Component Driven Development

...

Components

What is a component?

A component is a uniquely identifiable part, piece, assembly or subassembly, system or subsystem.

Think of a component like a tangible object such as a wheel. A wheel is part of a car but is also made up of other components such as a tire, valve and rim. A car utilizes several wheels in order to move forward relying on other components for propulsion.

What is a web component?

Web components are a suite of various technologies that allow for the construction of custom elements with encapsulated functionality for use in web documents.

What browsers support web components?

Desktop:

- Chrome 67+
- Firefox 63+
- Edge 79+
- Safari (limited) 10.1+

Mobile:

- Chrome for Android 80+
- Samsung Internet 6.2+
- iOS Safari (limited) 10.3+




What is a single file component (SFC)?

A single file component contains all of its template, styles and scripts in a single file.

Demo - SFC & Web Components

https://github.com/RyanLafferty/component-driven-development-demo/tree/master/sfc-demo

CDD

What is Component Driven Development (CDD)?

- Component Driven Development is a development methodology that focuses the build process around components.

What are some problems which CDD solves?

- Can help increase collaboration between designers and developers through the use of shared component libraries and component explorers
- Allows work to be broken up into smaller tickets, enabling the team to build new pages and containers in parallel
- Enables engineering departments to scale as teams can collaborate to create these shared component libraries

Benefits to Component Driven Development

- Reusable
  - Components can be shared across application using shared component libraries
- Modular
  - Easy to swap components
- Test Driven
  - Since components should be atomic and highly cohesive, testing them should be very easy
- Themable
  - Can build a unified look across applications

Best Practices

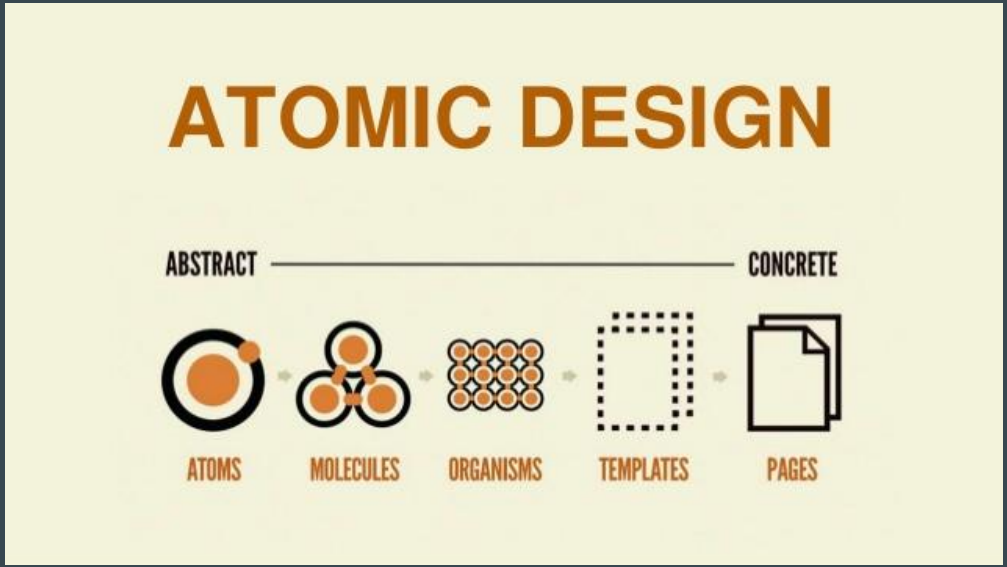
- High Cohesion
  - The component should perform a single, well defined task
- Loose Coupling
  - The component should have little to no knowledge of other components
- Reusable
  - Components should be built so that they may be reused and extended upon
- Easily Testable

What are some challenges which CDD presents?

- Increased cost in maintenance as we need to maintain these shared component libraries
- Additional points of failure and increased risk as bugs can be introduced through the shared component library
- An increased level of risk as any bug introduced when updating a component in a shared library can now affect multiple applications which rely on the library

How do we break up components?

