# Contributing to Material Components Web (MDC Web)

We'd love for you to contribute and make Material Components for the web even better than it is today!

Here are the guidelines we'd like you to follow:

- [Contributing to Material Components Web (MDC Web)](#contributing-to-material-components-web-mdc-web)

- [General Contributing Guidelines](#general-contributing-guidelines)

- [Development Process](#development-process)

- [Setting up your development environment](#setting-up-your-development-environment)

- [Building Components](#building-components)

- [Running development server](#running-development-server)

- [App Engine development server](#app-engine-development-server)

- [Building MDC Web](#building-mdc-web)

- [Linting / Testing / Coverage Enforcement](#linting--testing--coverage-enforcement)

- [Running Tests across browsers](#running-tests-across-browsers)

- [Coding Style](#coding-style)

- [Submitting Pull Requests](#submitting-pull-requests)

## General Contributing Guidelines

The Material Components contributing policies and procedures can be found in the main Material Components documentation repository’s [contributing page](https://github.com/material-components/material-components/blob/develop/CONTRIBUTING.md).

## Development Process

We strive to make developing Material Components Web as frictionless as possible, both for ourselves and for our community. This section should get you up and running working on the MDC Web codebase. Before beginning development, you may want to read through our overview on [architecture and best practices](./docs/code) to get a better understanding of our overall structure.

### Setting up your development environment

You'll need a recent version of [nodejs](https://nodejs.org/en/) to work on MDC Web. We [test our builds](https://travis-ci.com/material-components/material-components-web) using both the latest and LTS node versions, so use of one of those is recommended. You can use [nvm](https://github.com/creationix/nvm) to easily install and manage different versions of node on your system.

> \*\*NOTE\*\*: If you expect to commit updated or new dependencies, please ensure you are using npm 5, which will

> also update `package-lock.json` correctly when you install or upgrade packages.

Once node is installed, simply clone our repo (or your fork of it) and run `npm install`

```

git clone git@github.com:material-components/material-components-web.git # or a path to your fork

cd material-components-web && npm i

```

### Building Components

Each component requires the following items in order to be complete:

- An \*\*Engineering Outline Document\*\*, which should be linked to in a comment on the GitHub issue

where we're tracking the component. This way, the core team can review and sign off on the

outline doc. Outline docs should be signed off on \_before\_ submitting a PR.

Please [copy this google doc template](https://docs.google.com/document/d/1Kybm7XJDTy0KUcuMaw5bzirQNBsDqCPCae8U\_ag\_a1k/edit?usp=sharing) in order to make your outline.

We have found that enforcing an eng outline doc has allowed us to speed up development by

offering more informed feedback on component implementations. This results in components that

take into account all of the concepts MDC Web components should account for (RTL, a11y,

etc.) before they even reach the PR stage, meaning faster review and merge times :smile:.

- A \*\*foundation class\*\* which is integrated into actual components

- A \*\*component class\*\* using vanilla JS + SCSS

- A \*\*README.md\*\* in its subdir which contains developer documentation on the component, including usage.

- A \*\*set of unit tests\*\* within `packages/<mdc-component>/test/` with adequate coverage (which we enforce automatically).

You can find much more information with respect to building components within our [authoring components guide](./docs/authoring-components.md)

### Building MDC Web

```

npm run build # Cleans out build/ and builds unminified files for each MDC Web package

npm run build:min # Builds minified files for each MDC Web package

npm run dist # Runs both of the above commands sequentially

npm run build:demos # Cleans out build/ and builds demo CSS/JS files, e.g. for deploying to App Engine

```

### Linting / Testing / Coverage Enforcement

```

npm run lint:js # Lints javascript using eslint

npm run lint:css # Lints (S)CSS using stylelint

npm run lint # Runs both of the above commands in parallel

npm run fix:js # Runs eslint with the --fix option enabled

npm run fix:css # Runs stylefmt, which helps fix simple stylelint errors

npm run fix # Runs both of the above commands in parallel

npm run test:watch # Runs karma on Chrome, re-running when source files change

npm test # Lints all files, runs karma, runs typescript tests, and then runs coverage enforcement checks.

npm run test:unit # Only runs the karma tests

```

#### Running Tests across browsers

If you're making big changes or developing new components, we encourage you to be a good citizen and test your changes across browsers! A super simple way to do this is to use [sauce labs](https://saucelabs.com/), which is how we test our collaborator PRs on TravisCI:

1. [Sign up](https://saucelabs.com/beta/signup) for a sauce labs account (choose "Open Sauce" as your selected plan; [it's free](https://saucelabs.com/opensauce/)!)

2. [Download sauce connect](https://wiki.saucelabs.com/display/DOCS/Sauce+Connect+Proxy) for your OS and make sure that the `bin` folder in the downloaded zip is somewhere on your `$PATH`.

3. Navigate to your dashboard, scroll down to where it says "Access Key", and click "Show"

4. Enter your password when prompted

5. Copy your access key

6. Run `SAUCE\_USERNAME=<your-saucelabs-username> SAUCE\_ACCESS\_KEY=<your-saucelabs-access-key> npm test`

This will have karma run our unit tests across all browsers we support, and ensure your changes will not introduce regressions.

Alternatively, you can run `npm run test:watch` and manually open browsers / use VMs / use emulators to test your changes.

### Coding Style

Our entire coding style is enforced automatically through the use of linters. Check out our [eslint config](https://github.com/material-components/material-components-web/blob/master/.eslintrc.yaml) (heavily influenced by [Google's Javascript Styleguide][js-style-guide]) as well as our [stylelint config][css-style-guide] (heavily annotated, with justifications for each rule) for further details.

### Submitting Pull Requests

When submitting PRs, make sure you're following our commit message conventions (which are the same as [angular's](https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#commits)); our `commit-msg` hook should automatically enforce this. We also support [commitizen](https://www.npmjs.com/package/commitizen), which you can

use to auto-format commit messages for you.

When submitting PRs for large changes, be sure to include an adequate background in the description

so that reviewers of the PR know what the changes entail at a high-level, the motivations for making

these changes, and what they affect.

If you've done some experimental work on your branch/fork and committed these via `git commit --no-verify`, you can rebase them into one correctly-formatted commit before submitting.

Finally, it helps to make sure that your branch/fork is up to date with what's currently on master. You can ensure this by running `git pull --rebase origin master` on your branch.

> \*\*NOTE\*\*: Please do \_not merge\_ master into your branch. \_Always\_ `pull --rebase` instead. This ensures a linear history by always putting the work you've done after the work that's already on master, regardless of the date in which those commits were made.

[js-style-guide]: https://google.github.io/styleguide/jsguide.html

[css-style-guide]: https://github.com/material-components/material-components-web/blob/master/.stylelintrc.yaml