Contributing

------------

Looking to contribute something to Materialize? \*\*Here's how you can help.\*\*

### Table of contents:

- [Introduction](#introduction)

- [Communication channels](#communication-channels)

- [Using the issue tracker](#using-the-issue-tracker)

- [Issues and labels](#issues-and-labels)

- [Bug reports](#bug-reports)

- [Feature Requests](#feature-requests)

- [Code Examples](#code-examples)

- [Pull Requests](#pull-requests)

- [Documentation](#documentation)

- [Submitting Your Pull Request](#submitting-your-pull-request)

- [Translations](#translations)

- [Jasmine Testing Guide](#jasmine-testing-guide)

## Introduction

Please take a moment to review this document in order to make the contribution

process easy and effective for everyone involved.

Following these guidelines helps to communicate that you respect the time of

the developers managing and developing this open source project. In return,

they should reciprocate that respect in addressing your issue or assessing

patches and features.

### Communication channels

Before you now get lost in the repository, here are a few starting points for you to check out. You might find that others have had similar questions or that your question rather belongs in one place than another.

\* Chat: https://gitter.im/Dogfalo/materialize

\* Website: http://materializecss.com

\* Twitter: https://twitter.com/materializecss

## Using the issue tracker

The [issue tracker](https://github.com/Dogfalo/materialize/issues) is the preferred channel for [bug reports](#bug-reports), [features requests](#feature-requests) and [submitting pull requests](#pull-requests), but please respect the following conditions:

\* Please \*\*do not\*\* use the issue tracker for personal support requests. [Stack Overflow `materialize`](https://stackoverflow.com/questions/tagged/materialize) tag is the best place to get help or use our [Gitter channel](https://gitter.im/Dogfalo/materialize).

\* Please \*\*do not\*\* post comments like "+1" or ":thumbsup:". Use [GitHub's "reactions" feature](https://github.com/blog/2119-add-reactions-to-pull-requests-issues-and-comments) instead. We reserve the right to delete comments which violate this rule.

\* Please \*\*do not\*\* open issues without clearly stating the problem and desired result. [See the bug reports section](#bug-reports) for more information on creating effective issues.

\* Please \*\*do\*\* [search for duplicate or closed issues](https://github.com/Dogfalo/materialize/issues?utf8=%E2%9C%93&q=is%3Aissue) and make sure to go through our [labels](https://github.com/Dogfalo/materialize/labels), before you open a new issue. Duplicate issues will be closed.

\* Please \*\*close your own issue\*\* once it is resolved.

\* Every participant is \*\*expected to follow\*\* the project's [Code of Conduct](CODE\_OF\_CONDUCT.md) so please be courteous and respectful.

### Issues and labels

Our bug tracker utilizes several labels to help organize and identify issues. Here's what they represent and how we use them:

- `component:\*` - Each component has been provided a label. Add the labels for any component the issue is triggered by.

- `confirmed` - Issues that have been confirmed with a reduced test case and identify a bug in Materialize.

- `css` - Issues stemming from our compiled CSS or source Sass files.

- `js` - Issues stemming from our compiled or source JavaScript files.

- `docs` - Issues for improving or updating our documentation.

- `help-wanted` - Issues we need or would love help from the community to resolve.

- `meta` - Issues with the project itself or our GitHub repository.

- `on-hold` Issues or pull that should be worked on, but for any reason it won't be worked on/accepted as a PR anytime soon.

For a complete look at our labels, see the [project labels page](https://github.com/Dogfalo/materialize/labels).

### Bug reports

A bug is a \_demonstrable problem\_ that is caused by the code in the repository.

Good bug reports are extremely helpful! Unclear issues with little explanations will be closed.

#### Guidelines for bug reports:

1. \*\*Use the GitHub issue search\*\* - check if the issue has already been reported.

2. \*\*Check if the issue has been fixed\*\* - try to reproduce it using the latest `master` or development branch in the repository.

3. \*\*Isolate the problem\*\* &mdash; create a [reduced test case](https://css-tricks.com/reduced-test-cases/) using \*\*our\*\* [Codepen template](#code-examples).

A good bug report shouldn't leave others needing to chase you up for more information. Please try to be as detailed as possible in your report. What is your environment? What steps will reproduce the issue? What browser(s) and OS experience the problem? Do other browsers show the bug differently? What would you expect to be the outcome? All these details will help people to fix any potential bugs. Just make sure to fill out the issue template.

### Feature Requests

We like feature requests but make sure that it can be seen within the goals of the project and not just something you need individually. Also you should try and give as much examples and details about the new feature as possible. If you are requesting a component from the [Material design guidelines](https://material.io/guidelines/), make sure to include a link to the component.

### Code Examples

- Issues without a [Codepen](#code-examples) (where applicable) will be closed or ignored.

- Use this [Codepen](http://codepen.io/Dogfalo/pen/xbzPQV) to illustrate your problem.

## Pull requests

Good pull requests - patches, improvements, new features - are a fantastic help. Thanks for taking the time to contribute.

\*\*Please ask first\*\* before working on any significant pull request, otherwise you risk spending a lot of time working on something that the project's developers might not want to merge into the project.

\*\*Do not edit `materialize.css`, or `materialize.js`

directly!\*\* Those files are automatically generated. You should edit the

source files in [`/materialize/sass/`](https://github.com/Dogfalo/materialize/tree/master/sass)

and/or [`/materialize/js/`](https://github.com/Dogfalo/materialize/tree/master/js) instead.

