SIMPLIFY YOUR LIFE WITH SIG COMPONENTS

Created: 2022-10-25 Tue 12:10

CURRENT STATE OF COMPONENTS

Project	UI Framework
Pyregence	CLJS / Reagent
CAMEO / Carbon	CLJS / Reagent
CEO	JS / React
SERVIR Apps	JS / React

BUTTONS

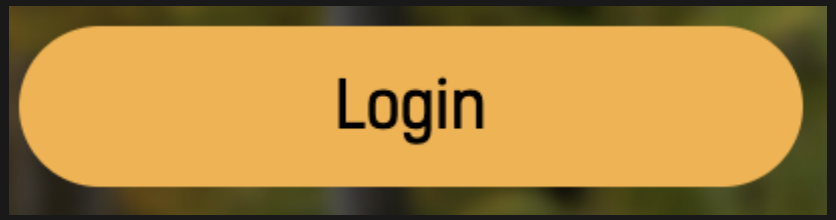
PYREGENCE


```
[:input {:class (<class $/p-form-button)
         :style ($/combine ($/align :block :right) {:margin-to
         :type  "submit"
         :value button-text}]
```

CEO


```
<button className="btn btn-lightgreen float-right mb-2" type="
    Reset Password
</button>
```

CAMEO




```
[:button {:style ($/combine $button-common
                            ($/bg-color :sig-orange))
          :on-click submit-login-info!}
 "Login"]])
```

CAMEO (FORGOT PASSWORD)

```
[:input {:class (style->class $/p-button)
         :style ($/combine [$/bg-color :sig-green] [$/align :b
         :type "button"
         :value "Request"
         :on-click submit-password-reset-request!}]]]]]])
```

THE GOAL

THE GOAL

- Stop re-inventing the wheel
- Make it easier to build, maintain UI's
- Make collaboration easier with designers / front-end engineers
- Easier Developer onboarding

NEW SIG

Pyrecast (Base) - https://adobe.ly/3Svebvn Palawan - https://adobe.ly/3zckxJi COMIMO - https://adobe.ly/3szGjTe

SIG COMPONENTS

- Reusable UI components (with BEM syntax)
- Interface to view existing UI Components (and props)
- Theme-able via BEM class names
- Address Accessibility requirements (Screen Readers, Contrast)

BEHAVE COMPONENTS

- Storybook: https://bit.ly/sig-story
- Style Guide: https://bit.ly/bhp-style
- Git Repo: https://gitlab.com/sig-gis/behave-components

USING BEHAVE COMPONENTS (CSS)

```
cd <PROJECT-DIR>
git submodule add git@gitlab.com:sig-gis/behave-components.git
ln -s <PROJECT-CSS-DIR>/components.css $PWD/behave-components/

// styles.css
@import "./css/components";
```

USING BEHAVE COMPONENTS (CLJS)

```
; form.cljs
(ns your-app.form
  (require [behave.components.core :as c]))

(defn root-component []
  [:div
    [c/button {:label "Hello World!"}]])
```

USING BEHAVE COMPONENTS (JS/REACT)

Work in Progress

```jsx
// Form.jsx
import React from 'react';
import {Button} from ./behave-components/react.js

class Car extends React.Component {
  render() {
    return <div>
             <Button label="Hello World"/>
           </div>;
  }
}
```

BEHAVE COMPONENTS

- Button (`c/button`)
- Card (`c/card`)
- Icon (`c/icon`)
- Progress Indicator (`c/progress-indicator`)
- Inputs
  - Checkbox (`c/checkbox`)
  - Dropdown (`c/dropdown`)
  - Number Input (`c/range-input`)
  - Radio Group (`c/radio-group`)
  - Radio Input (`c/radio-input`)
  - Range Input (`c/range-input`)
  - Text Input (`c/text-input`)
- Tab (`c/tab`)

CSS VARIABLES

THEME COLORS

- Colors should be given a human-readable CSS variable name, preferably less than 10 characters.
- If multiple hues of color are needed, add a number with the number increasing as the color darkens.

```
:root {
  --red: #FF0000;
  --bluegrey-1: #FAFAFA;
  ...
  --bluegrey-10: #2B3238;
}
```

STATEFUL COLORS

```
:root {
  --check-mark-green: #6CBC00;
  --sucess: #91D634;
  --warning: #FF7600;
  --error: #F51818;
}
```

FONT SIZES

```
:root {
  ...

  /* Fonts */
  --font-family: 'Roboto', 'sans-serif';

  /* Font Sizes */
  --font-size-10: 10px;
  --font-size-12: 12px;

  ...
  /* Font Weight */
  --font-weight-300: 300;
  --font-weight-500: 500;
  --font-weight-bold: bold;
```

BLOCK-ELEMENT-MODIFIER (BEM)

- In place of nested selectors (e.g. `.tab .label {}`), one selector with the block & element (e.g. `.tab__label {}`).
- When a modifier is introduced, use nested selectors to modify a block's elements (e.g. `.tab--small .tab__label { ... }`)

```
// Block is the top level 'component'
.button { }

// Elements are within a block and use '__' to nest within the
.button__icon {}
```

```
// Modifiers use '--' to alter the block
.button--small {}
.button--medium {}
.button--large {}
```

BUILDING COMPONENTS

- Ensure all nested elements have a class (e.g. `.button__label`)
- Modifiers should be block-level, not element-level
- States should be mapped to modifiers

BUILDING COMPONENTS WITH STATES (CLJS)

```clojure
(defn input-text [{:keys [label error?]}]
  [:div {:class ["input-text"
                 (when error?) "input-text--error"]}
   [:div {:class "input-text__label"} label
    [:input {:class "input-text__input"
             :type "text"}]]])
```

BUILDING COMPONENTS WITH STATES (CSS)

```
// Base style for the label within a text input
.input-text__label {
  color: black;
}

// When the "--error" modifier is applied to the block, the "_
// can be styled for that state.
.input-text--error .input-text__label {
  color: red;
}
```

EXAMPLE COMPONENT STATES (CSS)

```
.component--active {}
.component--disabled {}
.component--error {}
```

STORYBOOK

- Enables components to be viewed by Stakeholders, Designers & Team members
- Previews how Props to a Component
- Provides a single place for onboarding Team Members to UI Development

DEMO

https://bit.ly/bhp-story

WRITING A STORYBOOK STORY

- One story per component
- Can have story for components made up of other components (e.g. tab groups)
- Sidebar is organized via the `"title"` attribute (e.g. `Inputs/Checkbox` and `Inputs/Radio` will be grouped under `Inputs`)

WRITING A STORY PT. 1: SETTING IT UP

```
(ns behave.stories.checkbox-stories
  (:require [behave.components.core :refer [checkbox]]
            [behave.stories.utils :refer [->default]]
            [reagent.core :as r]))

(def ^:export default
  #js {:title     "Inputs/Checkbox"
       :component (r/reactify-component checkbox)})
```

WRITING A STORY PT. 2: DEFINING PROPS

```
(defn template [& [args]]
  (->default {:component checkbox
              :args      (merge {:label "Checkbox"
                                 :checked? false
                                 :disabled? false
                                 :error? false
                                 :on-change #(js/console.log "
                                args)
              :argTypes  {:on-change {:action "Changed!"}}}))
```

WRITING A STORY PT. 3: DEFINING STATES

```
(def ^:export Default  (template))
(def ^:export Checked  (template {:checked? true}))
(def ^:export Disabled (template {:disabled? true}))
(def ^:export Error    (template {:error? true}))
```

WRITE YOUR STORY