




# Frontend (React)



17356 S21  
JJ, Sean



# Concept: Frontend

---

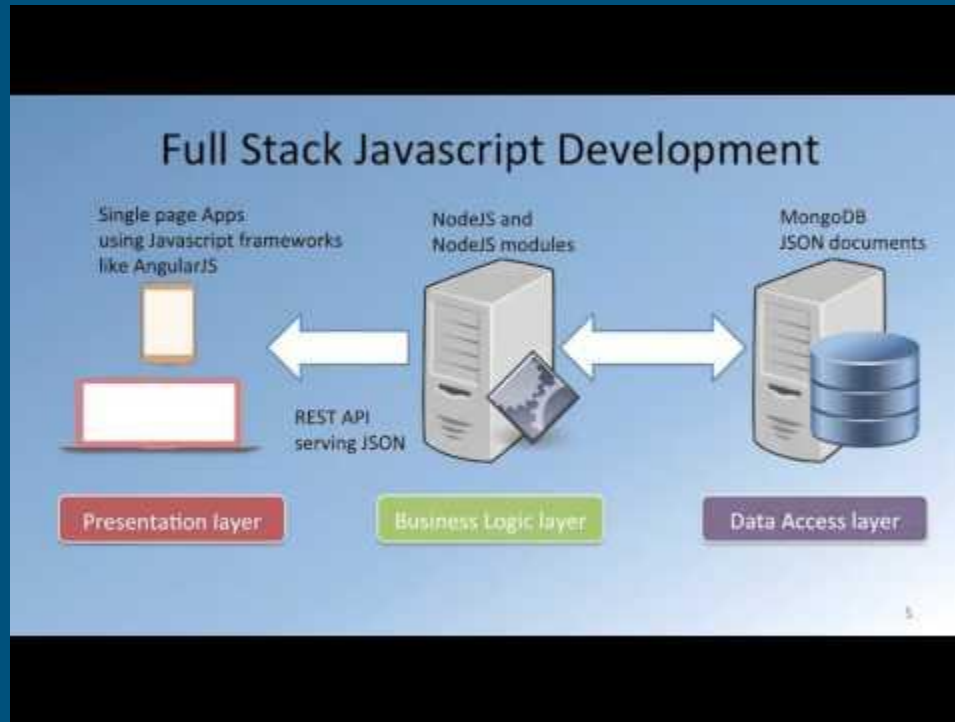
What the user sees + interacts with.

“Client-side code”

Probably know of HTML/CSS/JS. You can build vanilla frontends.

We will learn React (<https://reactjs.org>)

# Visual



# Interruption: Styling and CSS

---

We don't have a dedicated recitation for styling, because there are so many many systems you can follow.

Google is your friend.

My personal pick: Flexbox

[https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\\_layout/Flexbox](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox)

<https://www.youtube.com/watch?v=JJSoEo8JSnc>

# Intro to ReactJS (or just React)

---

Created 2011 (by Facebook)

“Frontend JS Library” (technically not a framework, but its chill)

Declarative, Component-Based

Uses JSX syntax (HTML inside your JS)

```
function getGreeting(user) {  
  if (user) {  
    return <h1>Hello, {formatName(user)}!</h1>;  
  }  
  return <h1>Hello, Stranger.</h1>;  
}
```

# React: Components + Props

---

“React Only Updates What’s Necessary”

Function vs Class components

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

You can nest components (intuitive)

STRICT Rule: “All React components must act like pure functions with respect to their props” (this does not apply to State)

# React: Components + State

---

Component State = saved (and usually important) information about a component

Changing state -> trigger a component reload

**Do not** modify state directly (will not trigger reload). Use React's state funcs.

# React: Data (States & Props) Flow

---

Parent-Child relationships (think Tree)

State is always local, but can flow downwards (to children) as props.

Common Workaround: pass a state-modifying function as prop to child.

- Child can then call the passed function to indirectly modify parent state.

Summary: State flows down (waterfall). Changes can *sometimes* flow back up.

(We will learn about state management libraries later on)



# React: Design Process

---

How to think + code like a React dev:

- Break UI into component list/hierarchy (form the Tree)
- Build static version of UI first
  - Compatible with data models, but no interactions
- Find simplest representation of UI state for each component
- Identify where state should live
- Add inverse data flow (changes go back up)

# Component Libraries/Frameworks

---

SUPER USEFUL (and fun to explore)

You no longer have to style everything by hand (b/c CSS bad).

Find one that you enjoy and read the docs on how to use it!

Popular ones:

- MaterialUI (google)
- Bootstrap
- Ant Design (Ant Financial, Alibaba)
- Evergreen

We're gonna use Geist UI (more obscure, to get used to learning weird things)