Thanks for contributing!

## Development

### Installing dependencies

```bash

npm install

```

You will find all building blocks that make up Radium in the [`src`](src) folder.

### Testing

For ease, we've wrapped up all our individual test commands into:

```sh

$ npm run build-lib OR watch-lib # One time / watched src file build

$ npm run test # Single pass of all tests.

$ npm run test-dev # Watch test file changes and rerun tests automatically.

```

#### Frontend

You will find tests for each module inside [`src/\_\_tests\_\_`](src/\_\_tests\_\_). Whenever making any changes, ensure that all existing tests pass by running `npm run test-frontend`. You can also have [`Karma`](http://karma-runner.github.io/) running in the background and run your tests every time you make a change by doing `npm run test-dev-frontend`.

If you are adding a new feature or some extra functionality, you should also make sure to accompany those changes with appropriate tests.

#### Backend

We have a small number of tests for SSR/Node.js usage in [`test`](test). Whenever making any relevant changes, ensure that all existing tests pass by running `npm run test-node`. You will need to have a babel watch running if you are changing source files since these tests rely on built files in `lib/`, which you can do easily with `npm run watch-lib` in a separate terminal.

To get watched test files automagically updated and run, use `npm run test-node-dev`.

### Linting

Before commiting any changes, be sure to do `npm run lint`; this will lint all relevant files using [ESLint](http://eslint.org/) and report on any changes that you need to make. You can also run `npm run fixlint` to fix most common lint errors automatically.

### Flow

- [Install flow](http://flowtype.org/docs/getting-started.html#installing-flow)

- Run `flow` to check for missing annotations and static type check failures

### Examples

You can run examples locally by simply doing `npm run examples`; this will start a webpack dev server (the config file is found at [examples/webpack.config.js](examples/webpack.config.js)) making Radium examples available at `http://localhost:8080`.

Please note that if you use `npm run test-dev` as above, Karma will use port `8080`. You can change the port used by the examples by running `npm run examples -- --port 8000`.

### Before submitting a PR...

Thanks for taking the time to help us make Radium even better! Before you go ahead and submit a PR, make sure that you have done the following:

- Run the tests (you did add tests, right?) using `npm run test-dev`.

- Run lint and flow using `npm run lint`

## Releasing a new version to NPM (only for project administrators):

1. Update `CHANGELOG.md`, following format for previous versions

2. Commit as "Changelog for version 0.XX.Y"

3. Run `npm version patch` (or `minor`, `major` as appropriate) to run tests and lint, build the `lib` ands `dist` directories, , then update `package.json` and add a git tag.

4. Run `npm publish` and publish to NPM if all is well.

5. Run `git push && git push --tags`

## Contributor Covenant Code of Conduct

### Our Pledge

In the interest of fostering an open and welcoming environment, we as

contributors and maintainers pledge to making participation in our project and

our community a harassment-free experience for everyone, regardless of age, body

size, disability, ethnicity, gender identity and expression, level of experience,

nationality, personal appearance, race, religion, or sexual identity and

orientation.

### Our Standards

Examples of behavior that contributes to creating a positive environment

include:

\* Using welcoming and inclusive language

\* Being respectful of differing viewpoints and experiences

\* Gracefully accepting constructive criticism

\* Focusing on what is best for the community

\* Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

\* The use of sexualized language or imagery and unwelcome sexual attention or

advances

\* Trolling, insulting/derogatory comments, and personal or political attacks

\* Public or private harassment

\* Publishing others' private information, such as a physical or electronic

address, without explicit permission

\* Other conduct which could reasonably be considered inappropriate in a

professional setting

### Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable

behavior and are expected to take appropriate and fair corrective action in

response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or

reject comments, commits, code, wiki edits, issues, and other contributions

that are not aligned to this Code of Conduct, or to ban temporarily or

permanently any contributor for other behaviors that they deem inappropriate,

threatening, offensive, or harmful.

### Scope

This Code of Conduct applies both within project spaces and in public spaces

when an individual is representing the project or its community. Examples of

representing a project or community include using an official project e-mail

address, posting via an official social media account, or acting as an appointed

representative at an online or offline event. Representation of a project may be

further defined and clarified by project maintainers.

### Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be

reported by contacting the project team at lauren.eastridge@formidable.com. All

complaints will be reviewed and investigated and will result in a response that

is deemed necessary and appropriate to the circumstances. The project team is

obligated to maintain confidentiality with regard to the reporter of an incident.

Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good

faith may face temporary or permanent repercussions as determined by other

members of the project's leadership.

### Attribution

This Code of Conduct is adapted from the [Contributor Covenant][homepage], version 1.4,

available at [http://contributor-covenant.org/version/1/4][version]

[homepage]: http://contributor-covenant.org

[version]: http://contributor-covenant.org/version/1/4/

[version]: http://contributor-covenant.org/version/1/4/

### Architecture

#### Enhancer HOC (enhancer.js)

- Component is wrapped by `Radium` which runs it through `enhancer.js::enhanceWithRadium()`.

- A composed component is created

- ES modules - `Reflect` is used to create enhanced component from prototype of original component

- Stateless components - a simple wrapper component is created which calls the original component

- Everything else - enhanced component is created by shallow copying the original component

- An enhanced component is created

- static `\_isRadiumEnhanced = true` property is added

- `\_radiumStyleState = {}` is added to component state

- In render, `resolveStyles.js:resolveStyles` is called on the original rendered elements before they are returned. (see resolveStyles section)

- in componentDidUpdate, unused state keys are removed if necessary

- in componentWillUnmount, mouse and media query listeners are removed

#### Style Resolver `resolveStyles.js::resolveStyles`

- Called on the original rendered elements before they are returned. It iterates over the elements and children, rewriting props to add event handlers required to capture user interactions (e.g. mouse over). It also replaces the style prop because it adds in the various interaction styles (e.g. :hover).

- extraStateKeyMap is created by determining which state fields are no longer needed [?]

- If the rendered elements are an array, they are recursively mapped over with resolveStyles

- New children are resolved

- Recurse into children and call `resolveStyles` on each one

- New props are resolved

- Recurse through props and call `resolveStyles` on any React components in props. Otherwise return props as-is

- Plugins are run ONLY on simple ReactDOM elements that have a style prop

- plugins are called with the component, its props and a bunch of helper functions

- if the props were changed/added by any plugin, then return those

- see plugins section for more

- If anything changed, clone the element and return that. Otherwise just return the element.

#### Plugins

- \_merge-styles-array-plugin\_ - If the component styles are an array, it deeply merges them. Otherwise, it returns them unmodified.

- \_check-props-plugin\_ - Recursively checks props and warns (in dev mode) if shorthand and longhand are mixed.

- \_resolve-media-queries-plugin\_ - Handles media query style entries (like '@media (...)': { ... }), applying them only when the appropriate media query is hit

- \_resolve-interaction-styles-plugin\_ - Handles `:hover`, `:active` and `:focus` styles by adding/wrapping mouse event listeners with functions that update radium's state.

- \_keyframes-plugin\_ - Handles keyframe styles

- \_visited-plugin\_ - Handles `:visited` styles

- \_remove-nested-styles-plugin\_ - Recursively flattens nested styles into a flat object

- \_prefix-plugin\_ - Uses browser detection and a mapping to add vendor prefixes to CSS properties and values as needed