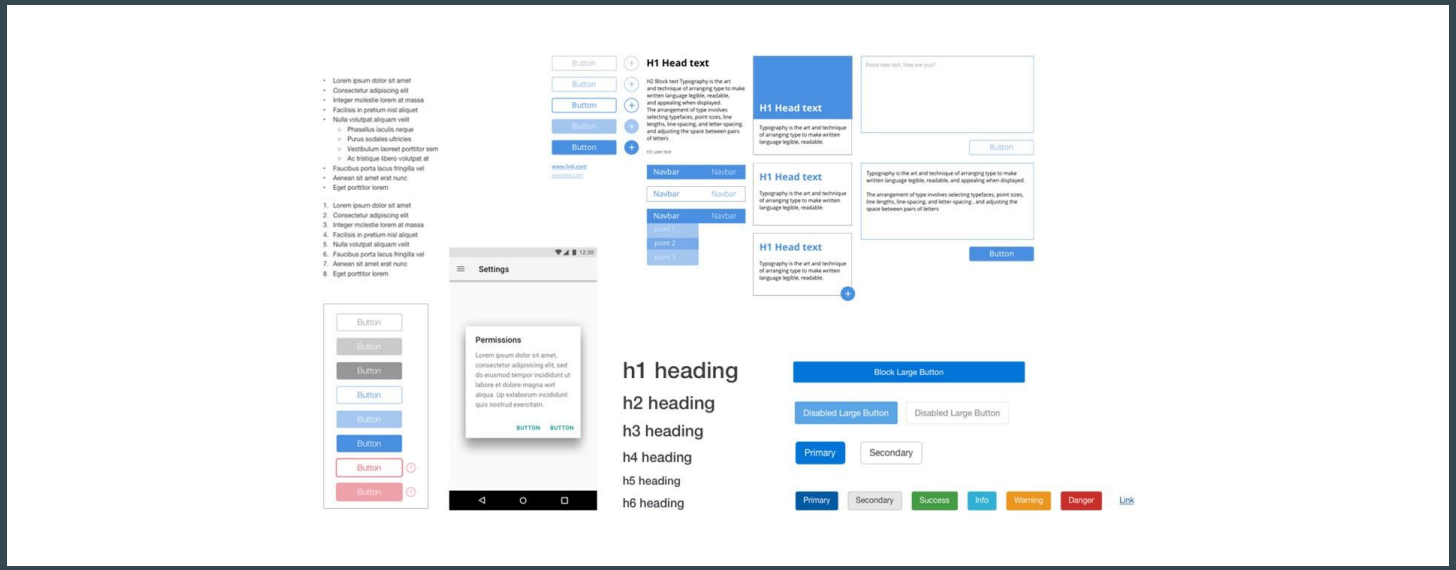
Components should be broken up in a manner that allows them to be reusable and easily testable. You should aim to create components that have one very specific task (high cohesion).



Component Types

What is a dumb / presentational component?

A component whose only responsibility is to present something to the DOM. It has no logic and is purely presentational.



What are containers?

A container is a logical construct created for component driven development. A container should be concerned with how things in the application work. It should be aware of the application's state and contain a number of child containers and / presentational components. A container should contain any logic required for your application and pass data and callbacks into the child containers and presentational components.

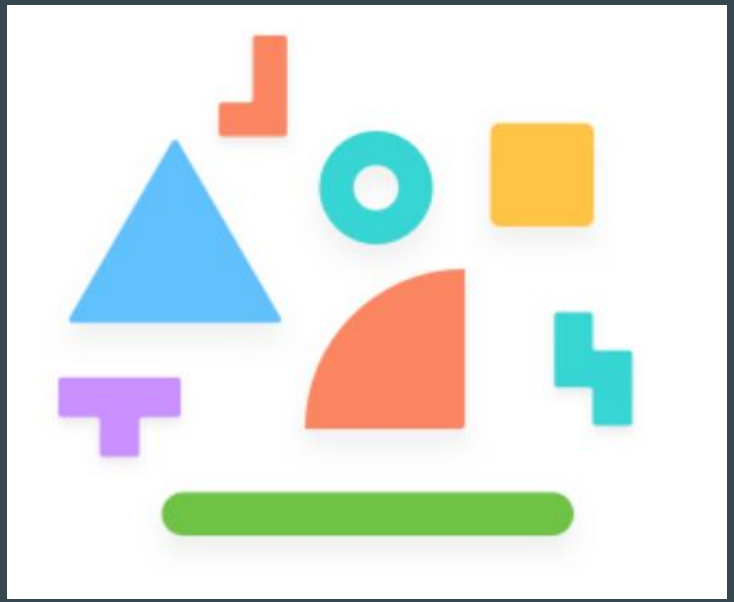
What are pages?

