# Contributing to Mockoon

There are many ways to contribute to Mockoon: opening bugs or issues, submitting pull requests, suggesting new features...

Before contributing please check the opened and unassigned issues. Any pull request should link to an opened issue. Pull requests not solving issues may not be accepted.

If you want to contribute significant changes please discuss with the maintainers before starting to work.

Please note that Mockoon promise is to be simple, easy to use, and fast. So not all features are worth implementing and maintainer may chose to not implement features that are out of Mockoon's scope.

> Prerequisites:

> - Node.js ^12.x.x

### Find an issue to work on

- Check for \*\*opened unassigned\*\* issues or open a new one (after searching for closed issues).

- Comment on the issue and request to work on it.

- After discussing the implementation, the issue will be assigned to you by a maintainer. As a rule, the assignee is the person working on the issue.

Please respect this workflow to ensure that:

- Your work is in line with Mockoon direction.

- It hasn't been already done/rejected.

- You are the only one working on an issue.

### Run the application in dev mode

- Clone the repository: `git@github.com:mockoon/mockoon.git`

- Run `npm install`.

- Run `npm run serve:app` then `npm run serve:electron` when the first command finish (`dist` folder must be available for Electron). Or directly run `npm run serve` which will run both commands in a row.

You will get hot reload on both Angular and Electron applications.

### Work on your feature or bugfix

- Start your `feature` or `fix` from `master`

- Cover it with spectron tests. You will find them in the `test` folder. Please try to cover at least the easiest test cases of your feature.

- Preferably squash your commits, except when it makes sense to keep them separate (one refactoring + feature development)

- Do not forget to add "Closes #xx" in one of the commit messages or in the pull request description (where xx is the GitHub issue number)

Branches naming convention:

- features and enhancements: `feature/name-or-issue-number`

- bug fixes: `fix/name-or-issue-number`

### Run the tests

Tests are written with Spectron and you can run them using `npm run test`. These tests will also be run on each commit or pull request by CircleCI.

When running the tests locally, you will first need to run `npm run build:dev` in order to have an application build to test against.

### Open a pull request

Open a pull request to be merge in the `master` branch. All branches should start from `master` and must be merged into `master`.

Ask maintainers to review the code and be prepared to rework your code if it does not match the style or do not follow the way it's usually done (typing, reducer, etc).

---

### \*\*[Maintainers only]\*\* Build and package the application for production

- Increment the version (which follows [semver](https://semver.org/)) in package.json file.

- Push.

- Create a release in GitHub 'mockoon' repository with a new tag. Respect the releases tag format `vx.x.x`.

\*\*/!\\ Mark the release as a pre-release, and only set it as a final release when all binaries are successfully build, tested and uploaded. /!\\\*\*

Binaries build will be automatically triggered through GitHub Actions. It will basically run `npm run build:prod` and package the application for different platforms with `npm run package:win|mac|linux`. Including Windows/macOS code signing (and notarization).

Next steps are:

- Download all the binaries from the GitHub Action, test them and add them to the new GitHub release.

- Publish the release (remove "pre-release" label).

Note that Windows and Mac OS versions need to be signed (and notarized) when packaged. This is the responsibility of @255kb.

\*\*/!\\ Auto update depends on GitHub release proper taging (`vx.x.x`) and binaries correct naming (as set in `package.json` and `update.service.ts`). Do not change them. /!\\\*\*

### Distribute the application

Only maintainers (@255kb) are entitled to build and package the application with Windows code signing and macOS certificates.

Some manual steps are required in order to properly distribute the application:

- A pull request must be created to update Homebrew cask repository (update the `version` and the `sha256` hash in the `Casks/mockoon.rb` file).

- Binary must be uploaded to the Snap store manually.

- Arch Linux repository must be updated (Docker image and script can be used in `./scripts/aur-version-bump`).

- Chocolatey package should be automatically updated after some days.