Coding Standards

Prepared by Md. Alif Bin Turjo

Indentation

Use 2 spaces for indentation (common in JS) and avoid mixing tabs and spaces.

```
function greet(name) {
  console.log(`Hello, ${name}`);
}
```

Quotes

Use single quotes

```
foo = 'bar';
```

Line length

For readability, avoid lines longer than 80 characters.
If a JavaScript statement does not fit on one line, the best place to break it, is after an operator or a comma.

```
document.getElementById("demo").innerHTML =
"Hello Dolly.";
```

Braces

Your opening braces go on the same line as the statement.

```
if (true) {
  console.log('winning');
}
```

Variable type

Use `const` and `let` instead of `var`

```
const PI = 3.14;      // For constants
let count = 0;        // For variables
```

Naming

Use Meaningful Variable and Function Names.
Avoid single-letter names. Be descriptive with your naming.
lowerCamelCase for variables, properties, and function names.
UpperCamelCase for class names.
UPPERCASE_SNAKE_CASE for constants.

```
const userAge = 25;
```

```
function BankAccount() {
}
```

```
const SECOND = 1 * 1000;
```

Operators

Always put spaces around operators (= + - * /), and after commas
Use === and !== over == and !=

```
let x = y + z;
const myArray = ["Volvo", "Saab", "Fiat"];
```

```
if (a !== '') {
  console.log('winning');
}
```

Functions

Write small functions

Return early from functions

```
function isPercentage(val) {
  if (val < 0) {
    return false;
  }

  if (val > 100) {
    return false;
  }

  return true;
}
```

Anonymous function

Use Arrow Functions for Anonymous Functions. Arrow functions also preserve `this` context, which is helpful in many cases.

```
setTimeout(() => {
  console.log("Done");
}, 1000);
```

Error handling

Implementing robust error handling using try-catch blocks for synchronous code and .catch() for promises.

```
try {
  const data = fs.readFileSync('file.txt', 'utf-8');
} catch (err) {
  console.error('Error reading file:', err);
}
```

Comments

Use slashes for both single-line and multi-line comments. Try to write comments that explain higher-level mechanisms or clarify difficult segments of your code. Don't use comments to restate trivial things.
Use // FIXME: to annotate problems.
Use // TODO: to annotate solutions to problems.

```
// 'ID_SOMETHING=VALUE' -> ['ID_SOMETHING=VALUE', 'SOMETHING', 'VALUE']
var matches = item.match(/ID_([^\n]+)=([^\n]+)/));

// This function has a nasty side effect where a failure to increment a
// redis counter used for statistics will cause an exception. This needs
// to be fixed in a later iteration.
function loadUser(id, cb) {
  // ...
}

var isSessionValid = (session.expires < Date.now());
if (isSessionValid) {
  // ...
}
```

```
  // FIXME: shouldn't use a global here
  total = 0;
```

```
// TODO: total should be configurable by an options param
this.total = 0;
```

References

- https://www.w3schools.com/js/default.asp
- https://github.com/felixge/node-style-guide
- https://github.com/airbnb/javascript