⚡Serato CSS

> A simple modern CSS framework full of handy sass dj mixins!

#🤖 Main Composition

- Animations → contains both default animations eg. floating & state animations eg. float-on-hover
- Base → contains CSS resets & hacks 😎
- Components → contains components and these can have sub-components eg. form* component has inputs* well as independent elements are components of their own eg button.
- Documentation → includes a simple markdown file that gets updated with documentation as a particular thing is introduced in the framework.
- Helpers → contains helper classes like spacing, position, typography, flex, grid, visibility, etc.
- Layout → contains main sections of html 5 eg. header, navbar, main, sidebar, footer, etc.
- Plugins → Ideally complicated CSS from community mixin djz.
- Themes → contains colors, gradients and dark-mode theme maybe!
- Utility → contains mixins, media queries, functions, derivatives, variables etc.
- Slang CSS → future feature to use some slang terms in css!

Note: Each folder has a "_exports.scss" file to exports all partials in that folder to the main "serato.scss" file which finally gets compiled and exported for release and publication.

# 📋References

- CSS tricks

- Smashing Magzine
- Una kravets
- Kevin Powell
- Pratham
- Codepen
- Frontend mentor
- Tailwind CSS
- Bulma CSS
- Sass
- SMACSS / BEM
- Material UI
- Bootstrap
- CSS battles
- Design Tools
- Design course

# 🚀Rough Ideas

- Headings (Typography: heading, subheading, normal, small, black etc → see sitka font variants in figma.
- Text effects (morphing, text gradient etc.)
- Paragraphs (truncate vs ellipsis)
- Border styles (outline, solid, dashed, animated, neon)
- Code and sample text styles
- Buttons (look for a good click effect → see material UI)
- Quotations (Notion style vs native quotes)
- States (hover, focus, disabled)

- Links (Native, Iconic, buttoned)
- Images (aspect ratio, fluid vs fixed size, rounded vs square)
- Shadows (see CSS-scan shadows)
- Tables (borderless, outline, flat → see material UI, tailwind and bootstrap tables)
- Lists (link-list, ordered vs un-ordered, menu vs listing)
- Element types (block vs flex vs inline vs grid)
- Spacing (margin, padding, position, z-index)
- Groupings (iconic-text)
- Layout (navbar, hero, mid/content, footer, sidebar)
- Responsiveness (media queries, prefers-reduced motion, dark mode etc.)
- Iframes (framed : phone / pc frame, blended / faded, embedded)
- Color Name & shades (see iOS watch color system on figma)
- JavaScript / React components
- Entities (@mentions)
- Forms & inputs
- Svg / Canvas
- Media (video, audio)
- Plugins
- Drag & drop
- Scrolling & scroll bars
- flip-v vs flip-h (common helpers → see design tools like figma)
- spacer / responsive divider
- Modals
- animations (see hvr & loader awesome & font awesome animation libraries)
- skeleton loaders (for inbuilt sized elements only)
- pre-loaders
- progress bars

- tooltips
- svg animations
- curve sections (→ see fireship you-tube tutorial)
- Shape helpers (quick styles to make shapes)
- carousel
- mapbox
- navbars

🇲🇦 ORGANIZATION IDEAS FROM (BEM, SMACSS, OOCSS, SCSS)

⇒ SMACSS

Base: To HTML CSS, no class / Id selectors, rather use sematic tags!

Layout: big pages sections - headers, sidebars etc

Module: encapusulated modeles, reusable etc

State: overrides defaults eg. is-open

Theme: optional

⇒ BEM

.{Block}_{Element}

.{Block}_{Element}—{Modifier}

.{Block}—{Modifier}

// Serato BEM

Uses hyphen "-" instead of underscore "_" cause it's easier to type and disambiguates the thing!

⇒ OOCSS

Everything is a component or model, is the media object. Every thing should be encapusulated to do one rensponsibility.

