HTML Style Guide

Naming

1. Write custom HTML tag names in Pascal case with self closing tag if you do not have slots inside, e.g. <MyTag/>.

Format

1. Write custom properties first, then HTML native properties in HTML tags.
2. Write custom props in kebab-case, e.g. <MyTag :my-prop=”1” />.
3. Write custom props in different lines if there are more than 3 props for an HTML tag.
4. Add comments to explain HTML tags that might be hard to understand and anywhere else necessary.
5. Set the indentation to 2 spaces.

Structure

1. Refactor the v-if statement into a computed property or a function if it is too long.

CSS Style Guide

Naming

1. Write words in lowercase and link them by hyphens (kebab-case or spinal-case).
2. For wrapper classes, use words wrapper, box, or container like <my-class>-wrapper, <my-class>-box, <my-class>-container, e.g. page-wrapper, item-box, card-container.
3. Write meaningful and descriptive names for classes.
4. Do not write abbreviation for words, e.g. msg and btn.

Format

1. Write no more than 4 words for each name.
2. Avoid inline styles on HTML elements and write a class for it instead, unless it is really necessary.
3. Avoid styling HTML tags in style scripts, add a class for the tags.
4. Avoid the overuse of !important.
5. Write font-size in em, if TV screen need to be supported.
6. Set the indentation to 2 spaces.
7. Write **no more than 3 levels** of nesting in SCSS code.
8. Write CSS code in mobile-first approach.
9. .my-class {
10. background-color: red;
11. font-size: 12px;
12. opacity: 0.7;
13. }
14. @media screen **and** (min-width: 768px) {
15. .my-class {
16. */\* This will override the opacity: 0.7 \*/*
17. opacity: 1;
18. }

}

1. Set correct breakpoints:

|  |  |  |
| --- | --- | --- |
| **From** | **Up to** | **Corresponding name** |
| 0 | 575px | XS |
| 576px | 767px | SM |
| 768px | 991px | MD |
| 992px | 1199px | LG |
| 1200px | 1599px | XL |
| 1600px | 2559px | XXL |
| 2600PX | above | TV |

Only add custom media queries if necessary.

Structure

1. Write general styles for basic elements into a shared file, these styles are shared by all components, e.g. button-main.
2. Write layout specific styles within each layout component. These styles are only shared with components within that layout, e.g. auth-layout.
3. Overwrite the styles of an element in its own component if necessary by adding a new class.
4. Write component specific styles into style tag, only import shared styles and SCSS base files.

JavaScript Style Guide

Naming

1. Write words in camel case.
2. Write meaningful and descriptive names, e.g. toggleInfoPanel, displayTableData, etc.
3. Write no more than 4 words for each variable name unless it’s really necessary.
4. Write question sentence as function name for checking functions, e.g. isPageLoaded, isPanelShown, etc.

Format

1. In methods, write function syntax like
2. <myFunctionName>: **function** () {
3. …

}

1. Add semi-colon (;) at the end of each line.
2. Always use curly braces ({}) even if there is just one line.
3. **if** (<condition>) {
4. … (even just one line)

}

1. Always use uppercase and snake\_case for exported constants, especially in router and store, e.g. export const AUTH\_LOGIN. For local constants, define them in camel case.
2. Add sufficient and descriptive comments to explain important lines of code or complex logic to help other developers understand.
3. Set the indentation to 2 spaces.

Structure

1. Try to avoid a function with too many lines (maximum around 50 lines). Refactor relevant lines of code into separate functions. Do not exceed the limit unless it is really necessary.
2. *// Possible grouping of relevant lines of code.*
3. login () {
4. **this**.checksBeforeLogin();
5. **this**.sendLoginRequest();
6. **this**.afterLoginRequest();

}

Vuex Store Style Guide

Naming Convention

1. Write words in camel case, e.g. myVariable, myFunction.
2. Write meaningful and descriptive names, e.g. getProductList, displayLoadingIcon, etc.
3. Write no more than 4 words for each variable name unless it is really necessary.
4. Write question sentence as function name for checking functions, e.g. isLoadingSet, isModalShown, etc.

Format

1. In methods, write function syntax like <my-function-name> () {...}.
2. my-**function**-name () {
3. ...

}

1. Add semi-colon (;) at the end of each line.
2. Always use curly braces ({}) even if there is just one line.
3. **if**(<condition>) {
4. ... (even just one line)

}

1. Always use uppercase + snake\_case for exported constants, e.g. SET\_LOADING.
2. Add sufficient and descriptive comments to explain important lines of code or complex logic to help other developers understand.
3. Set the indentation to 2 spaces.

Structure

1. Export each function as a constant and import in components.
2. All exported constants go to the top of the page below imports.
3. Try to avoid a function with too many lines (around 50 lines), refactor relevant lines of code into separate  
   functions. Do not exceed the limit unless it is really necessary.