EIS GROUP™

Design Document

EIS Core

Modular CSS Architecture

Author	Chris Co
Status	Draft
Document Number	
Document Version	00.01.00
Last Revision Date	October 12, 2020

Revision History

Name	Date	Reason For Changes	Version
Chris Co	Sep 18, 2020	Initial Version	00.01.00
Chris Co	Oct 12, 2020	Add examples	00.02.00

1 Design

1.0 Introduction

The **Modular CSS** architecture is an approach to CSS styling that takes into account the *best programming principles* taken from diverse methodologies, such as Semantic, Modular, Functional, Object Oriented Programming, etc.

Modular CSS is **User Centric** and **Results Driven**, focusing on maximizing the output with minimal effort from both developers and end users alike.

It is **cross-platform compatible** and **framework agnostic**. Examples below use LESS CSS preprocessor, but SASS is equally compatible.

To get started, download the **css files** into your project.

Or clone demo repository using Next.js setup with built-in support for both TypeScript and

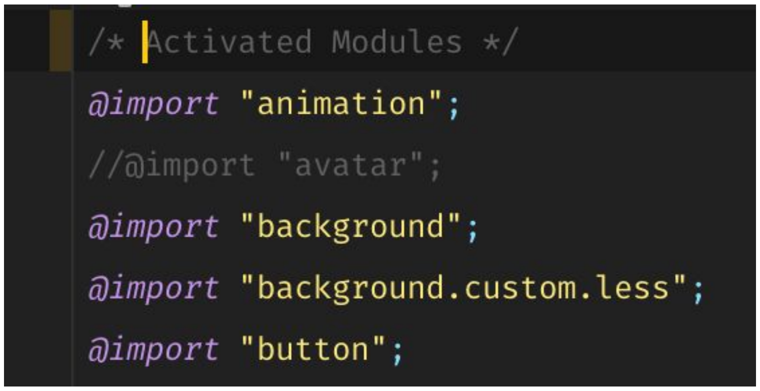
Javascript. With modification, the setup can work with jQuery, CRA, React Native, or any other frontend workflows. All styling resides in the style folder.

1.1 Conventions

- **Use class names** to apply CSS
  => Benefits: fast and easy to re-use/compose/refactor styles, and reduced bundle size. Greatly improve development speed with less-watcher-compiler and LiveReload extension, which reloads CSS without browser refresh, thus preserving all UI states.

  Example: <Link className="button primary" /> - apply button style with primary color

- **Compose styles into modules** that can be easily plugged in/out of the project
  => Benefits: *Write once, use everywhere*. Can partially mix any UI framework that exposes modular import, like Semantic UI.



```
/* Activated Modules */
@import "animation";
//@import "avatar";
@import "background";
@import "background.custom.less";
@import "button";
```

  Example: animation.less only contains animations that can be applied to any element.

- **Project specific styling** should be saved to ***.custom.less files**
  => Benefits: great reuse value, because custom styles only extend/override modules.

  Example: background.custom.less contains colours specific to the project style guide

- **Variables** and **definitions** should be in one place, or declared at the top of the file
  => Benefits: easy to maintain, quick on-boarding for new developers, self documenting.

  Example: _variable.less

- **Extend styles** when possible
  => Benefits: reduced bundle size and keep CSS DRY.

  Example: use syntax `&:extend(.style-to-extend);`

1.2 Naming

- CSS class names follow the **block__element** naming convention from BEM
  => Benefits: avoid namespace collision, without needing unintuitive styled components.

  Example: "form__input" - style input inside a form

- **Style modifiers** follow Semantic UI naming convention, instead of BEM modifiers
  => Benefits: less classes to define, shorter names, more freedom to combine styles

  Example: "form__input padding border" => input now has standard padding and border

1.3 Layout

- All elements should use **Flex** display
  => Benefits: flex layout is cross platform compatible (browser/mobile/desktop).

  Example: display grid or block do not work in React Native.

- **Compose styles** to customise layout
  => Benefits: proven layout styles require zero testing, and no additional work needed

  Example: "flex—row middle center" => centre inner content vertically and horizontally
  Reference: _layout.less

- **Wrap** elements for **responsive layout**
  => Benefits: layout fits any screen size without complex width calculations.

