# Contributing

HI! Thanks you for your interest in Puppeteer Recorder! We'd love to accept your patches and contributions, but please remember that this project was started first and foremost to serve the users of the Checkly API and Site transaction monitoring service.

## New feature guidelines

When authoring new features or extending existing ones, consider the following:

- All new features should be accompanied first with a Github issues describing the feature and its necessity.

- We aim for simplicity. Too many options, buttons, panels etc. detract from that.

- Features should serve the general public. Very specific things for your use case are frowned upon.

## Getting set up

1. Clone this repository

```bash

git clone https://github.com/checkly/puppeteer-recorder

cd puppeteer-recorder

```

2. Install dependencies

```bash

npm install

```

## Code reviews

All submissions, including submissions by project members, require review. We

use GitHub pull requests for this purpose. Consult

[GitHub Help](https://help.github.com/articles/about-pull-requests/) for more

information on using pull requests.

> Note: one pull request should cover one, atomic feature and/or bug fix. Do not submit pull requests with a plethora of updates, tweaks, fixes and new features.

## Code Style

- Coding style is fully defined in [.eslintrc](https://github.com/checkly/puppeteer-recorder/blob/master/.eslintrc.js)

- Comments should be generally avoided. If the code would not be understood without comments, consider re-writing the code to make it self-explanatory.

To run code linter, use:

```bash

npm run lint

```

## Commit Messages

Commit messages should follow the Semantic Commit Messages format:

```

label(namespace): title

description

footer

```

1. \*label\* is one of the following:

- `fix` - puppeteer bug fixes.

- `feat` - puppeteer features.

- `docs` - changes to docs, e.g. `docs(api.md): ..` to change documentation.

- `test` - changes to puppeteer tests infrastructure.

- `style` - puppeteer code style: spaces/alignment/wrapping etc.

- `chore` - build-related work, e.g. doclint changes / travis / appveyor.

2. \*namespace\* is put in parenthesis after label and is optional.

3. \*title\* is a brief summary of changes.

4. \*description\* is \*\*optional\*\*, new-line separated from title and is in present tense.

Example:

```

fix(code-generator): fix page.pizza method

This patch fixes page.pizza so that it works with iframes.

Fixes #123, Fixes #234

```

## Adding New Dependencies

For all dependencies (both installation and development):

- \*\*Do not add\*\* a dependency if the desired functionality is easily implementable.

- If adding a dependency, it should be well-maintained and trustworthy.

A barrier for introducing new installation dependencies is especially high:

- \*\*Do not add\*\* installation dependency unless it's critical to project success.

## Writing Tests

- Every feature should be accompanied by a test.

- Every public api event/method should be accompanied by a test.

- Tests should be \*hermetic\*. Tests should not depend on external services.

We use Jest for testing. Tests are located in the various `\_\_test\_\_` folders.

- To run all tests:

```bash

npm run test

```