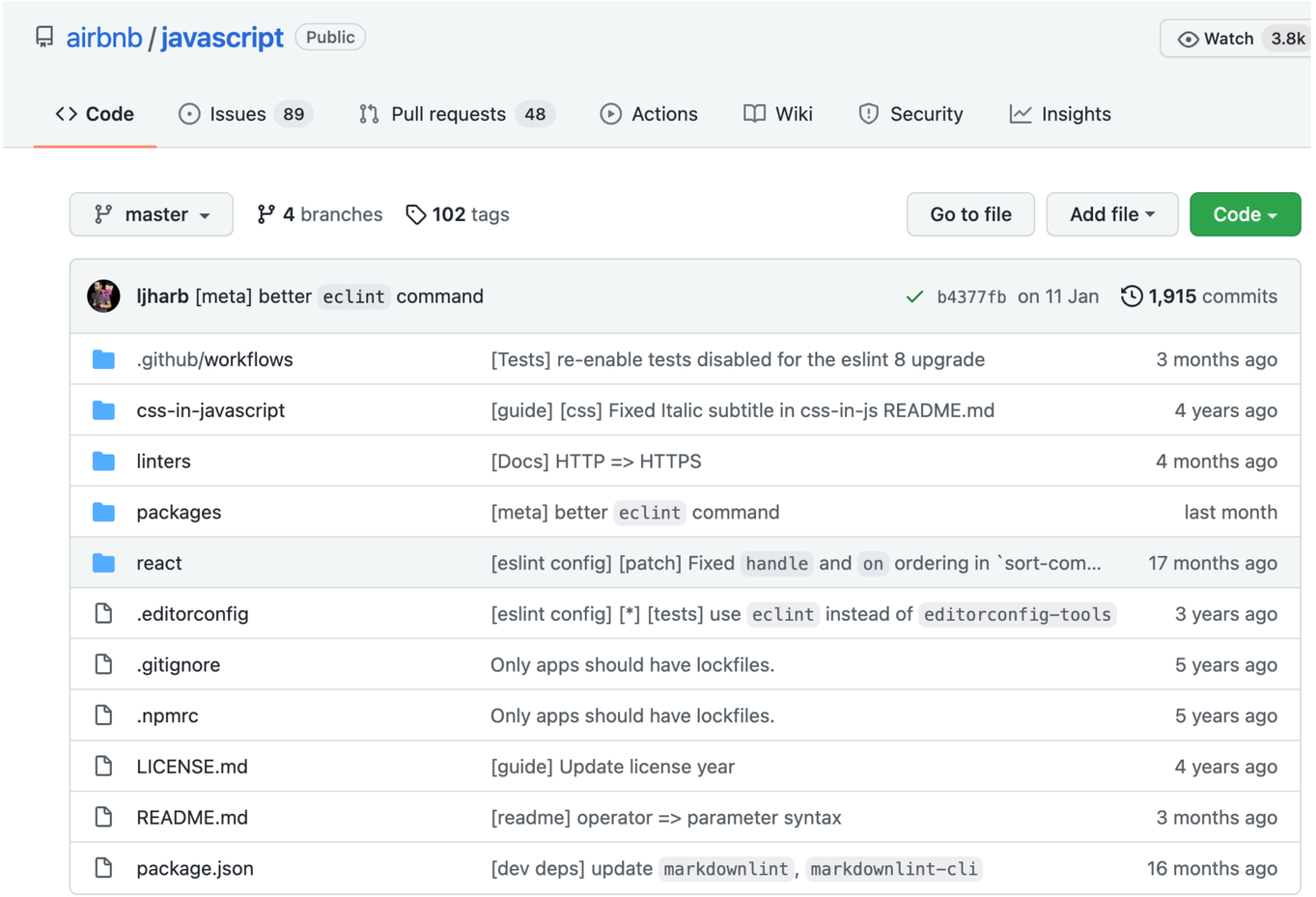
Airbnb JS Style Guide

Naming Conventions
Comments

Full Stack Web Development

Airbnb JavaScript Style Guide() {

“A mostly reasonable approach to JavaScript”



airbnb / javascript Public

Watch 3.8k

Code | Issues 89 | Pull requests 48 | Actions | Wiki | Security | Insights

master | 4 branches | 102 tags | Go to file | Add file | Code

ljharb [meta] better `eclint` command — b4377fb on 11 Jan — 1,915 commits

.github/workflows	[Tests] re-enable tests disabled for the eslint 8 upgrade	3 months ago
css-in-javascript	[guide] [css] Fixed Italic subtitle in css-in-js README.md	4 years ago
linters	[Docs] HTTP => HTTPS	4 months ago
packages	[meta] better `eclint` command	last month
react	[eslint config] [patch] Fixed `handle` and `on` ordering in `sort-com...	17 months ago
.editorconfig	[eslint config] [*] [tests] use `eclint` instead of `editorconfig-tools`	3 years ago
.gitignore	Only apps should have lockfiles.	5 years ago
.npmrc	Only apps should have lockfiles.	5 years ago
LICENSE.md	[guide] Update license year	4 years ago
README.md	[readme] operator => parameter syntax	3 months ago
package.json	[dev deps] update `markdownlint`, `markdownlint-cli`	16 months ago