Example: "flex—row wrap" => child elements wrap when container is too narrow
Reference: Layouts.js

1.4 Framework

- Treat external **framework as module**, not foundation
  => Benefits: import only what you need, not an entire framework for a fancy slider. Mix components from different UI frameworks/teams in a single project without bloated code.

  Example: using Dropdown from Semantic UI, and Table from AntDesign.

- **Separate styling from HTML markup**
  => Benefits: testing is simplified, components are more reusable, easy to change styling. Global style guide changes no longer require digging through hundreds of components.

  Example: apply styles through `className`, without direct `style` object manipulation.
  Reference: Modal.js

2 Why?

2.1 What Problem It Solves?

Having built over 20 projects on Bootstrap, and dozens more projects on Semantic UI and other UI frameworks, I realised that **80% of the time**, we either **override existing framework** to achieve the desired result (which produces more bugs later); or write **CSS from scratch**.

This happens **over and over again** for each new project!

The problem was that most of these overrides or writing from scratch were very similar in nature, because most apps need **common behaviours**, like: button/input/link hover/active/focus states, layout spacing, animation, font styling, icon size and glyphs, etc.

The second biggest issue was **incompatibility between UI frameworks**. Which meant little to **no code-reuse** and **hard to maintain** projects.

Thus, it made sense to abstract away these common **styling patterns** in a way that enables granular **customisation** and **extension**, without repeating the work every time.

2.2 How It Benefits Clients?

- **Reduced Support Cost**
- **Confidence in Elegantly Written Software**

2.3 How It Benefits Developers?

- **Reusability** - 60% of most app’s styling is standard across all projects. Modular CSS eliminates this effort altogether.

- **Flexible** and **Maintainable** projects. Because global changes, like onFocus behavior, are isolated into individual CSS classes - they are easy to locate and modify.

- **DRY** - bug fixes and improvements only require updates from one place.

- **Abstraction** and **Encapsulation** - turn ad hoc implementations into standardised API, making framework change, or integration into legacy systems a breeze.

- **Separation of Concerns** - CSS styling is decoupled from component’s logic, which makes testing fast and efficient.

- **Production Boost x2** at least - semantic class names combined with modular styling makes it very intuitive for both debugging and development of new features.

- **Delightful Developer User Experience** (DUX) - well written code makes people happy, improving overall workplace satisfaction level.

2.4 How It Benefits Business?

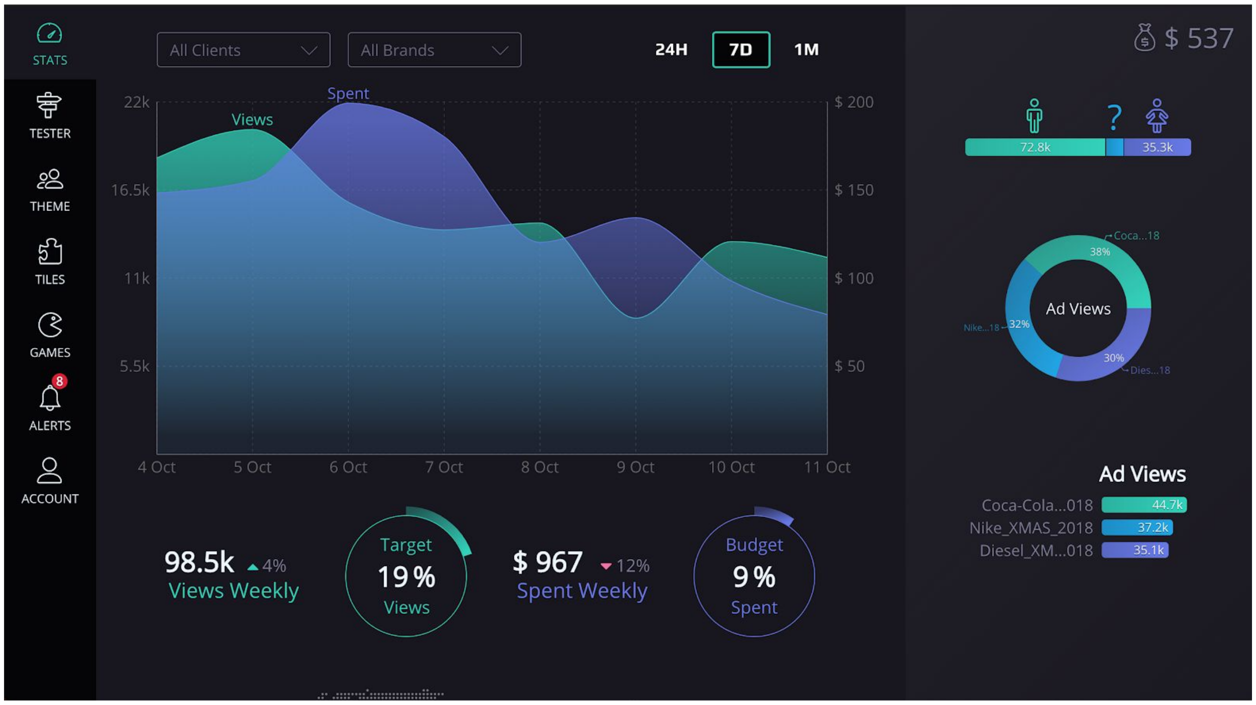
- **Improved Productivity** - faster delivery from R&D means less upfront planning, thus more projects get done.

