Contributing to MUI

If you're reading this, you're awesome! Thank you for helping us make this project great and being a part of the MUI community. Here are a few guidelines that will help you along the way.

Code of Conduct

MUI has adopted the Contributor Covenant as its Code of Conduct, and we expect project participants to adhere to it. Please read the full text so that you can understand what actions will and will not be tolerated.

A large spectrum of contributions

There are many ways to contribute to MUI, code contribution is one aspect of it. For instance, documentation improvements are as important as code changes.

Your first Pull Request

Working on your first Pull Request? You can learn how from this free video series:

How to Contribute to an Open Source Project on GitHub

To help you get your feet wet and get you familiar with our contribution process, we have a list of good first issues that contain changes that have a relatively limited scope. This label means that there is already a working solution to the issue in the discussion section. Therefore, it is a great place to get started.

We also have a list of good to take issues. This label is set when there has been already some discussion about the solution and it is clear in which direction to go. These issues are good for developers that want to reduce the chance of going down a rabbit hole.

You can also work on any other issue you choose to. The "good first" and "good to take" issues are just issues where we have a clear picture about scope and timeline. Pull requests working on other issues or completely new problems may take a bit longer to review when they don't fit into our current development cycle.

If you decide to fix an issue, please be sure to check the comment thread in case somebody is already working on a fix. If nobody is working on it at the moment, please leave a comment stating that you have started to work on it so other people don't accidentally duplicate your effort.

If somebody claims an issue but doesn't follow up for more than a week, it's fine to take it over but you should still leave a comment. If there has been no activity on the issue for 7 to 14 days, it is safe to assume that nobody is working on it.

Sending a Pull Request

MUI is a community project, so Pull Requests are always welcome, but, before working on a large change, it is best to open an issue first to discuss it with the maintainers.

When in doubt, keep your Pull Requests small. To give a Pull Request the best chance of getting accepted, don't bundle more than one feature or bug fix per Pull Request. It's often best to create two smaller Pull Requests than one big one.

1. Fork the repository.
2. Clone the fork to your local machine and add upstream remote:

```
git clone https://github.com/<your username>/material-ui.git
cd material-ui
git remote add upstream https://github.com/mui/material-ui.git
```

3. Synchronize your local `master` branch with the upstream one:

```
git checkout master
git pull upstream master
```

4. Install the dependencies with yarn (npm isn't supported):

```
yarn install
```

5. Create a new topic branch:

```
git checkout -b my-topic-branch
```

6. Make changes, commit and push to your fork:

```
git push -u origin HEAD
```

7. Go to the repository and make a Pull Request.

The core team is monitoring for Pull Requests. We will review your Pull Request and either merge it, request changes to it, or close it with an explanation.

Trying changes on the documentation site

The documentation site is built with MUI and contains examples of all the components. This is a great place to experiment with your changes. It's the local development environment used by the maintainers.

To get started:

```
yarn start
```

You can now access the documentation site locally. Changes to the docs will hot reload the site.

How to increase the chance of being accepted?

Continuous Integration (CI) runs a series of checks automatically when a Pull Request is opened. If you're not sure if your changes will pass, you can always open a Pull Request and the GitHub UI will display a summary of the results. If any of them fail, refer to Checks and how to fix them.

Make sure the following is true:

- The branch is targeted at `master` for ongoing development. We do our best to keep `master` in good shape, with all tests passing. Code that lands in `master` must be compatible with the latest stable release.

It may contain additional features, but no breaking changes. We should be able to release a new minor version from the tip of `master` at any time.

- If a feature is being added:
  - If the result was already achievable with the core library, explain why this feature needs to be added to the core.
  - If this is a common use case, consider adding an example to the documentation.
- When adding new features or modifying existing ones, please include tests to confirm the new behavior. You can read more about our test setup in our test README.
- If props were added or prop types were changed, the TypeScript declarations were updated.
- When submitting a new component, please add it to the lab.
- The branch is not behind its target branch.

Because we will only merge a Pull Request for which all tests pass. The following items need to be true:

- The code is formatted. If the code was changed, run `yarn prettier`.
- The code is linted. If the code was changed, run `yarn lint`.
- The code is type-safe. If TypeScript sources/declarations were changed, `yarn typescript` passed.
- The API docs are up-to-date. If API was changed, run `yarn proptypes && yarn docs:api`.
- The demos are up-to-date. If demos were changed, make sure `yarn docs:typescript:formatted` does not introduce changes. See about writing demos.
- The Pull Request title follows the pattern `[Component] Imperative commit message`. (See: How to Write a Git Commit Message for a great explanation).

If you have missed a step, don't worry, the Continuous Integration will run a thorough test on your commits and the maintainers of the project can assist.

