Coding Conventions and Guidelines (StandardJS Style Guide)

Naming Convention

1. General:

- Avoid overly generic or lengthy names. For example:
  Bad Practice: *data_structure, my_list, info_map, dictionary_for_the_purpose_of_storing_data_representing_word_definitions*
  Good Practice: *user_profile, menu_options, word_definitions*
- Use **camelCase** for variable and function names.
- Use **PascalCase** for class and component names.
- Abbreviations should be consistent (e.g., `httpServer`, not `HTTPServer`).

2. Files & Folders:

- Use ***kebab-case*** for files and folders.
- **Example:** user-profile.js, auth-service.js
- Next.js pages must follow its routing conventions (file name = route).

3. Variables:

- Always use ***const*** for values that don't change, ***let*** if reassigned.
- **Example:**
```
const count = 10
let name = 'Ornob '
```
- Never use ***var***.

4. Functions:

- Function names should be ***camelCase***.
- Examples:

```
function addData() {}
const fetchData = () => {}
```

5. Constants:

- Use UPPER_SNAKE_CASE for constants.
- **Example:**

```
const API_BASE_URL = 'https://api.example.com'
const MAX_COUNT = 10
```

6. Boolean Names:

- Prefix with is, has, can or should.
- **Example:** *isLoggedIn, hasAccess*. *canEdit*

7. React Components & Classes:

- Use **PascalCase** for React components and ES6 classes.
- **Example:**

```
class UserProfile extends Component {}
function AuthContext() {}
```

Code Layout

1. Indentation:

- Indentation means the number of spaces (or tabs) you put at the start of the line.
- Use 2 spaces per indentation level
- Spaces are preferred over tabs.
- **Example:**

❌ **Bad**

```
function getUser(){
    console.log('User')
}
```

☑ **Good**

```
function getUser () {
  console.log('User')
}
```

2. Maximum Line Length and Line break:

- Limit all lines to a maximum of 80-100 characters.
- Break long lines sensibly and keep indentation consistent.

3. Semicolons:

- **Do not use semicolons** — StandardJS automatically handles them.
- **Example:**

❌ **Bad**

```
const user = 'Sakib';
```

☑ **Good**

```
const user = 'Sakib'
```

4. Quotes

- Use single quotes (' ') for strings.
- In JSX, double quotes (" ") are allowed.
- **Example:**

```
const name = 'Ornob'
const element = <h1 className="title">Hello</h1>
```

❌ **Bad**

```
const user = {
  name: 'Ornob',
  age: 21,
}
```

☑ **Good**

```
const title = 'Hello World';
<div className="container"></div>;
```

5. Trailing Commas:

- Always include *trailing commas* in objects and arrays to make diffs cleaner.
- **Example:**

❌ **Bad**

```
const user = {
  name: 'Alice',
  age: 25
}
```

☑ **Good**

```
const user = {
  name: 'Ornob',
  age: 21
}
```

Others for MERN+Next.js

1. Functions:

- Prefer **arrow functions** for anonymous or inline functions.
- **Example:**

❌ **Bad**

```
function add (a, b) {
  return a + b
}
```

☑ **Good**

```
const add = (a, b) => a + b
```

2. Props:

- Always destructure props in React components for clarity.
- **Example:**

❌ **Bad**

```
function User (props) {
  return <p>{props.name}</p>
}
```

☑ **Good**

```
function User ({ name }) {
  return <p>{name}</p>
}
```

3. Async/Await + try/catch:

- Use async/await instead of .then() and .catch().
- Always handle errors properly using `try/catch`.
- Keep error messages clear and informative.
- **Example:**

❌ **Bad**

```
fetch('/api/users')
  .then(response => response.json())
  .then(data => {
    console.log('Users:', data);
  })
  .catch(err => {
    console.log(err);
  });
```

☑ **Good**

```
async function getUsers () {
  try {
    const res = await fetch('/api/users')
    const data = await res.json()
    console.log('Users:', data)
  } catch (err) {
    console.error('Failed to fetch users:', err)
  }
}
```

4. Mongoose

- Model names should be *PascalCase*.
- Collection names should be lowercase plural.
- **Example:**

❌ **Bad**

```
const user = mongoose.model('user', userSchema)
```

☑ **Good**

```
const User = mongoose.model('users', userSchema)
```

5. Documentation and Comments:

- Use **JSDoc** for functions and important modules.

```
/**
 * Calculates total price after tax.
 * @param {number} amount - Base amount
 * @param {number} taxRate - Tax rate in percentage
 * @returns {number} - Total amount after tax
 */
function calculateTotal (amount, taxRate) {
  return amount + (amount * taxRate / 100)
}
```

- In comments explain **WHY** not **WHAT**.
- **Example:**

❌ **Bad**

```
// Increment count by 1
count++
```

☑ **Good**

```
// This increment ensures reactivity triggers a re-render
count++
```

6. Imports & Exports:

- Use ES Modules (import/export) syntax.
- Do not use *require()* or *module.exports*.
- **Example:**

❌ **Bad**

```
const express = require('express')
module.exports = router
```

☑ **Good**

```
import express from 'express'
export default router
```

Important Tool:

StandardJS is an opinionated JavaScript style guide, linter, and formatter in one. It **automatically enforces clean, consistent code style** and eliminates the need for configuration files.

Why StandardJS?

- No semicolons (they're automatically handled).
- No configuration — just one rule set.
- Enforces consistent spacing, naming, and indentation.
- Prevents most common mistakes automatically.

Steps to Set Up StandardJS

1. **Install VS Code**
2. **Install the StandardJS Extension** from VS Code Marketplace.
3. **Install StandardJS in your project:**

```
npm install standard --save-dev
```

4. **Run StandardJS to check your code:**

```
npx standard
```

5. **Fix issues automatically:**

```
npx standard -fix
```

Example

To automatically format code on save:

```
{
  "editor.formatOnSave": true,
  "standard.enable": true
}
```

Summary

Using **StandardJS** ensures that:

- Your codebase remains **clean, readable, and consistent**.
- All developers follow the same **minimalist, semicolon-free** convention.
- You save time on formatting and focus more on logic and structure.

References:

- https://standardjs.com
- https://react.dev/learn
- https://nextjs.org/docs
- https://nodejs.org/en/docs