# Tips for Contributors

\* [docs/contributor-info/Releasing](./docs/contributor-info/releasing.md) for instructions on releasing.

\* [docs/contributor-info/pull-requests](./docs/contributor-info/pull-requests.md)

\* See other docs in [docs/contributor-info](./docs/contributor-info)

## To run tests:

\* After updating code via git, to prepare all examples and run all tests:

```sh

cd react\_on\_rails/

bundle && yarn && rake examples:gen\_all && rake node\_package && rake

```

In order to run tests in browser

```

yarn global add browserify babelify tape-run faucet

browserify -t babelify node\_package/tests/\*.js | tape-run | faucet

```

See Dev Initial Setup, below for, well... initial setup.

# IDE/IDE SETUP

It's critical to configure your IDE/editor to ignore certain directories. Otherwise your IDE might slow to a crawl!

\* /coverage

\* /tmp

\* /gen-examples

\* /node\_package/lib

\* /node\_modules

\* /spec/dummy/app/assets/webpack

\* /spec/dummy/log

\* /spec/dummy/node\_modules

\* /spec/dummy/client/node\_modules

\* /spec/dummy/tmp

\* /spec/react\_on\_rails/dummy-for-generators

# Configuring your test app to use your local fork

You can test the `react-on-rails` gem using your own external test app or the gem's internal `spec/dummy` app. The `spec/dummy` app is an example of the various setup techniques you can use with the gem.

```

??? test\_app

| ??? client

??? react\_on\_rails

??? spec

??? dummy

```

## Testing the Ruby Gem

If you want to test the ruby parts of the gem with an application before you release a new version of the gem, you can specify the path to your local version via your test app's Gemfile:

```ruby

gem "react\_on\_rails", path: "../path-to-react-on-rails"

```

Note that you will need to bundle install after making this change, but also that \*\*you will need to restart your Rails application if you make any changes to the gem\*\*.

## Testing the Node package for react-on-rails via Yalc

In addition to testing the Ruby parts out, you can also test the node package parts of the gem with an external application. First, be \*\*sure\*\* to build the NPM package:

```sh

cd react\_on\_rails/

yarn

# Update the lib directory with babel compiled files

yarn run build:watch

```

You need to do this once:

```

# Will send the updates to other folders

yalc publish

cd spec/dummy

yalc add react-on-rails

```

The workflow is:

1. Make changes to the node package.

2. We need yalc to push and then run yarn:

```

cd <top dir>

# Will send the updates to other folders

yalc push

cd spec/dummy

# Will update from yalc

yarn

```

When you run `yalc push`, you'll get an informative message

```

? yalc push

react-on-rails@12.0.0-12070fd1 published in store.

Pushing react-on-rails@12.0.0 in /Users/justin/shakacode/react-on-rails/react\_on\_rails/spec/dummy

Package react-on-rails@12.0.0-12070fd1 added ==> /Users/justin/shakacode/react-on-rails/react\_on\_rails/spec/dummy/node\_modules/react-on-rails.

Don't forget you may need to run yarn after adding packages with yalc to install/update dependencies/bin scripts.

```

#### Example: Testing NPM changes with the dummy app

1. Add `console.log('Hello!')` to [clientStartup.ts, function render](https://github.com/shakacode/react\_on\_rails/blob/master/node\_package/src/clientStartup.ts in `/node\_package/src/clientStartup.js` to confirm we're getting an update to the node package client side. Do the same for function `serverRenderReactComponent` in `/node\_package/src/serverRenderReactComponent.ts`.

2. Refresh the browser if the server is already running or start the server using `foreman start` from `react\_on\_rails/spec/dummy` and navigate to `http://localhost:5000/`. You will now see the `Hello!` message printed in the browser's console. If you did not see that message, then review the steps above for the workflow of making changes and pushing them via yalc.

# Development Setup for Gem and Node Package Contributors

## Dev Initial Setup

### Prereqs

After checking out the repo, making sure you have rvm and nvm setup (setup ruby and node), cd to `spec/dummy` and run `bin/setup` to install ruby dependencies. You can also run `bin/console` for an interactive prompt that will allow you to experiment.

### Local Node Package

Note, the example and dummy apps will use your local node\_package folder as the react-on-rails node package. This will also be done automatically for you via the `rake examples:gen\_all` rake task.

