SafariBookings

Coding Style Guide

Version 1.0
2023-01-27

Create readable and maintainable code

- **Use meaningful variable names**

  When you create variables, give them descriptive names that clearly explain their purpose. This will make your code easier to read and understand for both yourself and other programmers.

- **Use Comments to explain complexity**

  Comments help to explain complex pieces of code, making it easier to understand. They can also help to document your code, which can be useful when revisiting or updating it later.

- **(re-)Structure your code**

  ### No bulky functions!
  If a specific function becomes too bulky, refactor it by breaking it up.

  Breaking up long code into smaller functions will make it easier to read and easier to debug. Each function should have a clear purpose and should be named accordingly.

  ### Minimize repeating code!
  If you write/detect repeating code, refactor it and use a separate function or a local (PHP) closure, to reduce it.

Laravel php guidelines

- **Where to put your PHP code**

  SQL / Database specific code will be placed into the relevant Model

  Main focus of controller actions is to set up the $this->view_vars and possibly differentiate between views.

  No bulky controller action methods!

  If setting up the $this->view_vars takes a lot of logic and results in a bulky controller action method, move this 'action setup' code to a domain specific helper. You can easily pass the $controller->view_vars by reference to set it up.

  All other code goes into static domain specific (laravel) Helper Classes or into custom specific (OO) PHP Classes.

- **Minimize use of <php ?> and @php in blades**

  Use the regular functional blade directives like @if, @foreach, etc etc , to handle your code in the blade view.

  Prepare your blade variables (view_vars) in the controller action method or domain specific helper classes.

  use PHP closures (passed by view_vars) or helper methods to access specific code.

- **(re-)Structure your blades**

  If you write/detect repeating blade html, refactor it and use @include and a separate blade partial to reduce it.

Naming conventions

When you create variables, classes and functions give them descriptive names that clearly explain their purpose. We do not distinguish between scopes and member visibility.

UC :	*UPPERCASE*	*LC :*	*lowercase*
UCC :	*UpperCamelCase*	*LCC :*	*lowerCamelCase*

Variables	**LCC / LC**
Functions / Methods	**LCC**
Classes / Interfaces	**UCC**
Constants	**UCC**
Enumerations	**UCC / UC**

Coding conventions

Focus on short and readable code.

- Use of curly brackets,

```
[code or method] {
      .. code ..
}
```

- Use conditional (ternary) operators for assignments over if-then-else

```
$thisVariable = ($boolean)?$valueTrue:$valueFalse;
```

Writing HTML

- Write valid, clean and semantic HTML. Optimize performance, accessibility and maintainability by having a well-structured DOM, as well as providing alt attributes, titles and descriptions whenever possible.
- Keep styling definitions and helpers out of your HTML. For example, prevent the use of inline styles (`<span style="color:red;">`) or HTML tags inserted purely for layout purposes (`<section class="box__inner">`)
- Regarding assets: When displaying images, optimize for different situations by using optimal file types, a combination of `<picture>` & `<source>` tags, and/or the `srcset` attribute. Webfonts should be displayed through Google Fonts.
- Try to write code according to the rule of separation whenever possible: HTML for content, CSS for presentation, JavaScript for interactivity and non-essential presentation purposes.

Writing CSS

- Write responsive, mobile-first CSS that results in layouts displayed correctly on every device, browser and viewport size.
- Structure your CSS code by using the CSS preprocessor SASS. For each module create a separate SCSS file. Compile to one or more minified CSS files.
- Write class-based DRY CSS according to the BEM / BEMIT methodology (.nav, `.nav__item`, `.nav--secondary`)
- Split up CSS for usage *above* and *below* the fold.
- Similar to HTML: Keep good performance, accessibility, and maintainability in mind when writing CSS as well.

Writing Javascript

- Use javascript classes of (const) objects to group domain specific code and/or supporting methods.

  Minimize putting your Javascript code into the main scope.

- When using $(jQuery), try to put a $selector into a variable (that you could start with a '$' char) instead of repeating the selector code each time.

  You can always ($selector_var).refresh() on a jQuery selector variable to update the selector's elements.