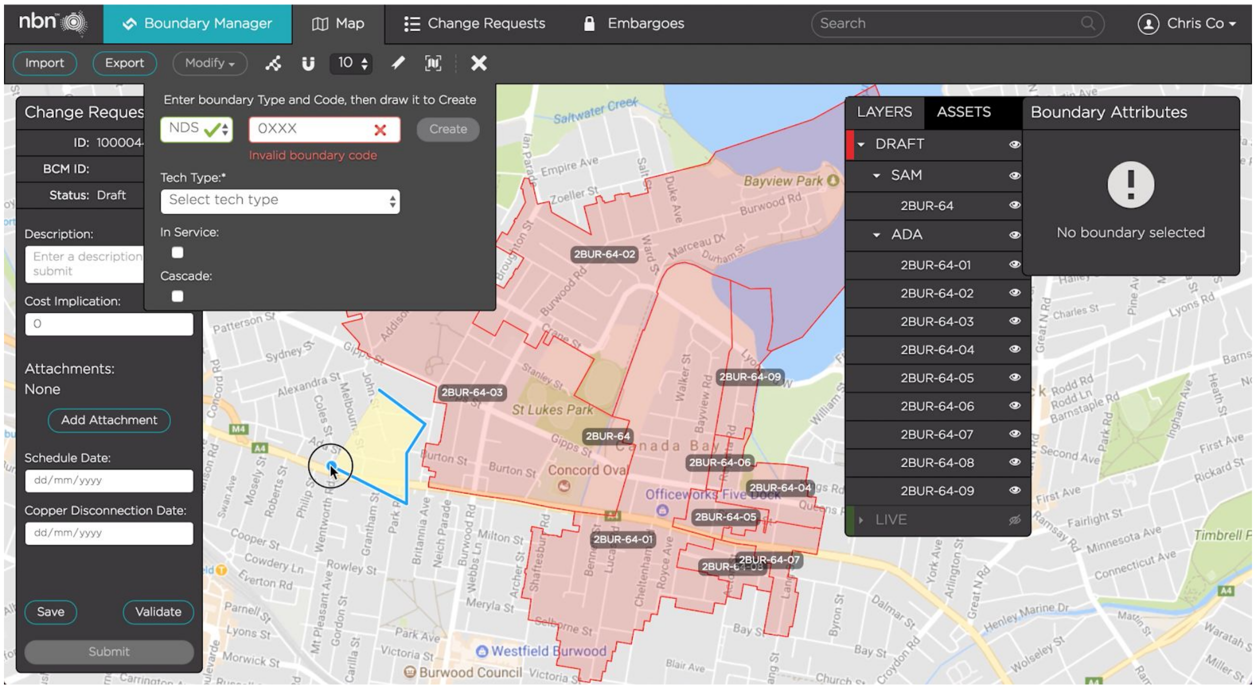
- **Intuitive and Easy to Understand** - semantic class names provide little to no friction for non-tech people to style UI, without having to search for and read documentation.

