Coding Conventions and Guidelines (Airbnb Style Guide)

Naming Convention

1. General:

- Avoid using names that are too general or too wordy. 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*
- When using ***CamelCase*** names, capitalize all letters of an abbreviation (e.g. *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 = 5 ;
let name = 'Sakib ' ;
```
- Never use ***var***.

4. Functions:

- Function names should be ***camelCase***.
- Examples: *addData()*, *fetchData()*

5. Constants:

- Use UPPER_SNAKE_CASE for constants.
- **Example:** *API_BASE_URL*, *MAX_COUNT*

6. Boolean Names:

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

7. React Components & Classes:

- Use *PascalCase* for React components & classes.
- **Example:** *UserProfile*, *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 greet(name){
if(name){
console.log("Hello " + name);
}else{
console.log("Hello Guest");
}
}
```

☑ **Good**

```
function greet(name) {
  if (name) {
    console.log("Hello " + name);
  } else {
    console.log("Hello Guest");
  }
}
```

2. Maximum Line Length and Line break:

- Limit all lines to a maximum of 100 characters.
- Make sure to indent the continued line appropriately.

3. Semicolons:

- Always end statements with semicolons.
- **Example:**

❌ **Bad**

```
const name = 'Alice'
console.log(name)
```

☑ **Good**

```
const name = 'Alice';
console.log(name);
```

4. Quotes

- Use single quotes ' ' in JavaScript.
- Use double quotes " " in JSX.
- **Example:**

❌ **Bad**

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

☑ **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: 'Alice',
  age: 25,
};
```

Others for MERN+Next.js

1. Functions:

- Always use *arrow functions* for consistency.
- **Example:**

❌ **Bad**

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

☑ **Good**

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

2. Props:

- Always *destructure* props for cleaner, more readable, and maintainable components.
- **Example:**

❌ **Bad**

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

☑ **Good**

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

3. Async/Await + try/catch:

- Always use *async/await* for asynchronous code and wrap it in *try/catch* to handle errors.
- **Example:**

❌ **Bad**

```
app.get('/user', (req, res) => {
  User.findById(req.params.id).then(user => {
    res.send(user);
  });
});
```

☑ **Good**

```
app.get('/user', async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    res.send(user);
  } catch (err) {
    res.status(500).send({ error: err.message });
  }
});
```

4. Mongoose

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

❌ **Bad**

```
mongoose.model('User', userSchema, 'UserCollection');
```

☑ **Good**

```
mongoose.model('User', userSchema, 'users');
```

5. Documentation and Comments:

- Use *JSDoc* for functions.

```
/**
 * Fetch user by ID
 * @param {string} id
 * @returns {Promise<User>}
 */
const getUser = async (id) => {
  return User.findById(id);
};
```

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

❌ **Bad**

```
// increment i
i++;
```

☑ **Good**

```
// retrying request after failure
i++;
```

6. Imports & Exports:

- Use ES6 imports, not *require*.
- **Example:**

❌ **Bad**

```
const express = require('express');
module.exports = app;
```

☑ **Good**

```
import express from 'express';
export default app;
```

Important Tool:

Airbnb Style Guide is a set of rules built on **ESLint**. **ESLint** checks your code for errors, bad practices, and style violations. is rule-based, not just formatting. It's like a grammar checker for code.

When you **install ESLint + Airbnb config**, it means ESLint will enforce the **Airbnb JavaScript Style Guide** in your project.

Steps to Set Up ESLint with Airbnb Style Guide

- Install VS Code.
- Install the ESLint Extension in the VS Code.
- Then Install the Airbnb config:
  npm install --save-dev eslint
  eslint-config-airbnb
  eslint-plugin-import
  eslint-plugin-jsx-a11y
  eslint-plugin-react
  eslint-plugin-react-hooks
- Extend Airbnb in *.eslintrc.json*:

```
{
    "extends": ["airbnb", "airbnb/hooks"]
}
```

- Now ESLint checks your code against Airbnb rules.

References:

1. https://github.com/airbnb/javascript
2. https://nextjs.org/docs
3. https://react.dev/learn
4. https://typescript-eslint.io/