### Documentation

When contributing to Materialize's documentation, you should edit the documentation source files in

[the `/materialize/jade/page-contents/` directory of the `master` branch](https://github.com/Dogfalo/materialize/tree/master/jade).

\*\*Do not edit the `gh-pages` branch.\*\* That branch is generated from the documentation source files and is managed separately by the Materialize maintainers.

### Submitting Your Pull Request

Adhering to the following process is the best way to get your work included in the project:

1. [Fork](https://help.github.com/fork-a-repo/) the project, clone your fork,

and configure the remotes:

```bash

# Clone your fork of the repo into the current directory

git clone https://github.com/<your-username>/materialize.git

# Navigate to the newly cloned directory

cd materialize

# Assign the original repo to a remote called "upstream"

git remote add upstream https://github.com/Dogfalo/materialize.git

```

2. If you cloned a while ago, get the latest changes from upstream:

```bash

git checkout master

git pull upstream master

```

3. Create a new topic branch (off the main project development branch) to contain your feature, change, or fix:

```bash

git checkout -b <topic-branch-name>

```

4. Commit your changes in logical chunks with messages written in english. Please adhere to these [git commit message guidelines](http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html) or your code is unlikely be merged into the main project.

5. Locally merge (or rebase) the upstream development branch into your topic branch:

```bash

git pull [--rebase] upstream master

```

6. Push your topic branch up to your fork:

```bash

git push origin <topic-branch-name>

```

7. [Open a Pull Request](https://help.github.com/articles/using-pull-requests/) with a clear title and description against the `master` branch. Reference any open issue in the description so it is automatically linked. Try and keep your commit history clean and concise. Once you submit your pull request, [Travis CI](https://travis-ci.org/Dogfalo/materialize) will automatically run your tests and will show a checkmark to show that all the tests have passed. Once this is done, well review your tests and code and make comments if there are issues or things we think could be improved. Then once everything looks good well merge the code in!

## Translations

If you want to help us translate the documentation into other languages, please send us an email at materializeframework@gmail.com telling us which language team you want to join. We use [Transifex](https://www.transifex.com) as our localization platform and we will send you an invite there.

## Jasmine Testing Guide:

\*\*References:\*\*

- [Jasmine Documentation](http://jasmine.github.io/2.0/introduction.html)

- [Grunt Jasmine Plugin](https://github.com/gruntjs/grunt-contrib-jasmine)

- [Example Jasmine Tests](https://github.com/Dogfalo/materialize/tree/master/tests/spec)

- [Travis CI](https://travis-ci.org/Dogfalo/materialize)

Before you start, make sure you install grunt and all its dependencies. To verify you have all the correct dependencies you can run `grunt travis` and it will run the tests. If you get an errors and have not made any changes, it means you have not installed the proper dependencies.

Materialize uses Jasmine as the testing framework. We also include a jQuery library which allows you to write tests using jQuery syntax.

### Starting Out

First to familiarize yourself with how the tests are structured, you can take a look inside the `tests/` directory. Each component should have its own folder. Follow the file-naming conventions that are used in the existing tests.

Before writing tests, make sure you are working off of a clean git branch of your fork. This will greatly simplify the Pull Request process.

### Writing Tests

Before writing tests, make sure you understand what the expected-behavior of the component actually is. Reading over the component code and documentation will greatly aid you in this regard.

Use `describe` blocks to section disparate portions of tests and `it` blocks inside those to further break up tests into features. Inside `it` blocks, you can have multiple except statements. As a general testing principle, be sure to try and test both the case and its inverse to lessen the chance for false positives.

Example:

```javascript

expect(toast.first('span').text()).toBe('I am toast content');

expect(toast.first('span').text()).not.toBe('I am toast');

```

You can use beforeEach, and afterEach in either block to designate code that will execute before or after each item. This is useful if you need to setup some scenario for each test, or reset some things after each test.

When writing expect statements (Jasmines form of assert), it is very important to write an expected behavior string so in the event of an error, the test that failed is very clear.

Example:

```javascript

expect(toast.length).toBe(0, 'because toast should be removed by now');

```

When this expect statement fails it will list the reason as because toast should be removed by now.

Because our components are very front end heavy, familiarize yourself with jQuery ways of interacting with the dom and our components. You can use methods like [trigger](http://api.jquery.com/trigger/), to simulate certain events like the user clicking a button.

We also understand that testing CSS properties is pretty tough so youll have to be creative when writing good tests that ensure the styling is still working. Try and cover as many cases as you can but dont worry if there are some edge cases. You can add comments describing some problematic edge cases in TODOs so we know about them.

### Useful Jasmine Tips

1. To only run a specific spec at a time, to avoid wasting your time running all our other tests, you can set the flag `--filter`. For example:

```bash

grunt travis --filter=tabs

```

This would only run specs with tabs in its name.

2. If you need a timeout in your test (waiting for some animation or action to be executed) you need to use the done callback. In your `it()` behavior function set done as an argument to your anonymous function. Then you can use javascripts window `setTimeout`s normally. And when you want the test to finish just call the `done()` function. For example:

```javascript

it ('should wait for a timeout', function(done) {

// Execute action

timeout(setTimeout(function() {

// Wait a second

// Test for result

done();

}, 1000);

});

```

\*\*Note:\*\* If you add done as a callback, and you dont call the `done()` function, it will stall forever and error after a max limit of around 5 seconds.

## License

\*\*IMPORTANT\*\*: By contributing your code, you agree to allow the project owners to license your work under the terms of the [MIT License](LICENSE).