Building a utility toolkit

There are tools that you use in all your projects and you'd rather not have to manually install every time.

The project attempts to create a set of those scripts packed as an NPM module, modeled after the `kcd-scripts` `react-scripts` and `@wordpress/scripts` packages, that will make it easier to start new projects by adding the scripts package as a development dependency.

The package will contain, at a minimum, the following:

- ESLint
    - Google Configuration
- CSS Linting
- Bundler configuration
    - Webpack
    - Rollup
- Prettier configuration
    - JS
    - HTML
- Carlos's specic tools
    - WP-env
    - Markdown Linting

If you have local configurations, they will override the ones provided in the package so you may not have to change the package unless you want to change the default.

Basic syntax

The basic command looks like this:

```
cal-scripts command --flags
```

The idea is that all the parameters after the command will be passed to the command as paramters.

The Challenges

I came accross several challenges when it comes to adapting someone else's projects for your own needs.

Deciding what to add

The first challenge is to decide what to add, what to change and what to leave alone.

Because this is based on an existing project, I have to consider that making any changes will require a lot of work and time so I want to make sure that whatever defaults I change make sense for future projects. When are local defaults enough for the current project?

Adding @wordpress/env

This is an example of how to add tools to your scripts package so you can use them in all your projects without having to install them manually each time.

`@wordpress/env` uses Docker to run a development environment to test plugins and themes.

Adding the package is a three step process:

1. Install a local version of the `@wordpress/env` package as a dependency
2. Create a script that runs the `wp-env` command. I call it `wpenv`
3. Modify the `run-scripts` script to include and run the `wpenv` script

Importing the file works as normal.

```
npm i @wordpress/env
```

The `wpenv` script uses cross-spawn and a utility script to run the `wp-env` command.

```
const spawn = require('cross-spawn')
const {resolveBin} = require('../utils')
```

```
const args = process.argv.slice(2)

const result = spawn.sync(
  resolveBin('wp-env', {
    executable: 'wp-env',
  }), args, {
    stdio: 'inherit',
})

process.exit(result.status);
```

The final part is to modify the `run-scripts` script to include and run wpenv.

```
function spawnScript() {
  const args = process.argv.slice(2)
  const scriptIndex = args.findIndex(x =>
    [
      'build',
      'format',
      'lint',
      'pre-commit',
      'test',
      'validate',
      'typecheck',
      'wpenv',
    ].include'format')

// The rest of the script goes here

}
```

With these items in place, and assuming that Docker is up and running, you can run `cal-scripts wpenv start` in a plugin or theme directory to start a Dockerized development environment.

Switching ESLint to use Google's shareable

configuration

Most of my projects use Google's eslint configuration that supports the Google JavaScript style guide (ES2015+ version).

While the guide covers a lot of things I don't use (like Closure Compiler), I still think it's useful to have a configration to follow for my projects.

Switching from one ESLint configuration requires a lot of work. The first part is to install the Google ESLint configuration.

```
npm i eslint-config-google
```

The first step is to configure the ESLint rules in the `package.json` file.

I want to work with ES2017 and earlier so I set the `ecmaVersion` to 2017.

Setting the env variable to `es6` tells ESLint not to flag ES6 and later constructs that were reserved words in earlier versions of the language.

The `extends` variable tells ESLint what configuration(s) to use. **The order does matter**: ESLint will evaluate configurations in the order they are listed so the rules on the last configuration listed will override previous, and possibly conflicting rules.

The `rules` variable is where you define custom rules for the `cal-scripts` package. It is empty as I don't have anything that I want to override yet. Instead, I chose to override rules in the files where the overrides happen.

```
"eslintConfig": {
  "parserOptions": {
    "ecmaVersion": 2017
  },

  "env": {
    "es6": true
  },
  "extends": [
```

```
    "kentcdodds",
    "kentcdodds/jest",
    "google"
  ],
  "kentcdodds""rules": {}
},
"eslintIgnore": [
  "node_modules",
  "coverage",
  "dist"
],
```

The next step is to modify the ESLint configuration bundled with the scripts package and referenced from the ESLint configuration at the root of the package.

The only difference is that this time I'm also loading ESLint's React configuration if any dependencies in the host project use React.

In the future I may also want to do something similar with Vue or any other tools that I use frequently.

```
const {ifAnyDep} = require('../utils');

module.exports = {
  extends: [
    require.resolve('eslint-config-google'),
    ifAnyDep('react', require.resolve('eslint-config-react')),
  ].filter(Boolean),
  rules: {},
};
```

The rest is just linting the project file and fixing all the errors ESLint reports. Just make sure that you don't introduce any new ones in the process of fixing the existing ones.

Lint Markdown files

Working with Markdown is troublesome. While CommonMark standardized the

Markddown syntax, there hasn't been a good way to lint Markdown files to enforce conventions.

I've chosen to use a MarkdownLint CLI Tool to build into the scripts package. It is similar in concept to what the VS Code Markdown Lint extension uses so, in theory, it should be easy to work with.

Adding the tool is no different than adding `wp-env` to the package.

1. Install the NPM package locally
2. Write and link the configuration file
3. Add it to the list of scripts available to run
4. Test the script

Install the NPM package

We install the NPM package as normal.

```
npm install markdownlint-cli
```

Write a configuration file

Markdownlint uses a list of rules that are built into tool. We can override and customize the rules using a `.markdownlint.json` configuration file.

The example below ignores line length and inline HTML rules for all the documents that use the configuration file.

```
module.exports = {
  'line-length': false,
  'no-inline-html': false,
};
```

We need to do this in two places. The first is in the `src/config/` directory where we create the `markdownlint.js` file where we store the rules we want to use.

The second is at the root of the project where we create a shortcut to the configuration file.

```
module.exports = require('./src/config/markdownlint');
```

Add it to the list of scripts available

Next, we need to let the package know that the `markdownlint` script is available. We do this by adding the script name to the `spawnScript` functions in the `run-scripts` script

```
function spawnScript() {
  const args = process.argv.slice(2);
  const scriptIndex = args.findIndex((x) =>
    [
      'build',
      'format',
      'lint',
      'pre-commit',
      'test',
      'validate',
      'typecheck',
      'wpenv',
      'markdownl'format'  ].includes(x),
  );

  // Rest of the script goes here
```

Test the script

I know I should write proper tests but I'm not sure how to do that with Jest just yet.

Instead I've run the package both internally to validate and test the contents of the `cal-scripts` package and I've installed it in a standalone new project to validate that I can run the scripts. Even failure tells me that they worked.

Conclusion

I have working set of scripts, there are still tests to be written but I need to become more familiar with Jest and the process of writing tests before I can finish that part.

You're welcome to test the scripts and see if they work for you.

The package has been published in the NPM registry.

The project is also hosted on Github. If you see something missing, wrong or that you think I should add, file an issue and let's discuss it.