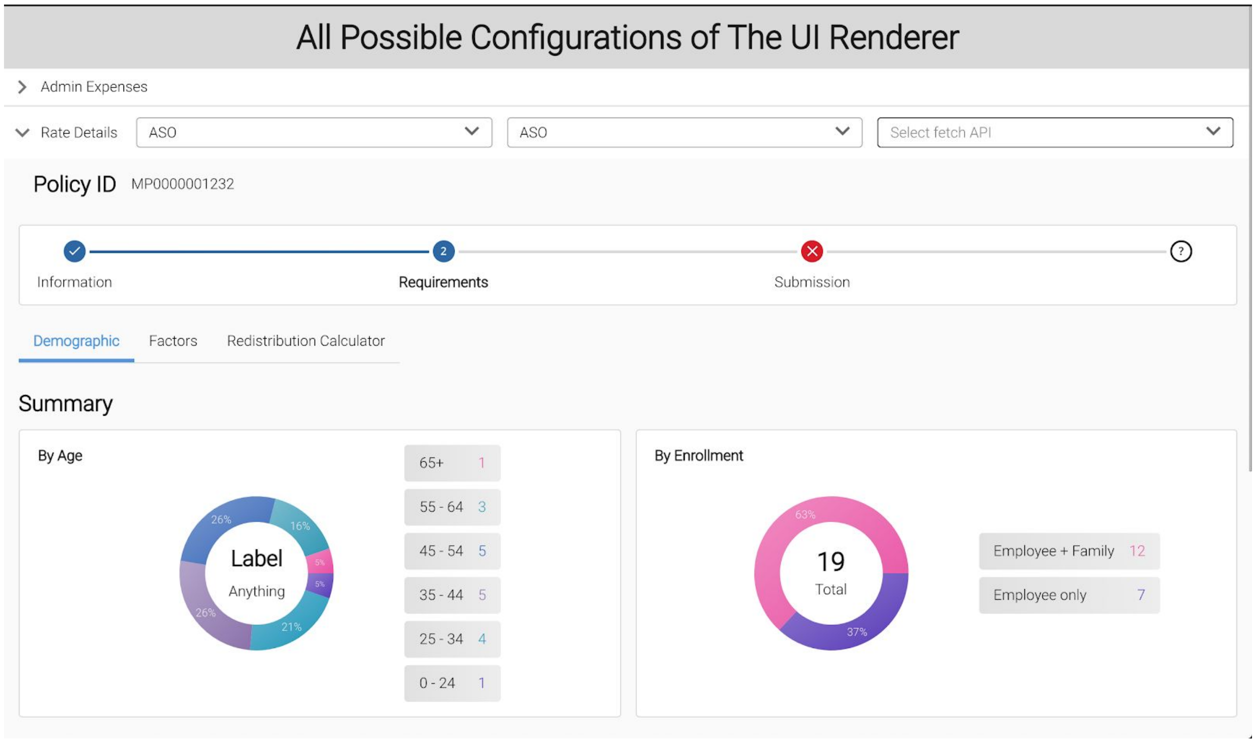
  Example: if you want content to be aligned to the right of a container, just add class "right" to the container (i.e. *what you think is what you get* **WYTIWYG**).

