended",
    "stylelint-config-sass-guidelines",
    "stylelint-a11y/recommended"
  ],
  "plugins": [
    "stylelint-scss"
  ],
  "rules": {
    "max-nesting-depth": 2,
    "declaration-no-important": true,
    "font-weight-notation": "named-where-possible"
  }
}
```

.stylelintignore

By default files inside `css/libs` will be ignored.

```
css/libs/
```

2.3.2 Dev dependencies

`SASS` config adds the following `devDependencies`:

- stylelint
- stylelint-a11y
- stylelint-config-recommended
- stylelint-scss

- stylelint-config-sass-guidelines
- sass
- node-sass-chokidar

Calavera will output the command you need to run to install the above dependencies inside your project.

> NOTE: The above also includes the SASS preprocessor itself as well as `node-sass-chokidar`. Quoting from its docs: "Why? Because Gaze in docker and various virtual machines uses a lot of resources whereas chokidar does not. Read about the advantages of chokidar"

2.4 eslint config

The eslint config adds the required configurations files and configuration for eslint. By default it also enables the `eslint:recommended`(https://eslint.org/docs/rules/) and `plugin:import/errors`(https://github.com/benmosher/eslint-plugin-import#rules) eslint extensions.

To add the `eslint` config to your project, add the following to `package.json`

```
...
"calavera": {
    "eslint": true
}
```

2.4.1 Skeletons

This adds the following files to the root of your project:

- .eslintrc.json

.eslintrc.json

```
{
    "extends": [
        "eslint:recommended",
        "plugin:import/errors"
    ],
    "rules": {
        "no-console": 1
    },
    "plugins": [
        "import"
    ],
    "parserOptions": {
        "ecmaVersion": 2018
    },
    "env": {
        "es6": true,
        "browser": true,
        "node": true
    }
}
```

2.4.2 Dev dependencies

`eslint` config adds the following `devDependencies`:

- eslint
- eslint-plugin-import

Calavera will output the command you need to run to install the above dependencies inside your project.

2.5 jest config

The jest config does not add any configuration files but, updates your `eslint` config with the relevant Jest `eslint` plugins. If you do not already have an `eslint` configuration file, this config will add a `eslint` config as specified in the eslint config docs and update it with the Jest plugins.

To add the `jest` config to your project, add the following to `package.json`

```
...
"calavera": {
    "jest": true
}
```

2.5.1 Skeletons

If no `eslint` config exists, this adds the following files to the root of your project:

- .eslintrc.json

.eslintrc.json

```
{
    "extends": [
        "eslint:recommended",
        "plugin:import/errors",
        "plugin:jest/recommended",
        "plugin:jest/style"
    ],
    "rules": {
        "no-console": 1
    },
    "plugins": [
        "import"
    ],
    "parserOptions": {
        "ecmaVersion": 2018
    },
    "env": {
        "es6": true,
        "browser": true,
        "node": true
    }
}
```

If your project already had an `eslint` config, the following two entries are added to the `extends` array:

- plugin:import/errors
- plugin:jest/recommended

2.5.2 Dev dependencies

`jest` config adds the following `devDependencies`:

- jest
- eslint-plugin-jest

If no `eslint` config existed, it will also add the following dependecnies:

- eslint
- eslint-plugin-import

Calavera will output the command you need to run to install the above dependencies inside your project.

2.6 typescript config

The typescript config adds the required configurations files for typescript.

There are three configuration options based on defaults provided by tsconfig bases

- Default recommended
- Node 12 recommended
- Transitional recommended

To add the default `typescript` config for use in non-Nodejs based projects, add the following to `package.json`

> NOTE: All config will also set the `outDir` to build

```
...
"calavera": {
    "typescript": true
}
```

To add the Node 12 target `typescript` config , add the following to `package.json`

```
...
"calavera": {
    "typescript": "node"
}
```

To add the transitional `typescript` config target which allow mixing of TypeScript and JavaScript files, add the following to `package.json`

```
...
"calavera": {
    "typescript": "transitional"
}
```

2.6.1 Skeletons

This adds the following files to the root of your project:

- tsconfig.json

  Please see the relevant config on the bases repo for details of the configuration specifics

2.6.2 Dev dependencies

`typescript` config adds the following `devDependencies`:

- typescript

And one of either of the following dependent on your config setting:

- @tsconfig/recommended
- @tsconfig/node12

Calavera will output the command you need to run to install the above dependencies inside your project.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search