Customisations

```
{
  "env": {
    "node": true,
    "es2021": true
  },
  "extends": [
    "airbnb-base",
    "prettier"
  ],
  "parserOptions": {
    "ecmaVersion": "latest",
    "sourceType": "module"
  },
  "rules": {
    "quotes": [
      "error",
      "double"
    ],
    "import/extensions": "off",
    "import/prefer-default-export": "off",
    "object-shorthand": "off",
    "no-unused-vars": "off",
    "no-underscore-dangle": "off",
    "no-param-reassign": "off",
    "no-undef": "off"
  }
}
```

.eslintrc.json

- Playtime uses a subset of the AirBnB guide
- The subset is defined by customising eslint, durning off specific rules.
- + tuning prettier rules

```
{
  "trailingComma": "es5",
  "tabWidth": 2,
  "semi": true,
  "singleQuote": false,
  "printWidth": 180
}
```

.prettier.json

Naming Conventions

Comments

Whitespace

Commas

Semicolons

References

Variables

Arrays

Strings

Objects

Properties

Destructuring

Blocks

Control Statements

Functions

Arrow Functions

Modules

Naming Conventions

Comments

Naming Conventions

Naming Conventions

- Avoid single letter names. Be descriptive with your naming. eslint: **id-length**

```
// bad
function q() {
  // ...
}

// good
function query() {
  // ...
}
```

- Use camelCase when naming objects, functions, and instances. eslint: **camelcase**

```
// bad
const OBJEcttssss = {};
const this_is_my_object = {};
function c() {}

// good
const thisIsMyObject = {};
function thisIsMyFunction() {}
```

Naming Conventions

- Use PascalCase only when naming constructors or classes. eslint: **new-cap**

```
// bad
function user(options) {
  this.name = options.name;
}

const bad = new user({
  name: "nope",
});

// good
class User {
  constructor(options) {
    this.name = options.name;
  }
}

const good = new User({
  name: "yup",
});
```

Naming Conventions

- Do not use trailing or leading underscores. eslint: **no-underscore-dangle**

> *"Why? JavaScript does not have the concept of privacy in terms of properties or methods. Although a leading underscore is a common convention to mean "private", in fact, these properties are fully public, and as such, are part of your public API contract. This convention might lead developers to wrongly think that a change won't count as breaking, or that tests aren't needed. tl;dr: if you want something to be "private", it must not be observably present."*

```
// bad
this.__firstName__ = "Panda";
this.firstName_ = "Panda";
this._firstName = "Panda";

// good
this.firstName = "Panda";

// good, in environments where WeakMaps are available
// see https://kangax.github.io/compat-table/es6/#test-WeakMap
const firstNames = new WeakMap();
firstNames.set(this, "Panda");
```

Naming Conventions

- Don’t save references to `this`. Use arrow functions or Function#bind.

```
// bad
function foo() {
  const self = this;
  return function () {
    console.log(self);
  };
}

// bad
function foo() {
  const that = this;
  return function () {
    console.log(that);
  };
}

// good
function foo() {
  return () => {
    console.log(this);
  };
}
```

Naming Conventions

- A base filename should exactly match the name of its default export.

```
// file 1 contents
class CheckBox {
  // ...
}
export default CheckBox;

// file 2 contents
export default function fortyTwo() { return 42; }

// file 3 contents
export default function insideDirectory() {}

// in some other file
// bad
import CheckBox from "./checkBox"; // PascalCase import/export, camelCase filename
import FortyTwo from "./FortyTwo"; // PascalCase import/filename, camelCase export
import InsideDirectory from "./InsideDirectory"; // PascalCase import/filename, camelCase export

// bad
import CheckBox from "./check_box"; // PascalCase import/export, snake_case filename
import forty_two from "./forty_two"; // snake_case import/filename, camelCase export
import inside_directory from "./inside_directory"; // snake_case import, camelCase export
import index from "./inside_directory/index"; // requiring the index file explicitly
import insideDirectory from "./insideDirectory/index"; // requiring the index file explicitly

// good
import CheckBox from "./CheckBox"; // PascalCase export/import/filename
import fortyTwo from "./fortyTwo"; // camelCase export/import/filename
import insideDirectory from "./insideDirectory"; // camelCase export/import/directory name/implicit "index"
// ^ supports both insideDirectory.js and insideDirectory/index.js
```

