# Apollo Contributor Guide

Excited about Apollo and want to make it better? We’re excited too!

Apollo is a community of developers just like you, striving to create the best tools and libraries around GraphQL. We welcome anyone who wants to contribute or provide constructive feedback, no matter the age or level of experience. If you want to help but don't know where to start, let us know, and we'll find something for you.

Oh, and if you haven't already, sign up for the [Apollo Slack](http://www.apollodata.com/#slack).

Here are some ways to contribute to the project, from easiest to most difficult:

\* [Reporting bugs](#reporting-bugs)

\* [Improving the documentation](#improving-the-documentation)

\* [Responding to issues](#responding-to-issues)

\* [Small bug fixes](#small-bug-fixes)

\* [Suggesting features](#suggesting-features)

\* [Big pull requests](#big-prs)

## Issues

### Reporting bugs

If you encounter a bug, please file an issue on GitHub via the repository of the sub-project you think contains the bug. If an issue you have is already reported, please add additional information or add a ? reaction to indicate your agreement.

While we will try to be as helpful as we can on any issue reported, please include the following to maximize the chances of a quick fix:

1. \*\*Intended outcome:\*\* What you were trying to accomplish when the bug occurred, and as much code as possible related to the source of the problem.

2. \*\*Actual outcome:\*\* A description of what actually happened, including a screenshot or copy-paste of any related error messages, logs, or other output that might be related. Places to look for information include your browser console, server console, and network logs. Please avoid non-specific phrases like “didn’t work” or “broke”.

3. \*\*How to reproduce the issue:\*\* Instructions for how the issue can be reproduced by a maintainer or contributor. Be as specific as possible, and only mention what is necessary to reproduce the bug. If possible, try to isolate the exact circumstances in which the bug occurs and avoid speculation over what the cause might be.

Creating a good reproduction really helps contributors investigate and resolve your issue quickly. In many cases, the act of creating a minimal reproduction illuminates that the source of the bug was somewhere outside the library in question, saving time and effort for everyone.

### Improving the documentation

Improving the documentation, examples, and other open source content can be the easiest way to contribute to the library. If you see a piece of content that can be better, open a PR with an improvement, no matter how small! If you would like to suggest a big change or major rewrite, we’d love to hear your ideas but please open an issue for discussion before writing the PR.

### Responding to issues

In addition to reporting issues, a great way to contribute to Apollo is to respond to other peoples' issues and try to identify the problem or help them work around it. If you’re interested in taking a more active role in this process, please go ahead and respond to issues. And don't forget to say "Hi" on Apollo Slack!

### Small bug fixes

For a small bug fix change (less than 20 lines of code changed), feel free to open a pull request. We’ll try to merge it as fast as possible and ideally publish a new release on the same day. The only requirement is, make sure you also add a test that verifies the bug you are trying to fix.

### Suggesting features

Most of the features in Apollo came from suggestions by you, the community! We welcome any ideas about how to make Apollo better for your use case. Unless there is overwhelming demand for a feature, it might not get implemented immediately, but please include as much information as possible that will help people have a discussion about your proposal:

1. \*\*Use case:\*\* What are you trying to accomplish, in specific terms? Often, there might already be a good way to do what you need and a new feature is unnecessary, but it’s hard to know without information about the specific use case.

2. \*\*Could this be a plugin?\*\* In many cases, a feature might be too niche to be included in the core of a library, and is better implemented as a companion package. If there isn’t a way to extend the library to do what you want, could we add additional plugin APIs? It’s important to make the case for why a feature should be part of the core functionality of the library.

3. \*\*Is there a workaround?\*\* Is this a more convenient way to do something that is already possible, or is there some blocker that makes a workaround unfeasible?

Feature requests will be labeled as such, and we encourage using GitHub issues as a place to discuss new features and possible implementation designs. Please refrain from submitting a pull request to implement a proposed feature until there is consensus that it should be included. This way, you can avoid putting in work that can’t be merged in.

Once there is a consensus on the need for a new feature, proceed as listed below under “Big PRs”.

## Big PRs

This includes:

- Big bug fixes

- New features

For significant changes to a repository, it’s important to settle on a design before starting on the implementation. This way, we can make sure that major improvements get the care and attention they deserve. Since big changes can be risky and might not always get merged, it’s good to reduce the amount of possible wasted effort by agreeing on an implementation design/plan first.

1. \*\*Open an issue.\*\* Open an issue about your bug or feature, as described above.

2. \*\*Reach consensus.\*\* Some contributors and community members should reach an agreement that this feature or bug is important, and that someone should work on implementing or fixing it.

3. \*\*Agree on intended behavior.\*\* On the issue, reach an agreement about the desired behavior. In the case of a bug fix, it should be clear what it means for the bug to be fixed, and in the case of a feature, it should be clear what it will be like for developers to use the new feature.

4. \*\*Agree on implementation plan.\*\* Write a plan for how this feature or bug fix should be implemented. What modules need to be added or rewritten? Should this be one pull request or multiple incremental improvements? Who is going to do each part?

5. \*\*Submit PR.\*\* In the case where multiple dependent patches need to be made to implement the change, only submit one at a time. Otherwise, the others might get stale while the first is reviewed and merged. Make sure to avoid “while we’re here” type changes - if something isn’t relevant to the improvement at hand, it should be in a separate PR; this especially includes code style changes of unrelated code.

6. \*\*Review.\*\* At least one core contributor should sign off on the change before it’s merged. Look at the “code review” section below to learn about factors are important in the code review. If you want to expedite the code being merged, try to review your own code first!

7. \*\*Merge and release!\*\*

### Code review guidelines

It’s important that every piece of code in Apollo packages is reviewed by at least one core contributor familiar with that codebase. Here are some things we look for:

1. \*\*Required CI checks pass.\*\* This is a prerequisite for the review, and it is the PR author's responsibility. As long as the tests don’t pass, the PR won't get reviewed.

2. \*\*Simplicity.\*\* Is this the simplest way to achieve the intended goal? If there are too many files, redundant functions, or complex lines of code, suggest a simpler way to do the same thing. In particular, avoid implementing an overly general solution when a simple, small, and pragmatic fix will do.

3. \*\*Testing.\*\* Do the tests ensure this code won’t break when other stuff changes around it? When it does break, will the tests added help us identify which part of the library has the problem? Did we cover an appropriate set of edge cases? Look at the test coverage report if there is one. Are all significant code paths in the new code exercised at least once?

4. \*\*No unnecessary or unrelated changes.\*\* PRs shouldn’t come with random formatting changes, especially in unrelated parts of the code. If there is some refactoring that needs to be done, it should be in a separate PR from a bug fix or feature, if possible.

5. \*\*Code has appropriate comments.\*\* Code should be commented, or written in a clear “self-documenting” way.

6. \*\*Idiomatic use of the language.\*\* In TypeScript, make sure the typings are specific and correct. In ES2015, make sure to use imports rather than require and const instead of var, etc. Ideally a linter enforces a lot of this, but use your common sense and follow the style of the surrounding code.