JavaScript

Table of contents

General

Try to follow the AirBnB styling guide as far as possible. Sometimes you might have to deviate from the guide; such as when targeting Internet Explorer without a transpiler, you have to use `var` instead of `const` or `let`.

When using a complete toolchain with a transpiler, eslint should be configured to warn about any style violations.

Naming conventions

- Use camelCase when naming objects, functions, and instances.

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

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

- Use PascalCase only when naming constructors or classes.

```
// 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',
});
```

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

```
// 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 = [
  // ...
];
```

Bundling

Always use bundling to some degree in production. Commonly used vendor scripts should always be bundles, but however be aware that bundles do not grow too large. Bundling tools such as webpack provide an efficient way to split bundles when they grow too large.

TODO : bundles do not grow too large - what is the guidelines here?

Example of bundling

Minification

Always minify in production.

TODO : Example and how is minification done

Use minified version when external library is used

jQuery

- Prefix jQuery object variables with a `$`.

```
// bad
const sidebar = $('.sidebar');

// good
const $sidebar = $('.sidebar');

// good
const $sidebarBtn = $('.sidebar-btn');
```

- Cache jQuery lookups.

```
// bad
function setSidebar() {
  $('.sidebar').hide();

  // ...

  $('.sidebar').css({
    'background-color': 'pink',
  });
}

// good
function setSidebar() {
  const $sidebar = $('.sidebar');
  $sidebar.hide();

  // ...

  $sidebar.css({
    'background-color': 'pink',
  });
}
```

- Use `find` with scoped jQuery object queries.

```
// bad
$('ul', '.sidebar').hide();
$('.sidebar').find('ul').hide();
$('.sidebar').children('ul').hide();

// good
$('.sidebar ul').hide();
$('.sidebar > ul').hide();
$sidebar.find('ul').hide();
```

- Handle failure and complete events when performing AJAX calls.

```
$.getJSON('http://example.com/json', (data) => {
  // Do something with the JSON data
}).fail((jqXHR, textStatus, errorThrown) => {
  // Show an error message
}).always(() => {
  // Hide a loading indicator if required
});
```

TypeScript

Iterators (Loops)

- Don’t use iterators. Prefer JavaScript’s higher-order functions instead of loops like `for-in` or `for-of`.
  - Use `map()` / `every()` / `filter()` / `find()` / `findIndex()` / `reduce()` / `some()` / … to iterate over arrays, and `Object.keys()` / `Object.values()` / `Object.entries()` to produce arrays so you can iterate over objects.

```
const numbers = [1, 2, 3, 4, 5];

// bad
let sum = 0;
for (let num of numbers) {
  sum += num;
}
sum === 15;

// good
let sum = 0;
numbers.forEach((num) => {
  sum += num;
});
sum === 15;

// best (use the functional force)
const sum = numbers.reduce((total, num) => total + num, 0);
sum === 15;

// bad
const increasedByOne = [];
for (let i = 0; i < numbers.length; i++) {
  increasedByOne.push(numbers[i] + 1);
}

// good
const increasedByOne = [];
numbers.forEach((num) => {
  increasedByOne.push(num + 1);
```

```
});

// best (keeping it functional)
const increasedByOne = numbers.map((num) => num + 1);
```

Properties

- Use dot notation when accessing properties.

```
const luke = {
  jedi: true,
  age: 28,
};

// bad
const isJedi = luke['jedi'];

// good
const isJedi = luke.jedi;
```

- Use bracket notation `[]` when accessing properties with a variable.

```
const luke = {
  jedi: true,
  age: 28,
};

function getProp(prop) {
  return luke[prop];
}

const isJedi = getProp('jedi');
```

- Use additional trailing commas for cleaner git diffs.

```
// bad
const hero = {
  firstName: 'Dana',
  lastName: 'Scully'
};

const heroes = [
  'Batman',
```

```
  'Superman'
];

// good
const hero = {
  firstName: 'Dana',
  lastName: 'Scully',
};

const heroes = [
  'Batman',
  'Superman',
];
```

Variables

> Do not use `const` or `let` when targeting Internet Explorer without a transpiler.

- Always use `const` or `let` to declare variables. Not doing so will result in global variables. We want to avoid polluting the global namespace.

```
// bad
superPower = new SuperPower();

// good
const superPower = new SuperPower();
```

- Use `const` for all of your references; avoid using `var`.

```
// bad
var a = 1;
var b = 2;

// good
const a = 1;
const b = 2;
```

- Use one `const` or `let` declaration per variable or assignment.

```
// bad
const items = getItems(),
    goSportsTeam = true,
    dragonball = 'z';

// bad
// (compare to above, and try to spot the mistake)
const items = getItems(),
    goSportsTeam = true;
    dragonball = 'z';

// good
```

```
const items = getItems();
const goSportsTeam = true;
const dragonball = 'z';
```

- Group all your `const`s and then group all your `let`s.

```
// bad
let i, len, dragonball,
    items = getItems(),
    goSportsTeam = true;

// bad
let i;
const items = getItems();
let dragonball;
const goSportsTeam = true;
let len;

// good
const goSportsTeam = true;
const items = getItems();
let dragonball;
let i;
let length;
```

- Assign variables where you need them, but place them in a reasonable place.

```
// bad - unnecessary function call
function checkName(hasName) {
  const name = getName();

  if (hasName === 'test') {
    return false;
  }

  if (name === 'test') {
    this.setName('');
    return false;
  }
```

```
  return name;
}

// good
function checkName(hasName) {
  if (hasName === 'test') {
    return false;
  }

  const name = getName();

  if (name === 'test') {
    this.setName('');
    return false;
  }

  return name;
}
```

- Don’t chain variable assignments.

```
// bad
(function example() {
  // JavaScript interprets this as
  // let a = ( b = ( c = 1 ) );
  // The let keyword only applies to variable a; variables b and c become
  // global variables.
  let a = b = c = 1;
}());

console.log(a); // throws ReferenceError
console.log(b); // 1
console.log(c); // 1

// good
(function example() {
  let a = 1;
  let b = a;
  let c = a;
}());
```

```
console.log(a); // throws ReferenceError
console.log(b); // throws ReferenceError
console.log(c); // throws ReferenceError

// the same applies for `const`
```

- Avoid linebreaks before or after `=` in an assignment. If your assignment violates `max-len`, surround the value in parens

```
// bad
const foo =
  superLongLongLongLongLongLongLongFunctionName();

// bad
const foo
  = 'superLongLongLongLongLongLongLongString';

// good
const foo = (
  superLongLongLongLongLongLongLongFunctionName()
);

// good
const foo = 'superLongLongLongLongLongLongLongString';
```