# Contributing to Saladict

:+1::tada: First off, thanks for taking the time to contribute! :tada::+1:

## How to Contribute

:warning: Unless it is a small hot fix, before you write any code and get your hands dirty, please open an issue or make a WIP pull request to elaborate what you are trying to do and how you are going to implement it. Just to make sure we are on the same page and nobody's time and effort are wasted.

- Read [How to get started](#how-to-get-started).

- Follow [code style](#code-style) and [commit style](#commit-style).

- Before submit, run [test](#testing) and [build](#building) locally. Or leave it to CI.

## How to get started

```bash

git clone git@github.com:crimx/ext-saladict.git

cd ext-saladict

yarn install

yarn pdf

```

## UI Tweaking

Run `yarn fixtures` to download fixtures(only need to run once).

Run `yarn storybook` to view all the components.

Run `yarn start --wextentry [entry id]` to view a certain entry with WDS in a fake WebExtension environment.

## Testing

Run `yarn test` to run Jest. Supports all the Jest [options](https://jestjs.io/docs/en/cli).

## Building

Run `yarn build` to start a full build.

Toggle:

- `--debug`: Remove compression and generate sourcemaps.

- `--analyze`: Show detailed Webpack bundle analyzer.

## How to add a dictionary

For safety and maintainability reason, Saladict will not support adding dictionaries on the fly. All dictionaries must be merged to this project via pull requests.

If dictionary implementation makes use of private API please move it to an independent project, release on NPM, then import it to Saladict.

1. Create a directory at [`src/components/dictionaries/`](./src/components/dictionaries/), with the name of the dict ID.

1. Use any existing dictionary as guidance, e.g. [Bing](./src/components/dictionaries/bing). Copy files to the new directory.

1. Replace the favicon with a new LOGO.

1. Edit `config.ts` to change default options. See the `DictItem` type and explanation for more details. Register the dictionary in [app config](./src/app-config/dicts.ts) so that TypeScript generates the correct typings. Dict ID \*\*MUST\*\* follow alphabetical order.

1. Update `\_locales.json` with the new dictionary name. Add locales for options, if any.

1. `engine.ts` \*\*MUST\*\* export at least two functions:

1. `getSrcPage` function which is responsible for generating source page url base on search text and app config. Source page url is opened when user clicks the dictionary title.

1. `search` function which is responsible for fetching, parsing and returning dictionary results. See the typings for more detail.

- Extracting information from a webpage \*\*MUST\*\* use helper functions in [../helpers.ts](./components/dictionaries/helpers.ts) for data cleansing.

- If the dictionary supports pronunciation:

1. Register the ID at [`config.autopron`](https://github.com/crimx/ext-saladict/blob/a88cfed84129418b65914351ca14b86d7b1b758b/src/app-config/index.ts#L202-L223).

1. Include an [`audio`](https://github.com/crimx/ext-saladict/blob/a88cfed84129418b65914351ca14b86d7b1b758b/src/typings/server.ts#L5-L9) field in the object which search engine returns.

1. Other exported functions can be called from `View.tsx` via `'DICT\_ENGINE\_METHOD'` message channel. See `src/typings/message` for typing details and search `DICT\_ENGINE\_METHOD` project-wise for examples. Messages \*\*MUST\*\* be sent via `message` from `'@/\_helpers/browser-api'` instead of the native `sendMessage` function.

1. Search result will ultimately be passed to a React PureComponent in `View.tsx`, which \*\*SHOULD\*\* be a dumb component that renders the result accordingly.

1. Selectors in `\_style.scss` \*\*SHOULD\*\* follow [ECSS](http://ecss.io/chapter5.html#anatomy-of-the-ecss-naming-convention)-ish naming convention.

### Develop the dictionary UI live

To develop the component in Storybook we need to intercept http requests from dictionary engines and replace with the downloaded results.

1. Add `fixtures.js` at `test/specs/components/dictionaries/[dictID]`.

- See other dictionaries for example.

- You can offer url or axios config (See `mojidict` dictionary). All results from previous requests will be passed to the next request as array.

1. Run `yarn fixtures` to download fixtures.

1. Edit `test/specs/components/dictionaries/[dictID]/request.mock.ts`. It will intercept requests and return the downloaded fixtures.

1. Run `yarn storybook`.

### Add Testing

1. Add `[dictID]/engine.spec.ts` to test the engine.

## Code Style

This project follows the TypeScript variation of [Standard](https://standardjs.com) JavaScript code style.

If you are using IDEs like VSCode, make sure \*eslint\* and \*prettier\* plugins are installed. Or you can just run [building command](#building) to perform a TypeScript full check.

## Commit Style

This project follows [conventional](https://conventionalcommits.org/) commit style.

You can run `yarn commit` and follow the instructions, or use [VSCode Conventional Commits](https://github.com/vivaxy/vscode-conventional-commits) extension in VSCode.