Coding convention

I. Naming conventions

I.1. Component Names

- Component names should be in `PascalCase`.
- Component names should be descriptive.
- Component names should be short and unique.

```
ComponentName
```

- Component in a module should start with module name

```
UserForm
UserTable
```

I.2. Component TS Interface

- Component TS interface names should be in `PascalCase`.
- Do not start with an `I`
- Put the component name in the interface name.
- Place the interface above the component.

```
interface ComponentNameProps {
  // ...
}
```

I.3. In-Component function

- In-component function names should be in `camelCase`.
- Start with verb to indicate the action of the function.
- Start with `is` to indicate the boolean result.
- Start with `get` to indicate the value result.
- Start with `set` to indicate the setter function.
- Start with `has` to indicate the existence of something.
- Start with `add` to indicate the addition of something.
- Start with `remove` to indicate the removal of something.

```
isSomething(): boolean;
getSomething(): string;
setSomething(value: string): void;
hasSomething(): boolean;
addSomething(value: string): void;
removeSomething(value: string): void;
```

I.4. Variables

- Variables should be in `camelCase`.
- Use `const` by default, unless a variable needs to be reassigned. Then use `let`. `var` is not allowed.

```
const foo = otherValue;  // Use if "foo" never changes.
let bar = someValue;     // Use if "bar" is ever assigned into later on.
```

I.4. In-Component states, hooks

- In-component states and hooks should be in `camelCase`.
- Hooks caller should be on `top` of the component.
- State definition should be on `top` (next to hooks call) of the component.

```
const { something } = useSomething();
const [foo, setFoo] = useState(initialValue);
```

I.5. Constant, enum

- Constant names should be in `UPPER_SNAKE_CASE` (aka `SCREAMING_SNAKE_CASE`).
- Constant should end with `_VALUE`. (e.g. `MY_CONSTANT_VALUE`).
- Enum names should be in PascalCase.
- Enum values should be in `UPPER_SNAKE_CASE` (aka `SCREAMING_SNAKE_CASE`).
- Enum should end with an `noun` (e.g. `SomethingTypes`).
- Should be descriptive.
- **If duplicate to other constant in other modules, should be moved to `shared`**

```
const MY_CONSTANT_VALUE = 1;
enum SomethingTypes {
  THIS,
  THAT,
}
```

I.6. Utility function

- Utility function names should be in `camelCase`.
- Start with `get` to indicate the value result.
- Start with `has` to indicate the existence of something.
- Start with `add` to indicate the addition of something.
- Start with `remove` to indicate the removal of something.

II. Comments

II.1. JSDoc Comments

- Type in /** */ to start using JSDoc.
- **Do not** declare `@params` and `@return` type in comments. Should be clearly defined in each interface that applies to the function. Only required when add informations. Omit at all other cases.

```
/**
 * POSTs the request to start coffee brewing.
 * @param amountLitres The amount to brew. Must fit the pot size!
 */
brew(amountLitres: number, logger: Logger) {
  // ...
}
```

II.2. In-line, Block comments

- Should be on top of the code, briefly describe the function or the process flow of data.

```
// Inline comment
...something hard to understand...

/* Block comment */
...something hard to understand...
```

III. Code Style

III.1. Exceptions

- Always use `new Error()` when instantiating exceptions

```
throw new Error('Foo is not a valid bar.');
```

III.2. Promises

- In most case, using `async await` to handle API.
- Always wrap `async` function in `try catch` block.
- Use `Promise.all` to handle multiple promises.
- Always declare loading state for async tasks.

```
const [loading, setLoading] = useState(false);
const getAPIData = async () => {
    setLoading(true);
    try {
        const data = await fetchData();
        return data;
    } catch (error) {
        throw error;
    } finally {
        setLoading(false);
    }
}
```

III.3. Equality Checks

- Always use triple equals (`===`) and not equals (`!==`).

```
if (foo === bar) {
  // ...
}
```

III.4. Nullability

- Only use `null` when it is explicitly intended.
- Use `?` (aka. Optional chaining) to indicate nullable type.