Pages are another logical construct that we have created for component driven development. They are a high order container component which contain a number of child containers and / or components.

Component Libraries

How can we break up component libraries?

Instead of importing an entire component library we can build our library in such a way that each component can be built and versioned as its own package.



Example Product - bit

bit is an IaaS (Infrastructure as a Service) product which enables component driven development without worrying about anything related to underlying infrastructure and operations.

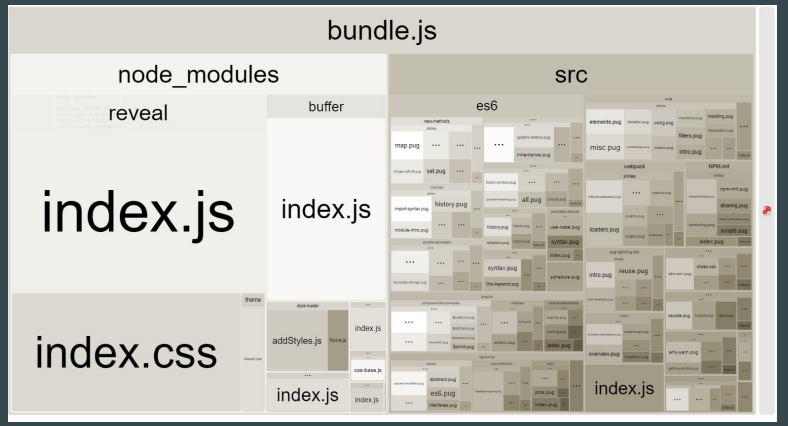
See: https://bit.dev/

How to identify issues in our library(ies) / application(s)?

We can use bundle analyzers to identify large assets and dependencies which are bloating our bundle.

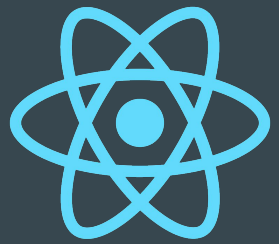
We can also use the analyzers to identify and address any duplicate dependencies that may be present in our bundle.

Example: [webpack-bundle-analyzer]



Popular Frontend Frameworks

- React
- Vue
- Angular
- Svelte




Styling

Inline CSS

Styles are defined directly on the element.

Pros

- No setup required

Cons

- Not reusable
- Can produce messy, hard to read and debug code

CSS Stylesheets

Stylesheets are imported and used in our components.

Pros

- No setup required

Cons

- Not reusable
- Potential conflicts

Sass & Scss Stylesheets

Sass / Scss stylesheets are preprocessed by node-sass before being imported and used in our components.

Pros

- Minimal setup required

Cons

- Potential conflicts
- Requires a dependency (node-sass) to translate scss / sass to css

CSS in JS

Styles are defined outside of the component and then injected into the exported component. This requires the use of a JS to CSS compiler in order to compile and inject the styles into the component.

Pros

- Minimal setup required
- Reusable
- Conflict Free

Cons

- Requires a dependency which will inject the styles into the component

Styled Components

An extension of CSS in JS. Styled components gives us css alongside some additional features used to create variables in javascript. These variables are used to extend a base set of style components.

Pros

- Minimal setup required
- Reusable
- Conflict Free

Cons

- Requires a dependency which will inject the styles into the component
- Reduced freedom

CSS Modules

CSS modules are a build process where a loader will translate the imported modules into locally scoped classes. This process ensures that the imported classes are conflict free.

Pros

- Reusable
- Conflict Free

Cons

- Extensive setup required
  - Must configure a loader

Sass & Scss Modules

An extension of css modules that uses another loader to translate the Sass and Scss modules into css modules. They are one of the best approaches to styling your components.

Pros

- Reusable
- Conflict Free

Cons

- Extensive setup required
  - Must configure multiple loaders

Demo - Styling

https://github.com/RyanLafferty/react-component-driven-development-demo/tree/master/demo/src/ComponentStylingMethods

Linting

What is a linter?

A linter, is a tool that analyzes source code to flag programming errors, bugs, stylistic errors, and suspicious constructs. - Wikipedia

Eslint

A tool used to lint javascript (js, jsx).

Configuration

Eslint uses the following file for configuration `.eslintrc`. The configuration is defined in JSON. It can be used to specify the rules that eslint will use against your codebase.