3 Examples

3.1 Apps using Modular CSS







All Possible Configurations of The UI Renderer

> Admin Expenses

Rate Details: ASO | ASO | Select fetch API

Policy ID MP0000001232

Information · Requirements · Submission

Demographic | Factors | Redistribution Calculator

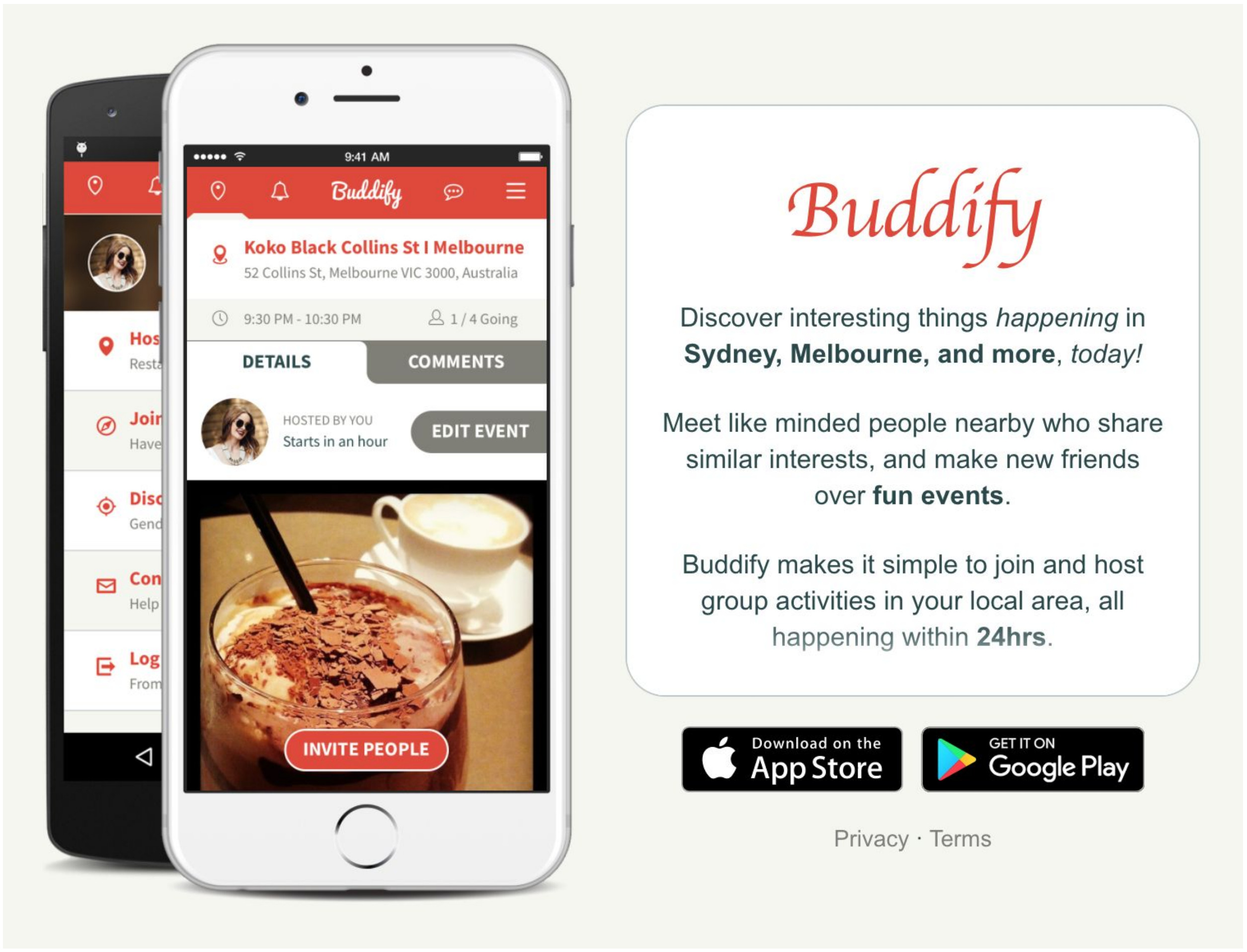
Summary

By Age

65+	1
55 - 64	3
45 - 54	5
35 - 44	5
25 - 34	4
0 - 24	1

By Enrollment

Employee + Family	12
Employee only	7



Buddify

Discover interesting things *happening* in **Sydney, Melbourne, and more**, *today!*

Meet like minded people nearby who share similar interests, and make new friends over **fun events**.

Buddify makes it simple to join and host group activities in your local area, all happening within **24hrs**.

Download on the App Store | GET IT ON Google Play

Privacy · Terms