# CONTRIBUTING

This guide is for anyone who wants to contribute to the Webiny project.

## Working on an issue

> IMPORTANT: Before working on a PR, please open an issue and discuss your intended changes with the maintainers. They may provide invaluable info and point you in the right direction to get the most out of your contribution.

### Branches

`master` is the main branch on which we develop. All `issue` branches should be branched from `master` branch, unless you're working on an issue that belongs to one of our [projects](https://github.com/webiny/webiny-js/projects). In that case, a project branch will be specified in the project board. If you're not sure about the branch, don't hesitate to contact us.

### Workflow

- create an issue branch

- commit your changes (please follow our [commit message format](#commit-messages) rules)

- open a PR

- try to keep your PRs small in scope (try to only work on 1 issue in a single PR)

- you can add as many commits as you wish to your PR

### Commit messages

We are strongly following the [Conventional Commits specification](https://www.conventionalcommits.org/).

You can use `yarn commit` to commit via `commitizen` or you can commit manually, or via IDE. Just make sure your commit messages are properly written.

#### ?? IMPORTANT NOTE

Always follow the specification (even though later down the road, your PR might get merged by squashing all commits). Otherwise, the PR will [automatically get rejected](https://github.com/webiny/action-conventional-commits), and you will have to change your commit messages, which can only be done by creating a new PR.

##### Additional explanation

When merging larger PRs, squashing all commits into a single one often doesn't make sense, and in those cases, we are doing a regular merge - without squash. And when that is about to happen, it's important that all commit messages were properly written.

### Pull Requests (PRs)

When submitting a PR, please do not leave the PR title / body incomplete or empty. We are an open-source project, which means a lot of people will be contributing to it (hopefully ??), so we have to maintain a certain level of professionalism.

Try to make your PRs as informative as possible:

- link the related issue you are resolving

- add some info about how you've resolved the issue

- if applicable, attach an image as well

For example:

![Snipaste\_2020-04-19\_20-10-44](https://user-images.githubusercontent.com/5121148/79717886-20ebe280-82db-11ea-9c23-f46e5ab01724.png)

There's nothing nicer than a well-written PR. ?

## Repo overview

Once you clone the repository, you will have a monorepo which consists of a bunch of different packages, located in the `/packages` directory.

- `app-\*` packages are used only in React apps

- `api-\*` are only used when building API services

- `handler-\*` are utility packages to create serverless function handlers

- `serverless-\*` are packages containing infrastructure components used to deploy your infrastructure

In the root of the project, you'll find the `api` and `apps` folders. These are the [stacks](https://docs.webiny.com/docs/deep-dive/stacks/) we use as our sandbox for development of Webiny packages.

The setup of the repo is identical to the one created by `create-webiny-project`.

## Prerequisites

1. Node `10.14` or higher (to manage your Node versions we recommend [n](https://www.npmjs.com/package/n) for OSX/Linux, and [nvm-windows](https://github.com/coreybutler/nvm-windows) for Windows)

2. `yarn 1.0` or higher (because our project setup uses workspaces). Yarn `v2` is not yet supported.

If you don't already have `yarn`, visit [yarnpkg.com](https://yarnpkg.com/en/docs/install) to install it.

3. A verified AWS account with an [IAM user for programmatic usage](https://www.youtube.com/watch?v=tgb\_MRVylWw)

4. Webiny uses MongoDB as its go-to database, so you'll need to have one ready. We recommend [Mongo Atlas](https://docs.atlas.mongodb.com/getting-started/) (there is a free tier for developers, so don't worry about having to pay for anything).

> IMPORTANT: it's important to give the outside world access to your database because the database will be accessed from your cloud functions, thus you'll never have a fixed IP address. See the [Whitelist Your Connection IP Address](https://docs.atlas.mongodb.com/getting-started/#whitelist-your-connection-ip-address). Make sure you add a `0.0.0.0/0` entry.

> The `MONGODB\_SERVER` value should be in the format of a MongoDB connection string such as:

> `mongodb+srv://{YOUR\_USERNAME}:{YOUR\_PASSWORD}@someclustername.mongodb.net`.

> WINDOWS USERS: make sure you have installed [Visual C++ Redistributable for Visual Studio 2015](https://www.microsoft.com/en-in/download/details.aspx?id=48145). This is required to run tests using Jest [plugin for in-memory MongoDB server](https://github.com/shelfio/jest-mongodb).

> WINDOWS USERS: it's best to use `git-bash` as a terminal to work with Webiny as `cmd` won't work. If you have `Git` installed, most likely you already have the `git-bash` installed. If you're using VSCode IDE, you will be able to easily switch to the `bash` terminal. Alternatively you can install the [cmder](https://cmder.net/) terminal emulator.