Environments

Environments set some eslint global variables based on the target environment. This includes javascript language feature sets. To use a preset we add the following parameter to our configuration `env`.

Presets

Presets are a way of defining a base set of rules for eslint to use. To use a preset we add the following parameter to our configuration `extends`.

Stylelint

A tool used to lint our styles (css, scss and sass).

Configuration

Stylelint's configuration is defined in the following file `.stylelintrc.json`. It can be used to specify the rules that stylelint will use against your codebase.

Demo - Linting

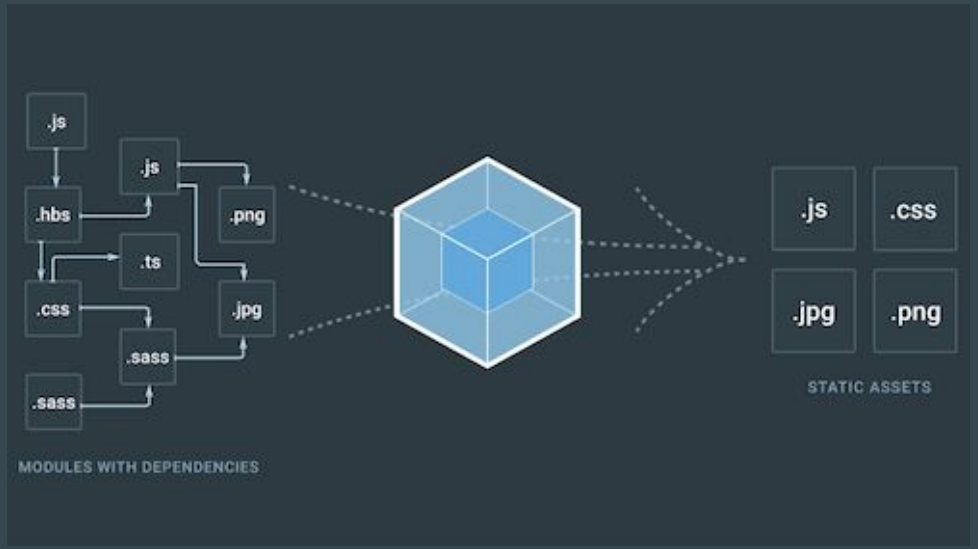
https://github.com/RyanLafferty/react-component-driven-development-demo/tree/master/demo

Building

What is a bundler?

A bundler is a tool that takes all of your dependencies (js, css, scss, png, etc) and packages them together. The output is one or more packages which can consist of styling sheets and scripts (we sometimes call these chunks as our code it to split into multiple bundles).

Example: Webpack



What is a loader?

A loader is a tool used by webpack that allows you to preprocess assets. This allows us to bundle assets beyond just simply javascript.

Example Loaders:

- babel-loader: Loads ES2015+ code and transpiles to ES5 using Babel
- file-loader: Emits the file into the output folder and returns the (relative) URL
- style-loader: Add exports of a module as style to DOM
- css-loader: Loads CSS file with resolved imports and returns CSS code
- sass-loader: Loads and compiles a SASS/SCSS file

Demo - Building

https://github.com/RyanLafferty/react-component-driven-development-demo/tree/master/demo

Presentation

How do we present our components?