⇒ Mobile first design

Don't apply stlyes in first place which you aught to overide in a media query, instead, go mobile first approach eg.

```
.card {
width: 568px;
height: auto;
    @include media-sm {
    width: 100%;
    }
}
```

instead this below is mobile first!

```
.card {
width: 100%;
    @include media-md {
    width:  568px;
    height: auto
    }
}
```

⇒ Flat CSS , lower specificity and low html structure constrictioning ie. use more easy to evaluate selectors for browsers to laod stuff faster.

⇒ The helper-to-markup prolem:

The multi inline helper classes tie css to a specific html structure or presentation.

Solution: use sass @extend to extend all those into a single easy to change class eg.

.primary-rail {

@extend .pull-left, .col-md-6, .small

}

instead of in html : <h1 class='pull-left .col-md-6 small'>foo</h1>

Even use this approach to create classes from mixins and extend these instead of calling the mixin directly, it allows to easily update the class in one instance instead of looking for the mixin applications all over.

However for user can surely still just use mixin as is, just not recommended!

⇒ states

Element is in a certain condition eg. is-open or positon / visual state eg. is-fixed or when-mobile.

Examples of state modifiers:

- is-hidden (expanded)
- is-open (closed)
- is-active (inactive)
- is-small (medium, large)
- when-mobile (desktop, widescreen) eg. medium-when-desktop
- is-width-full
- has-icon (image, text)
- is-dark, not the —dark modeifer for dark theme!!!
- is-hovered (focused)
- is-animating ???
- is-success (error,warning,pending)
- is-selected

other states mdoifiers — visual states

—disabled

—non-selectable

—outline

—animated ???

⇒ Helper class naming:

flex- or grid- eg.

flex-row means a flex item with flex-direction row.

This is to say, if you want a flex without anything else, add flex class, if you want that with flex-direction row then instead do flex-row and same with grid-column or row. Another one can be, a button without any mods is button, a button outline stlye is button-outline, and then a block is block, an inline block is then block-inline. Like the second name after hyphen tells how to mod the element. Doule hyphens are for special modifiers like the dark theme modifier —dark.

TIP: check default vs !important in sass???

Again only use classes for components eg. .nav instead of nav sematic or id='nav'

Since these are to be targeted separately not globally like the sematic elements themselves.

Use class = ' button has-icon' for a button that has icon and text in it, but class='button-iconic' for an icon acting as a button, a clickable icon. A combo can be 'button-outline has-icon' but not 'button-iconic has-icon'

⇒ !important flag:

Conditional states like is-active usually depend on javascript and are usually overidding other existing states, hence use !important flag, only on these states!

⇒ Component-aware-states: eg. tab-is-active or menu-is-active should contain the component name in thier naming or reference and reside just along thier respective component nest.

And for just-in-time loading of css, visual, positional or generic states which are active by default eg. is-animated should be part of base or global scope that they get loaded with base loading. And the other states which are just invocked after wards can be loaded just in time.

⇒ In scope media queries:

All media queries should go along thier respective selectors, we define the item media query within it's nest or scope.

⇒ for extending default styles, use sub-component naming eg. button-disabled which inherits all the default stlyes of a button but addson the disabling styles. However recommended to noly extend mixin derived classes, the latter removes support for just-in-time loading!!! rather use 'btn is-small' if possible.

⇒ Optionated, we take common use case as defaults.

For example is rare that you gonna do flex-jusitfy-items or align-content and the usual is justify-content and align items, so we default to that and provide mods for the latter. ie.

class='justify-between justify-between—items align-center align-center—content' the double hypen modifiers are special combos not commonly used!!!

⇒ special classes: 'twins-row' and 'twins-column' defines two items binded together when the class is applied to parent, and this parent should only contain these two as the direct children, else do flex. And this is handy if you have an icon along a text and want them to be side by side.

Specifying 'is-size-x-desktop' is kind of extrataneous, we do 'medium-when-desktop' to mean is-medium on desktop viewport.