## Local setup

1. Fork and clone the repo

2. Install all dependencies:

```

yarn

```

3. Run `yarn setup-repo`. This will setup all the necessary environment config files and build all packages to generate `dist` folders and TS declarations. You need to manually update the DB connection string, edit your `.env.json` file.

4. Configure your MongoDB connection data in `.env.json`. See https://docs.webiny.com/docs/get-started/quick-start/#2-setup-database-connection for more details.

5. Deploy you API to use with local React apps by running `yarn webiny deploy api --env=local` from the project root directory. Once deployed, it will automatically update you React apps' `.env.json` files with the necessary variables.

> IMPORTANT: `webiny` should be run from the root of the Webiny project, and since `api` and `apps` folders are a `sandbox` present in the project root directory, this is the place to run your `webiny` commands from.

6. Begin working on React apps by navigating to `apps/{admin|site}` and run `yarn start`. React apps are regular `create-react-app` apps, slightly modified, but all the CRA rules apply.

7. Run `watch` on packages you are working on so that your changes are automatically built into the corresponding `dist` folder. React app build will automatically rebuild and hot-reload changes that happen in the `dist` folder of all related packages.

The easiest way to run a watch is by running `lerna run watch --scope=your-scope --stream --parallel`. For more details visit the [official lerna filtering docs](https://github.com/lerna/lerna/tree/master/core/filter-options).

## Tests

You can find examples of tests in some of the utility packages (`validation`, `i18n`, `plugins`).

`api-files` and `api-headless-cms` contain examples of testing your GraphQL API.

We'll be strongly focusing on tests in the near future, and of course contributions of tests are most welcome :)

### [Cypress](https://www.cypress.io/) tests

#### Prerequisites

Before running the tests, make sure you have a working API and app deployed to the actual cloud, since Cypress tests should be ran against a real environment, in which the app will live. More on this at the end of this section.

#### Configuration

Once you have a working API and app deployed to the cloud, make sure to open the `cypress.json` in the project root, and configure all of the variables.

Tests that are testing different sections of the Admin app also require the `DEFAULT\_ADMIN\_USER\_USERNAME` and `DEFAULT\_ADMIN\_USER\_PASSWORD` values, which represent the credentials of the default full-access admin account (set in the Admin app, in the initial installation process).

#### Opening the Cypress app

Once you've configured all of the variables, you can run the following command in the project root: `cypress open`. This will open the Cypress app, which will enable you to choose the test you wish to execute. You can run only your test, which is ideal if you're in the process of creating it.

#### Should I run the tests against a local development server or a project deployed to the cloud?

In general, Cypress tests should be ran against a project deployed into the cloud, mainly because of the existing tests that

are making assertions related to the server side rendering (SSR) and CDN cache invalidations, which is not active in local development.

The only problem with this approach is that if you're in process of creating a new test, and you need to change something in the UI in order to make it easier to test (e.g. adding a "data-testid" attribute to a HTML element), you'll need to redeploy the app, which might get a bit frustrating if your making a lot of changes (since a single deploy can take up to 180s).

But, if your test doesn't involve assertions related to SSR and CDN cache invalidation, while creating the test, you can actually run it against a local development server (set `SITE\_URL` variable to e.g. "http://localhost:3001"). This way you'll be able to see your changes in the browser much faster, and get back to your test faster as well.

#### Where are tests located?

All of the tests can be found in the `cypress/integration` folder (in the project root). In there, you will find just a single `admin` folder, because at the moment we only have tests for the Admin app and various modules introduced by other Webiny apps (Page Builder, Form Builder, Security, ...).

Follow the same structure if you're about to add a new test. Also make sure to check other tests before creating one.

#### How to test `site` app in the cloud?

When deployed to the cloud, the `site` app (which basically represents the public-facing website) is using SSR and CDN caching in order to improve SEO compatibility and drastically speed up the site, respectively.

The problem occurs when you make changes in the Admin, and you want to test that these changes are actually visible on the website. Because of the CDN cache, changes won't be immediately there, but only after 5-10 seconds. In some cases it can take even longer for the page to refresh.

The initial "quick" solution was to just use `.wait(30000)` commands in order to wait for the CDN cache to be invalidated. But as you might've noticed, this isn't very effective, since in some cases CDN could be invalidate way before 30 seconds. On the other hand, sometimes 30 seconds wasn't long enough, and the tests would continue making assertions on the old page content, which would result in a failed test.

That's why we've created a custom `visitAndReloadOnceInvalidated` Cypress command. The following code shows a usage example:

```js

cy.findByText("Save something")

.click()

.visitAndReloadOnceInvalidated(Cypress.env("SITE\_URL"))

.continueTestingAsUsual();

```

The `visitAndReloadOnceInvalidated` command will immediately visit the URL you're trying to test and will continuously refresh the page until the change was detected, after which the next assertions will start to get executed.

The page will be refreshed every ~3 seconds for 10 times. If there are no changes after that, the command will throw an error, and the test will fail.