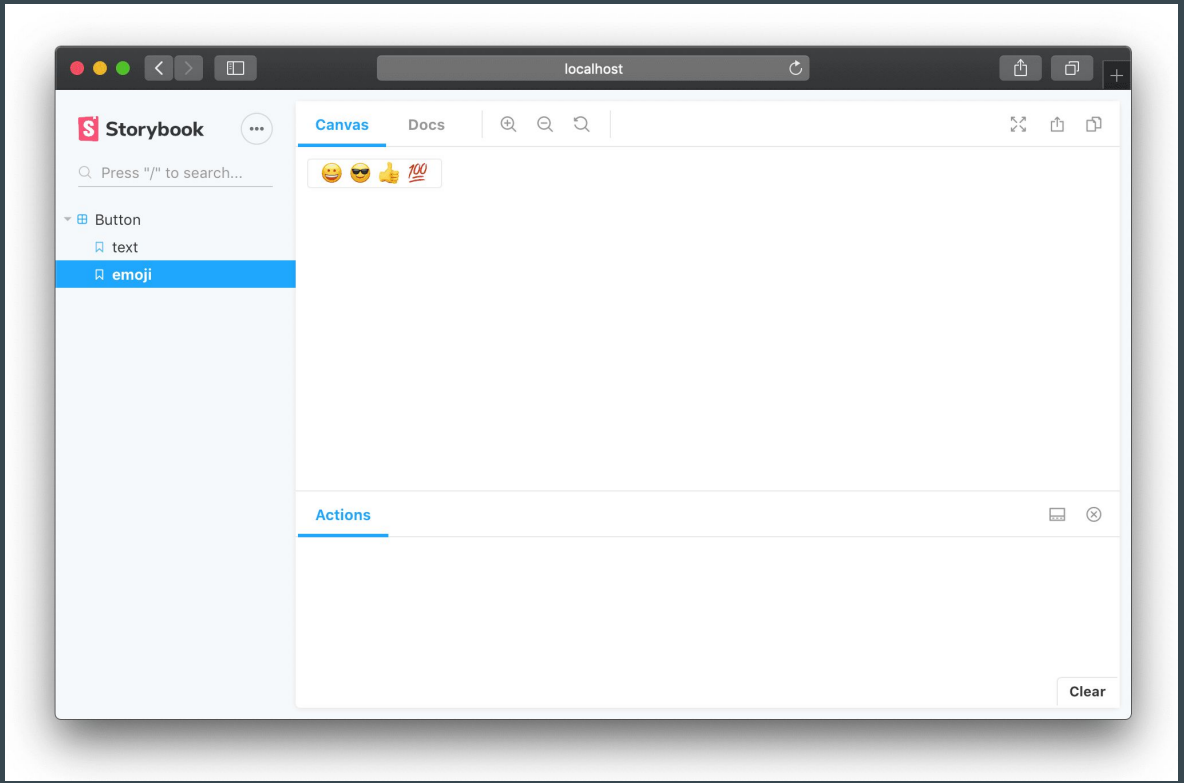
We present our components through the use of component explorers. They allow us to interact with our components independently and outside of the application.

Example - Storybook

A component explorer.

Supported Frameworks:

- React
- React Native
- Vue
- Angular
- Mithril
- Marko
- HTML
- Svelte
- Ember
- Riot
- Preact



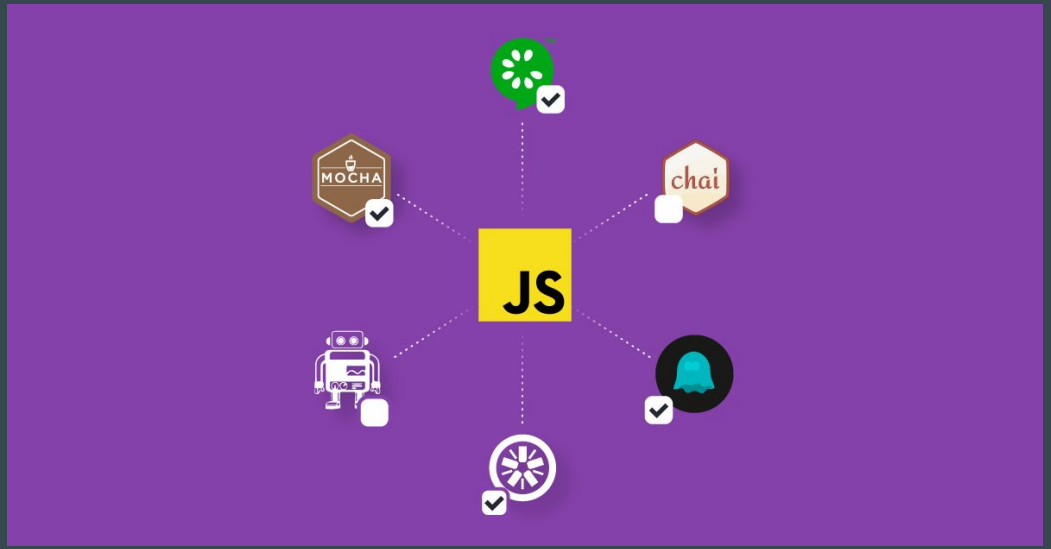
Demo - Component Explorers

https://github.com/RyanLafferty/react-component-driven-development-demo/tree/master/demo

Testing

What is a testing framework?

A testing framework is a set of rules which are used for creating test cases for use in automation.



