# Setup Guide

This document describes how to set up ESLint and ESLint-plugin-Meteor in Meteor projects. *It must have steps for Meteor projects before 1.3 and with 1.3. It should further show how to use only selected rules (or link to the page of the ESLint documentation)*

This guide assumes you have npm installed.

## Setup

If you don't have a `package.json` at the root of your Meteor project, create one with `npm init --yes`. Next, add `private: true` to your package (to avoid some warnings and prevent publishing your project to npm's registry accidentally).

Now, install ESLint and ESLint-plugin-Meteor as development dependencies:

```
$ npm install eslint eslint-plugin-meteor --save-dev
```

Next, an ESLint configuration needs to be created to enable some rules. Create a file called `.eslintrc.json` at the root of your project.

A minimal configuration should look like this:

```
{
  "parserOptions": {
    "ecmaVersion": 6,
    "sourceType": "module",
    "ecmaFeatures": {
      "jsx": true
    }
  },
  "plugins": ["meteor"],
  "extends": ["plugin:meteor/recommended"]
}
```

And that's it 🎉!

More information on setting up ESLint can be found here.

An article with detailed setup instructions specifically for Meteor projects can be found here.

## Tips

Here are some more tips to further improve the setup.

### Add environments

An environment tells ESLint about defined globals. Since Meteor code can run in the browser and on the server, it's wise to add `browser` and `node`. As Meteor supports ES2015, `es6` should be added as well. And of course the `meteor` environment itself. Since Meteor 1.3 applications can use modules, they should be enabled for ESLint as well.

```
{
  /* ... */
  "parserOptions": {
    "ecmaVersion": 6,
    "sourceType": "module"
  },
  "env": {
    "es6": true,
    "browser": true,
    "node": true,
    "meteor": true
  },
  /* ... */
}
```

### Collections and globals

ESLint needs to know about globals defined in your application. Add the globals key to `.eslintrc.json`:

```
{
  /* ... */
  "globals": {
    "MyCollection": true,
    "moment": false
  },
  /* ... */
}
```

Here, you can define all globals your application uses. This is also the place to add globals provided through packages from Atmosphere. The boolean values tell ESLint whether it is okay for your application code to overwrite these globals ( `true` ) or not ( `false` ).

### Usage with React

If you are using React, you should:

- enable JSX syntax (see: ESLint configuration documentation)
- use ESLint-plugin-React (see: eslint-plugin-react)

## ESLint-config-airbnb

Use a rule preset like eslint-config-airbnb. It has lots of well-thought-out rules.

### Using YAML instead

ESLint supports different formats in which the configuration can be specified. If `.eslintrc.json` is renamed to `.eslint.yaml` then the full configuration can be written like this:

```
---
  root: true
```

```yaml
env:
  es6: true
  browser: true
  node: true
  meteor: true

parserOptions:
  ecmaVersion: 6
  sourceType: module
  ecmaFeatures:
    jsx: true

plugins:
  - meteor

extends:
  - airbnb/base
  - plugin:meteor/recommended

globals:
  # Collections
  MyCollection: true
  # ..

  # Packages
  moment: false # exported by momentjs:moment
  # ..
```