Contributing to Create React App's E2E tests

This is an end to end kitchensink test suite, but has multiple usages in it.

Running the test suite

Tests are automatically run by the CI tools. In order to run them locally, without having to manually install and configure everything, the `npm run e2e:docker` CLI command can be used.

This is a script that runs a **Docker** container, where the node version, git branch to clone, test suite, and whether to run it with `yarn` or `npm` can be chosen. Run `npm run e2e:docker --help` to get additional info.

If you need guidance installing **Docker**, you should follow their official docs.

Writing tests

Each time a new feature is added, it is advised to add at least one test covering it.

Features are categorized by their scope:

- *env*, all those which deal with environment variables (e.g. `NODE_PATH`)
- *syntax*, all those which showcase a single EcmaScript syntax feature that is expected to be transpiled by **Babel**
- *webpack*, all those which make use of webpack settings, loaders or plugins

Using it as Unit Tests

In it's most basic for this serve as a collection of unit tests on a single functionality.

Unit tests are written in a `src/features/**/*.test.js` file located in the same folder as the feature they test, and usually consist of a `ReactDOM.render` call.

These tests are run by **jest** and the environment is `test`, so that it resembles how a **Create React App** application is tested.

Using it as Integration Tests

This suite tests how the single features as before behave while development and in production. A local HTTP server is started, then every single feature is loaded, one by one, to be tested.

Test are written in `integration/{env|syntax|webpack}.test.js`, depending on their scope.

For every test case added there is only a little chore to do:

- a `case` statement must be added in `src/App.js`, which performs a dynamic `import()` of the feature
- add a test case in the appropriate integration test file, which calls and awaits `initDOM` with the previous `SwitchCase` string

A usual flow for the test itself is something similar to:

- add an `id` attribute in a target HTML tag in the feature itself
- since `initDOM` returns a `Document` element, the previous `id` attribute is used to target the feature's DOM and `expect` accordingly