\*Side note: It's critical to use the alias section of the webpack config to avoid a double inclusion error. This has already been done for you in the example and dummy apps, but for reference:\*

```js

resolve: {

alias: {

react: path.resolve('./node\_modules/react'),

'react-dom': path.resolve('./node\_modules/react-dom'),

},

},

```

### Install NPM dependencies and build the NPM package for react-on-rails

```sh

cd react\_on\_rails/

yarn

yarn build

```

Or run this which builds the yarn package, then the webpack files for spec/dummy, and runs tests in

spec/dummy.

```sh

# Optionally change default selenium\_firefox driver

export DRIVER=selenium\_firefox

cd react\_on\_rails/

yarn run dummy:spec

```

### Run NPM JS tests

```sh

cd react\_on\_rails/

yarn test

```

### Run spec/dummy tests

```sh

cd react\_on\_rails/spec/dummy

rspec

```

### Run most tests and linting

```sh

cd react\_on\_rails/

yarn run check

```

### Starting the Dummy App

To run the dummy app, it's \*\*CRITICAL\*\* to not just run `rails s`. You have to run `foreman start` with one of the Procfiles. If you don't do this, then `webpack` will not generate a new bundle, and you will be seriously confused when you change JavaScript and the app does not change. If you change the webpack configs, then you need to restart foreman. If you change the JS code for react-on-rails, you need to run `yarn run build`. Since the react-on-rails package should be sym linked, you don't have to `yarn react-on-rails` after every change.

### RSpec Testing

Run `rake` for testing the gem and `spec/dummy`. Otherwise, the `rspec` command only works for testing within the sample apps, like `spec/dummy`.

If you run `rspec` at the top level, you'll see this message: `require': cannot load such file -- rails\_helper (LoadError)`

After running a test, you can view the coverage results SimpleCov reports by opening `coverage/index.html`.

Turbolinks 5 is included in the test app, unless "DISABLE\_TURBOLINKS" is set to YES in the environment.

Run `rake -T` or `rake -D` to see testing options.

`rake all\_but\_examples` is typically best for developers, except if any generators changed.

See below for verifying changes to the generators.

### Install Generator

In your Rails app add this gem with a path to your fork.

```ruby

gem 'react\_on\_rails', path: '../relative/path/to/react\_on\_rails'

```

Then run `bundle`.

The main installer can be run with ```rails generate react\_on\_rails:install```

Then use yalc to add the npm module.

Be sure that your ran this first at the top level of React on Rails

```

yalc publish

```

Then add the node package to your test app:

```

yalc add react-on-rails

```

### Testing the Generator

The generators are covered by generator tests using Rails's generator testing helpers, but it never hurts to do a sanity check and explore the API. See [generator\_testing\_script.md](generator\_testing\_script.md) for a script on how to run the generator on a fresh project.

`rake run\_rspec:example\_basic` is a great way to run tests on one generator. Once that works, you should run `rake run\_rspec:examples`. Be aware that this will create a hug number of files under a `/gen-examples` directory. You should be sure to exclude this directory from your IDE and delete it once your testing is done.

### Linting

All linting is performed from the docker container for CI. You will need docker and docker-compose installed locally to lint code changes via the lint container. You can lint locally by running `npm run lint && npm run flow`

\* [Install Docker Toolbox for Mac](https://www.docker.com/toolbox)

\* [Install Docker Compose for Linux](https://docs.docker.com/compose/install/)

Once you have docker and docker-compose running locally, run `docker-compose build lint`. This will build the `reactonrails\_lint` docker image and docker-compose `lint` container. The initial build is slow, but after the install, startup is very quick.

### Linting Commands

Run `rake lint`.

Alternately with Docker:

Run `rake -D docker` to see all docker linting commands for rake. `rake docker:lint` will run all linters. For individual rake linting commands please refer to `rake -D docker` for the list.

You can run specific linting for directories or files by using `docker-compose run lint rubocop (file path or directory)`, etc.

`docker-compose run lint bash` sets you up to run from the container command line.

### Updating Rubocop

2 files require updating to update the Rubocop version:

1. `react\_on\_rails.gemspec`

2. `spec/dummy/Gemfile`

### Docker CI - Test and Linting

Docker CI and Tests containers have a xvfd server automatically started for headless browser testing with selenium and Firefox.

Run `docker-compose build ci` to build the CI container. Run `docker-compose run ci` to start all rspec tests and linting. `docker-compose run --entrypoint=/bin/bash` will override the default CI action and place you inside the CI container in a bash session. This is what is run on Travis-CI.

Run `docker-compose build tests` to build the tests container. Run `docker-compose run tests` to start all RSpec tests.

# Advice for Project Maintainers and Contributors

What do project maintainers do? What sort of work is involved? [sstephenson](https://github.com/sstephenson) wrote in the [turbolinks](https://github.com/turbolinks/turbolinks) repo:

> [Why this is not still fully merged?](https://github.com/turbolinks/turbolinks/pull/124#issuecomment-239826060)