Popular Javascript testing frameworks

- MochaJS
- Jasmine
- Jest
- Karma
- Puppeteer

What needs to be tested in a component?

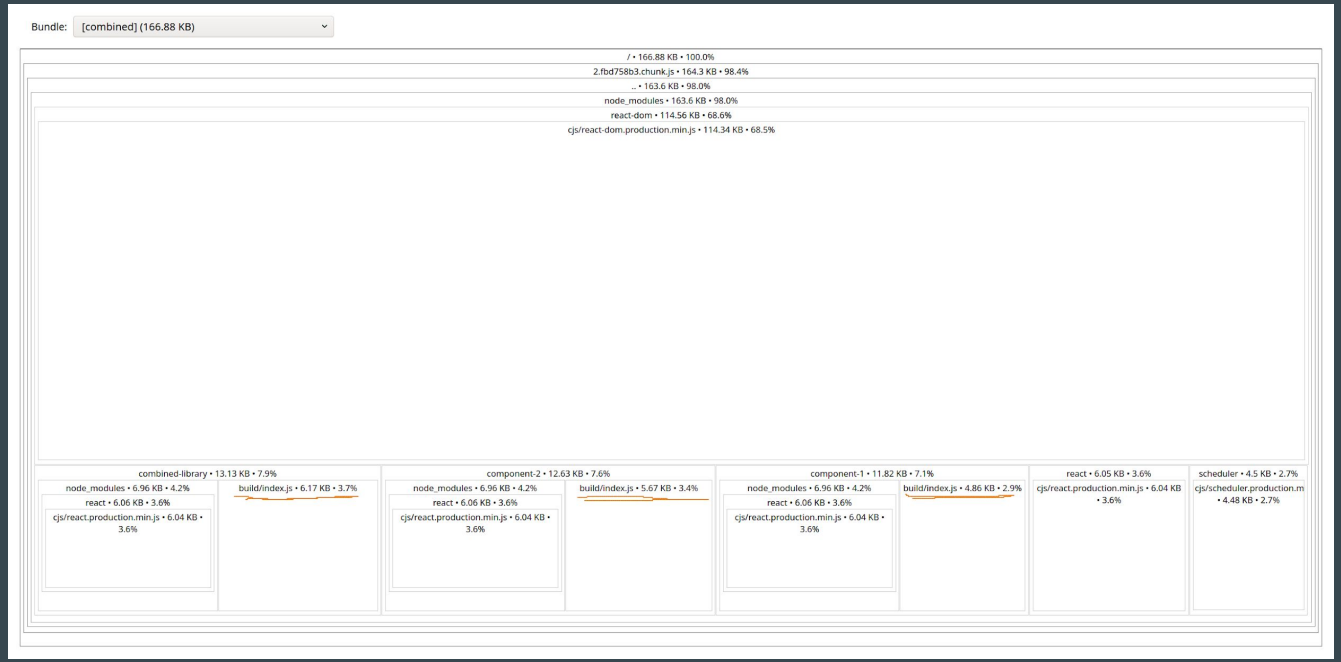
- Template
  - The structure of the component (snapshot testing)
    - Ensure that all variants of the component are covered
- Logic
  - Functionality needs to be tested within the component
    - Interactions (clicks, input, etc..)
      - Local state transitions
    - Local methods (validators, etc..)
    - Callbacks, Hooks
      - These should be mocked so that we can ensure that they are called as their logic should be tested separately outside of the component

To be continued...

Workshop - Component Driven Development

https://github.com/RyanLafferty/react-component-driven-development-demo/tree/master/workshop

Demo - Component Library Splitting



https://github.com/RyanLafferty/react-component-driven-development-demo/tree/master/react-component-library-breakdown-demo

Sources

- https://www.sitepoint.com/react-components-styling-options/
- https://webpack.js.org/
- https://stylelint.io/
- https://eslint.org/
- https://jestjs.io/
- https://developer.mozilla.org/en-US/docs/Web/Web_Components
- https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0
- https://www.npmjs.com/package/yorkie
- https://codys.club/blog/2015/07/04/webpack-create-multiple-bundles-with-entry-points/
- https://medium.com/the-s-curve/why-component-driven-design-drives-great-software-products-7cace364e815
- https://blog.hichroma.com/component-driven-development-ce1109d56c8e
- https://www.wikipedia.org/
- https://ryanlafferty.github.io/react-presentations/