Checks and how to fix them

If any of the checks fails click on the *Details* link and review the logs of the build to find out why it failed. For CircleCI you need to log in first. No further permissions are required to view the build logs. The following section gives an overview of what each check is responsible for.

ci/codesandbox

This task should not fail in isolation. It creates multiple sandboxes on CodeSandbox.com that use the version of MUI that was built from this Pull Request. Use it to test more complex scenarios.

ci/circleci: checkout

A preflight check to see if the dependencies and lockfile are ok. Running `yarn` and `yarn deduplicate` should fix most of the issues.

ci/circleci: test_static

Checks code format, and lints the repository. The log of the failed build should explain how to fix the issues.

ci/circleci: test_unit-1

Runs the unit tests in a `jsdom` environment. If this fails then `yarn test:unit` should[1] fail locally as well. You can narrow the scope of tests run with `yarn test:unit --grep ComponentName`. If `yarn test:unit` passes locally, but fails in CI, consider Accessibility tree exclusion in CI.

ci/circleci: test_browser-1

Runs the unit tests in multiple browsers (via BrowserStack). The log of the failed build should list which browsers failed. If Chrome failed then `yarn test:karma` should[1] fail locally as well. If other browsers failed debugging might be trickier. If `yarn test:karma` passes locally, but fails in CI, consider Accessibility tree exclusion in CI.

ci/circleci: test_regression-1

Renders tests in `test/regressions/tests` and makes screenshots. This step shouldn't fail if the others pass. Otherwise, a maintainer will take a look. The screenshots are evaluated in another step.

ci/circleci: test_types

Typechecks the repository. The log of the failed build should list all issues.

ci/circleci: test_bundle_size_monitor

This task is mostly responsible for monitoring the bundle size. It will only report the size if the change exceeds a certain threshold. If it fails there's usually something wrong with the way the packages or docs were built.

argos

Evaluates the screenshots taken in `test/regressions/tests` and fails if it detects differences. This doesn't necessarily mean your Pull Request will be rejected as a failure might be intended. Clicking on *Details* will show you the differences.

deploy/netlify

Renders a preview of the docs with your changes if it succeeds. Otherwise `yarn docs:build` or `yarn docs:export` usually fail locally as well.

codecov/project

Monitors coverage of the tests. A reduction in coverage isn't necessarily bad but it's always appreciated if it can be improved.

Misc

There are various other checks done by Netlify to check the integrity of our docs. Click on *Details* to find out more about them.

Updating the component API documentation

The component API in the component `propTypes` and under `docs/pages/api-docs` is auto-generated from the JSDoc in the TypeScript declarations. Be sure to update the documentation in the corresponding `.d.ts` files (e.g. `packages/mui-material/src/Button/Button.d.ts` for `<Button>`) and then run:

```
$ yarn proptypes
$ yarn docs:api
```

Coding style

Please follow the coding style of the project. MUI uses prettier and eslint, so if possible, enable linting in your editor to get real-time feedback.

- `yarn prettier` reformats the code.
- `yarn lint` runs manually the linting rules.

Finally, when you submit a Pull Request, they are run again by our continuous integration tools, but hopefully, your code is already clean!

How to add a new demo in the documentation

If, for example, you want to add new demos for the button component, you have to take the following steps:

1. Add a new React component file under the related directory

In this case, you are going to add the new file to the following directory:

```
docs/src/pages/components/buttons/
```

and give it a name: `SuperButtons.js`.

2. Edit the page Markdown file

The Markdown file is the source for the website documentation. So, whatever you wrote there will be reflected on the website. In this case, the file you need to edit is `docs/src/pages/components/buttons/buttons.md`.

Changes should only be applied to the English version e.g. only `app-bar.md` and not `app-bar-de.md`. For contributions concerning translations please read the section about translations.

```
+### Super buttons
+
+Sometimes, you need a super button to make your app looks **superb**. Yea ...
+
+{{"demo": "pages/components/buttons/SuperButtons.js"}}
```

3. Write the content of the demo

MUI documents how to use this library with TypeScript.

If you are familiar with this language, write the demo in TypeScript, and only, in a *.tsx file. When you're done run `yarn docs:typescript:formatted` to automatically create the JavaScript version.

If you are not familiar with that language, write the demo in JavaScript, a core contributor might help you to migrate it to TypeScript.

4. You are done 🎉

In case you missed something, we have a real example that can be used as a summary report.

How can I use a change that wasn't released yet?

Codesandbox CI is used to publish a working version of the packages for each pull request, "a preview".

In practice, you can check the Codesandbox CI status of a pull request to get the URL needed to install these preview packages:

```
diff --git a//package.json b//package.json
index 791a7da1f4..a5db13b414 100644
--- a/package.json
+++ b/package.json
@@ -61,7 +61,7 @@
   "dependencies": {
     "@babel/runtime": "^7.4.4",
     "@mui/styled-engine": "^5.0.0-alpha.16",
-    "@mui/material": "^5.0.0-alpha.15",
```

```
+    "@mui/material": "https://pkg.csb.dev/mui/material-
ui/commit/371c952b/@mui/material",
     "@mui/system": "^5.0.0-alpha.16",
```

Alternatively, you can open the Netlify preview of the documentation, and open any demo in Codesandbox. The documentation automatically configures the dependencies to use the preview packages.

Translations

Translations are handled via Crowdin. You don't need to apply any changes to localized versions of our markdown files i.e. files having a `-locale` suffix. Crowdin automatically takes care of syncing these changes across the localized versions.

Roadmap

To get a sense of where MUI is heading, or for ideas on where you could contribute, take a look at the roadmap.

License

By contributing your code to the mui/material-ui GitHub repository, you agree to license your contribution under the MIT license.