**Introduction**

* + **Components**
    - * Any user interface can be broken into smaller parts
      * These parts are components of the UI.
      * A component is a reusable chunk of code.
      * Components make it possible to divide any UI into independent, reusable pieces and think about these pieces in isolation.
      * Like a pure function, a component should ideally do just one thing.
  + **React eliminates time from our code**
    - * React is declarative
      * We should be able to look at a component and have an idea of what is going to be rendered from it
      * Just thinking about snapshots in time (given this state, this is what the screen should look like)
      * Rather than describing the steps to reach a specifying the steps needed to reach a certain state, we just need to tell React which state we want to be in
      * States change based on user interaction
* **Separation of Concerns Introduction**
  + Common in other JS libraries (handlebars) and language templating engines
  + Needs of one component vs the needs of many
    - One component
    - Adding State
    - Continued Growth (3, 5, 10+ components)
    - Need for Global State
  + Wouldn’t put everything into our App.js
  + React was suppose to eliminate the need to separate our concerns
  + **Separation of Concerns** -  is a design principle for separating an application or component into distinct sections. Each section addressing a specific function.
  + **State Hoisting**
    - * Putting the state used by one component in a parent or grandparent component
      * Likely already using it with Apollo, Context, MobX, Flux, and / or Redux
      * Callback functions and dispatch methods

**Presentation and Container Components**

* + **Presentation Components (dumb components)**
    - * Concerned with how things look
      * Don’t contain state
      * Primary use case for functional components before hooks
        1. Functional components are a snapshot of their immutable values
      * May contain both Presentation and Container Components
      * Likely contain some DOM markup
      * Have no dependencies on global state (stores, context)
      * Don’t specify how the data is loaded or mutated
      * Receive data and callbacks exclusively via props.
  + **Container Components (smart components)**
    - * Are concerned with *how things work*.
      * Traditionally class components although that has changed with the introduction of hooks
      * May contain both Presentation and/or Container Components
      * Does not contain DOM markup
      * Uses local and global state to provide data to presentational components

**Benefits of separating our concerns**

* + **Maintainability**
  + **Reusability**
  + UI is no longer tied to its state, and thus can be re-used elsewhere
  + Prevents duplicate code leading to smaller bundles
  + **Composition**
  + Layout components leveraging this.props.children.
  + These wrap other components, and add to the DOM structure, but don’t influence their children in any other way
  + Components that have styling attached Card UI’s, styled header tags, field sets, etc
  + **Easier Optimizations**
  + using shouldComponentUpdate, a PureComponent, or memo (hooks) to control render
  + **Easier testing!**
  + Presentational components can be snapshot tested based on their props alone
  + Don’t have to worry about Container components creating situations that are difficult to test (portals, complex state UI, etc)

**Great in Theory but how does it work in practice?**

* + **Prop Drilling**
    - Component at the top gets most of the global data and disperses it down the tree
      * (Show diagram)
  + **Pub Sub**
  + Individual components get the information they need from global state
    - (Show diagram)

**Takeaways**

* + Remember, **components don’t have to emit DOM.**
  + They only need to provide composition boundaries between UI concerns
  + Testing and optimization are easier when concerns are separated
  + Everything talked about today is applicable with hooks and classes