# CONTRIBUTING

Thanks for your interest in FoalTS!

There are several ways to contribute. \*\*Reporting bugs are greatly appreciated\*\*, so do not hesitate to open an issue/PR for that!

## Security Vulnerabilities

If you think you have found a security hole, please do NOT submit an issue but send an email directly to loic.poullain@centraliens.net.

## Pull Requests

There are [pending issues](https://github.com/FoalTS/foal/issues?q=is%3Aissue+is%3Aopen+label%3A%22help+wanted%22) that may require your help.

If you wish to submit a PR, please first submit an issue for discussion (or add a comment on an existing issue).

PRs that correct grammatical errors or small bugs can be submitted directly.

## Development Environment

The framework development environment uses [lerna](https://lernajs.io/) for managing packages and [docker](https://www.docker.com/) for database provisioning.

\*\*Steps:\*\*

1. Install docker.

2. Install lerna

```

npm install -g lerna

```

3. Start the databases.

```sh

npm run start-docker # use `npm run stop-docker` to stop them

```

4. Install the root dependencies.

```

npm install

```

5. Install the dependencies of each package and build each package.

```

lerna bootstrap

```

6. Check code format.

```

npm run lint

```

7. Run all the tests.

```

lerna run --no-bail test

```

Tests can also be run individually for each package using `npm run test` or `npm run dev:test` (watch mode) at the root of the package directory.

## Dependency Policy

\*\*Do not add new dependencies\*\* (unless they have been improved). Do not install `@types` packages.

FoalTS is based on very few dependencies for all these reasons:

- Adding a new dependency often means installing many other packages on which it depends. This phenomenon is often referred to as a \*black hole\* in Node's ecosystem.

- The size of the `node\_modules` directory grows very fast. This can slow down deployment and cause problems if a size limit is imposed on the directory (e.g. in a serverless architecture).

- Due to the large number of dependencies to load, the application may be slow to start.

- The application is more vulnerable to the release of malicious packages. This is what happened on July 12, 2018 when an [attacker compromised the npm account](https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes) of an ESLint maintainer.

- We have no guarantee that the maintainers follow the same Foal safety rules (2FA enabled on both Github and npm).

- When a new version of an external package is released (bug fixes, security updates, new features, etc.), it takes time to review each change made in the new version and time to verify that the framework still works as expected with it.

- Packages may support different versions of TypeScript and Node than those supported by the framework.

- External packages can become unmaintained.

- Semantic versioning is not always respected, which is problematic if we want to integrate a security update without introducing breaking changes.

- If we need a new feature in the external dependency, it may take time for the maintainer(s) to implement it. The feature may also be rejected.

- The `@types` packages very often lead to issues.

- The types may be outdated with respect to the current version.

- Semantic versioning is often not respected, which causes the code to break between two \*patch\* versions.

- Type choices may be arbitrary and not decided by the official maintainers.

- Two packages using the same `@types` module but with different versions may not work properly together.

- Type packages depend on each other by specifying `\*` as the version number which causes incompatibilities and great difficulty in defining a replicable environment.

- The installation is often polluted by messages of indirect dependencies in search of funds.

Some packages, however, can override this policy and be installed if they meet one of the following criteria:

- Rewriting the entire package would require too much work and would be difficult to maintain in the long term. Examples: `TypeORM`, `Mongoose`.

- The code requires very specific knowledge. Examples: `pump`, `jsonwebtoken`, `TypeORM`, `Mongoose`.

- The packages are base packages of the Express.Js framework and can therefore be considered stable, safe and mature. Examples: `cookie-parser`, `morgan`.

> Dependencies (except peer ones) should point to \*minor\* versions (`~1.2.0` instead of `^1.2.0`).

## Testing and Documentation Policy

\*\*Testing and documentating the framework is put on a very high priority\*\*. Each line of code must be tested. It is okay to delay the release of a new version if it is to ensure that it is based on robust testing.

If you wish to submit a PR, please use the \*Test-Driven Developpement (TDD)\* approach:

1. Write a test.

2. Check that the test fails.

3. Write just enough code to make the test pass.

4. Check that the test succeeds.

5. Reiterate.

This method may seem cumbersome at first glance, but it ensures that every line of code in the framework is tested. Reviewers must pull the branch and verify that the tests are actually testing something. If they change even one line of code, they must see that at least one of the tests fails.

A PR without robust tests is automatically rejected.

## Semantic Versioning

The framework follows the semantic versioning specification.

| Code status | Stage | Example version |

| --- | --- | --- |

| Backward compatible bug fixes | Patch release | 1.0.1 |

| Backward compatible new features | Minor release | 1.1.0 |

| Changes that break backward compatibility | Major release | 2.0.0 |

## Long-Term Support Policy and Schedule

All of major releases are supported for 18 months.

- 12 months of \*active support\* (new features, bug fixes, etc).

- 6 months of \*maintenance (LTS)\* (critical fixes and security patches).

| Release | Status | Active Start | Maintenance Start | End-of-life | Node min version | TS min version |

| --- | --- | --- | --- | --- | --- | --- |

| 2.x | \*Pending\* | Summer 2020 | Summer 2021 | 2021-12-31 | 10.x | 3.5 |

| 1.x | \*Active\* | 2019-07-11 | Summer 2020 | 2020-12-31 | 8.x | 3.5 |

| 0.8 | \*End-of-Life\* | 2019-02-16 | - | 2019-07-11 | 8.x | 2.9 |

## Project Architecture

### `@foal/cli` Package Structure

The directory `src/generate/` contains the source code of the commands `foal createapp` and `foal generate`.

Here is the list of its sub-directories:

| Directory | Description |

| --- | --- |

| generators | Contains the code which renders the templates or updates the files |

| fixtures | Contains some pieces of code used to test the file "updaters" |

| specs | Defines how the generated files should look like in different scenarios (specifications) |

| templates | Contains the actual templates used to generate the files |

| utils | Contains some helpers shared by all the generators |

## Conventions

### Import declarations

Import declarations should be organized in three distinct blocks depending on if they refer to the standard library, a 3P package or a FoalTS component.

Example:

```typescript

// std

import { strictEqual } from 'assert';

// 3p

import { Column } from 'typeorm';

// FoalTS

import { something } from '../somewhere';

```