Naming Conventions

- Use camelCase when you export-default a function. Your filename should be identical to your function's name.

```
function makeStyleGuide() {
  // ...
}

export default makeStyleGuide;
```

- Use PascalCase when you export a constructor / class / singleton / function library / bare object.

```
const AirbnbStyleGuide = {
  es6: {
  },
};

export default AirbnbStyleGuide;
```

Naming Conventions

- Acronyms and initialisms should always be all uppercased, or all lowercased.

> *"Why? Names are for readability, not to appease a computer algorithm."*

```
// bad
import SmsContainer from "./containers/SmsContainer";

// bad
const HttpRequests = [
  // ...
];

// good
import SMSContainer from "./containers/SMSContainer";

// good
const HTTPRequests = [
  // ...
];

// also good
const httpRequests = [
  // ...
];

// best
import TextMessageContainer from "./containers/TextMessageContainer";

// best
const requests = [
  // ...
];
```

Naming Conventions

- You may optionally uppercase a constant only if it (1) is exported, (2) is a `const` (it can not be reassigned), and (3) the programmer can trust it (and its nested properties) to never change.

> *"Why? This is an additional tool to assist in situations where the programmer would be unsure if a variable might ever change. UPPERCASE_VARIABLES are letting the programmer know that they can trust the variable (and its properties) not to change."*
>
> - *What about all `const` variables? - This is unnecessary, so uppercasing should not be used for constants within a file. It should be used for exported constants however.*
> - *What about exported objects? - Uppercase at the top level of export (e.g. `EXPORTED_OBJECT.key`) and maintain that all nested properties do not change.*

```
// bad
const PRIVATE_VARIABLE = "should not be unnecessarily uppercased within a file";

// bad
export const THING_TO_BE_CHANGED = "should obviously not be uppercased";

// bad
export let REASSIGNABLE_VARIABLE = "do not use let with uppercase variables";

// ---

// allowed but does not supply semantic value
export const apiKey = "SOMEKEY";

// better in most cases
export const API_KEY = "SOMEKEY";

// ---

// bad - unnecessarily uppercases key while adding no semantic value
export const MAPPING = {
  KEY: "value"
};

// good
export const MAPPING = {
  key: "value"
};
```

Comments

Comments

- Use `/** ... */` for multiline comments.

```
// bad
// make() returns a new element
// based on the passed in tag name
//
// @param {String} tag
// @return {Element} element
function make(tag) {

  // ...

  return element;
}

// good
/**
 * make() returns a new element
 * based on the passed-in tag name
 */
function make(tag) {

  // ...

  return element;
}
```

Comments

- Use `//` for single line comments. Place single line comments on a newline above the subject of the comment. Put an empty line before the comment unless it's on the first line of a block.

```
// bad
const active = true;  // is current tab

// good
// is current tab
const active = true;

// bad
function getType() {
  console.log("fetching type...");
  // set the default type to "no type"
  const type = this.type || "no type";

  return type;
}

// good
function getType() {
  console.log("fetching type...");

  // set the default type to "no type"
  const type = this.type || "no type";

  return type;
}

// also good
function getType() {
  // set the default type to "no type"
  const type = this.type || "no type";

  return type;
}
```

Comments

- Start all comments with a space to make it easier to read. eslint: **spaced-comment**

```
// bad
//is current tab
const active = true;

// good
// is current tab
const active = true;

// bad
/**
 *make() returns a new element
 *based on the passed-in tag name
 */
function make(tag) {

  // ...

  return element;
}

// good
/**
 * make() returns a new element
 * based on the passed-in tag name
 */
function make(tag) {

  // ...

  return element;
}
```

Comments

- Prefixing your comments with `FIXME` or `TODO` helps other developers quickly understand if you're pointing out a problem that needs to be revisited, or if you're suggesting a solution to the problem that needs to be implemented. These are different than regular comments because they are actionable. The actions are `FIXME: -- need to figure this out` or `TODO: -- need to implement`.

- Use `// FIXME:` to annotate problems.

```
class Calculator extends Abacus {
  constructor() {
    super();
    // FIXME: shouldn't use a global here
    total = 0;
  }
}
```

- Use `// TODO:` to annotate solutions to problems.

```
class Calculator extends Abacus {
  constructor() {
    super();
    // TODO: total should be configurable by an options param
    this.total = 0;
  }
}
```

Airbnb JS Style Guide

Naming Conventions
Comments

Full Stack Web Development