# Contributing to CodeTriage

We want to make contributing to this project as straight forward as possible, whether it's:

- Reporting a bug

- Discussing the current state of the code

- Submitting a fix

- Proposing new features

## Code Changes Happen Through Pull Requests

We use github to host code, to track issues and feature requests, as well as accept pull requests.

Pull requests are the best way to propose changes to the codebase.

We actively welcome your pull requests:

1. Fork the repo and create your branch from `master`.

2. If you've added code that should be tested, add tests.

3. If you've changed APIs, update the documentation.

4. Ensure the test suite passes.

6. Issue the pull request using the PULL\_REQUEST\_TEMPLATE.md

## Report bugs using Github's issues

We use GitHub issues to track public bugs. Report a bug by [opening a new issue](https://github.com/codetriage/codetriage/issues);

## Write bug reports with detail, background, and sample code

See our ISSUE\_TEMPLATE.md when submitting an issue.

\*\*Bug Reports\*\* tend to have:

- A quick summary and/or background

- Steps to reproduce

- Give sample code if you can. Include sample code that \*anyone\* can run to reproduce.

- What you expected would happen

- What actually happens

- Notes (possibly including why you think this might be happening, or stuff you tried that didn't work)

# Run CodeTriage

### Dependencies

Make sure you have bundler, then install the dependencies:

```shell

$ gem install bundler

$ bundle install

```

### Install Redis

CodeTriage requires Redis for background processing.

\*\*Homebrew\*\*

If you're on OS X, Homebrew is the simplest way to install Redis:

```shell

$ brew install redis memcached && brew cask install phantomjs

$ redis-server

```

You now have Redis running on 6379.

\*\*Other\*\*

See the Download page on Redis.io for steps to install on other systems: [http://redis.io/download](http://redis.io/download)

### Database

\* Copy database.yml and if you need, setup your own database credentials

\* Create database (you must use PostgreSQL)

\* Run migrations

```shell

$ cp config/database.example.yml config/database.yml

$ bin/rake db:create db:schema:load db:seed

```

### Environment

If you want your users to sign up with Github, register a [GitHub a new OAuth Application](https://github.com/settings/applications/new). The urls you are asked to provide will be something like this:

- URL: `http://localhost:3000`

- Callback URL: `http://localhost:3000/users/auth/github/callback`

Then add the credentials to your .env file:

```shell

$ echo GITHUB\_APP\_ID=foo >> .env

$ echo GITHUB\_APP\_SECRET=bar >> .env

$ echo PORT=3000 >> .env

```

### Running the app

Start your app using [Heroku Local](https://devcenter.heroku.com/articles/heroku-local)

```shell

$ heroku local -f Procfile.development

12:00:03 AM web.1 | [70676] - Worker 0 (pid: 70696) booted, phase: 0

12:00:04 AM worker.1 | INFO: Booting Sidekiq with redis options {:url=>nil}

```

CodeTriage should now be running at [http://localhost:3000](http://localhost:3000)

## Tests

```shell

$ bin/rake db:create RAILS\_ENV=test

$ bin/rake db:schema:load RAILS\_ENV=test

```

You may need a github API token to run tests locally. You can get this by spinning your local server, clicking the "sign in" button and going through the OAuth flow.

Note you may need to create your own app on GitHub for CodeTriage

[here](https://github.com/settings/developers), and replace the values

previously set (via the above) for `GITHUB\_APP\_ID` and `GITHUB\_APP\_SECRET` in

order to complete the OAuth flow locally.

Once you've done this spin down your server and run this:

```

$ bin/rails c

> `echo GITHUB\_API\_KEY=#{User.last.token} >> .env`

```

Make sure it shows up in your `.env`:

```

> puts File.read(".env")

```

Now you should be able to run tests

```

$ bin/rake test

```

## Writing tests

If you need to mock out some github requests please use VCR. Put anything that may create an external request inside of a vcr block:

```

VCR.use\_cassette('my\_vcr\_cassette\_name\_here') do

# ... code goes here

end

```

Make sure to name your cassette something unique. The first time you run tests you'll need to set a [record mode](https://relishapp.com/vcr/vcr/v/2-8-0/docs/record-modes). This will make a real-life request to github using your `GITHUB\_API\_KEY` you specified in the `.env` and record the result. The next time you run your tests it should use your "cassette" instead of actually hitting github. All secrets including your `GITHUB\_API\_KEY` are filtered out, so you can safely commit the resultant. When running on travis the VCR cassettes are used to eliminate/minimize actual calls to Github.

The benefit of using VCR over stubbing/mocking methods is that we could swap out implementations if we wanted.

Make sure to remove any record modes from your VCR cassette before committing.

## Modifying Tests

If you need to modify a test locally and the VCR cassette doesn't have the correct information, you can run with [new episodes](https://relishapp.com/vcr/vcr/v/2-8-0/docs/record-modes/new-episodes) enabled.

Make sure to remove any record modes from your VCR cassette before committing.

## Flow

- A user subscribes to a repo

- Consume API: Once a day, find all the repos that haven't been updated in 24 hours, produce issue subscription.

- Issue Assigning [repo]: Find all users subscribed to that repo that haven't been assigned an issue in 24 hours, pick a random issue that the user is not a part of and send them an email.

## Any contributions you make will be under the MIT Software License

In short, when you submit code changes, your submissions are understood to be under the same MIT License

(see LICENSE.txt) that covers the project. Feel free to contact the maintainers if that's a concern.

By contributing, you agree that your contributions will be licensed under its MIT License.