**An Overview of React JS**

1. History of React
2. Why React
3. 7 Key Trade Offs
4. Potential Issues with React

**History of React**

1. 2011 Created by Facebook
2. 2012 Used by Instagram
3. 2013 open sourced
4. 2014 Embraced by many large companies
5. 2015 React Native released
6. 2016 React 15 released (previous version was 0.14)

**Today:** Over 30K components at Facebook full time dev staff Used by many in fortune 500

**Why React?**

**1) Flexibility**

React is a library not a framework unlike Angular and Ember

***Where can I use react?***

* web apps/ static sites
* Mobile -React Native
* Desktop (can use electron to installable desktop app)
* Server Rendering using Next.js
* React for Virtual Reality websites and 360 experiences with React VR
* So Learn React Once and we can write Applications everywhere!!
* A low risk way to migrate to React by adding one component at a time-we can start with small portions
* Continue to run on all the browsers - Facebook cannot afford to run only on few browsers

***Different React renders***

1. react-dom for web apps
2. react-native to write native friendly code.
3. react-vr

**2) Developer experience**

Offers a simple API -few concepts to master

**3) Corporate Investment**

Facebook committed to react.

**4) Community**

Huge active Community Companies using react:

a) Apple b) microsoft c) Amazon twitter d) dropbox e) paypal f) slack g) netflix h) Tesla

***Overview of the eco system***

1. React Router
2. Redux
3. Mobx
4. Jest
5. GraphQL
6. Next.js

*It is a big DEAL! 70,000 stars on GitHub 1,100 + contributors Millions of downloads/month*

**5) Performance**

Updating the DOM is expensive! Without virtual DOM Blindly update DOM using new state with Virtual DOM Update the DOM in the most efficient way

React size ~35K

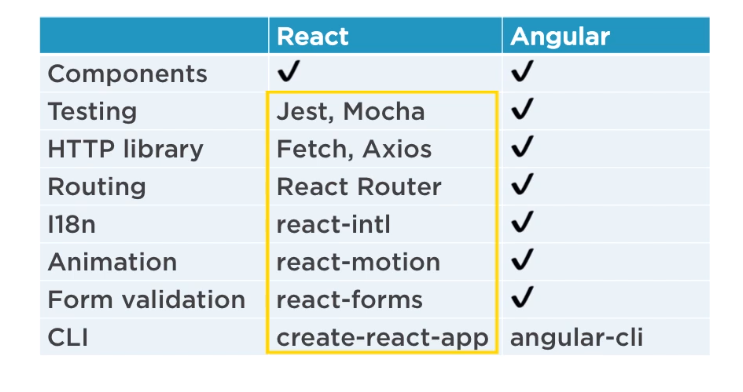
**6) Testability**

* 1. Little to no configuration required
  2. Run in memory via Node so, no browser required
  3. Fast
  4. reliable deterministic unit tests
  5. write quickly update easily

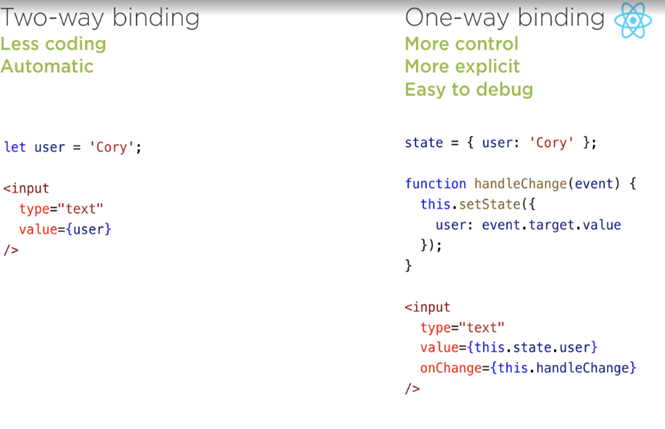
*Components are pure functions making testing easy (Reliable, Deterministic, no side effects) For React the most popular testing framework is JEST (create-react-app boilerplate)*

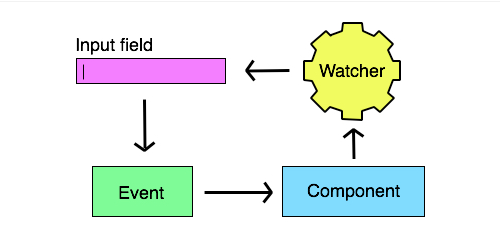
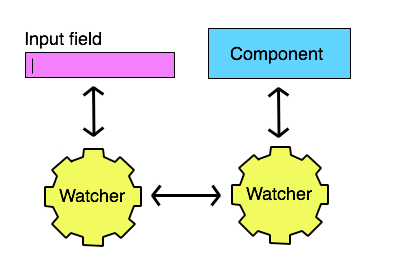
**7 Key Trade Offs**

**1) Framework vs library**

****

**2) Concise vs Explicit**

****

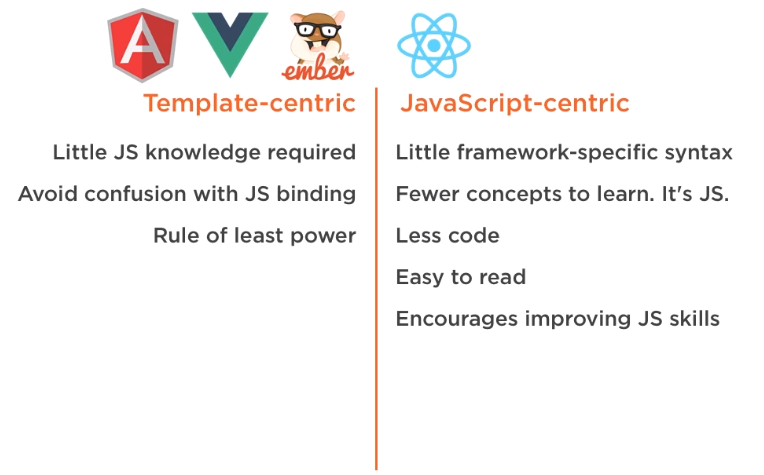
**One Way**

**Two Way**

**3) Template Centric vs Javascript Centric**

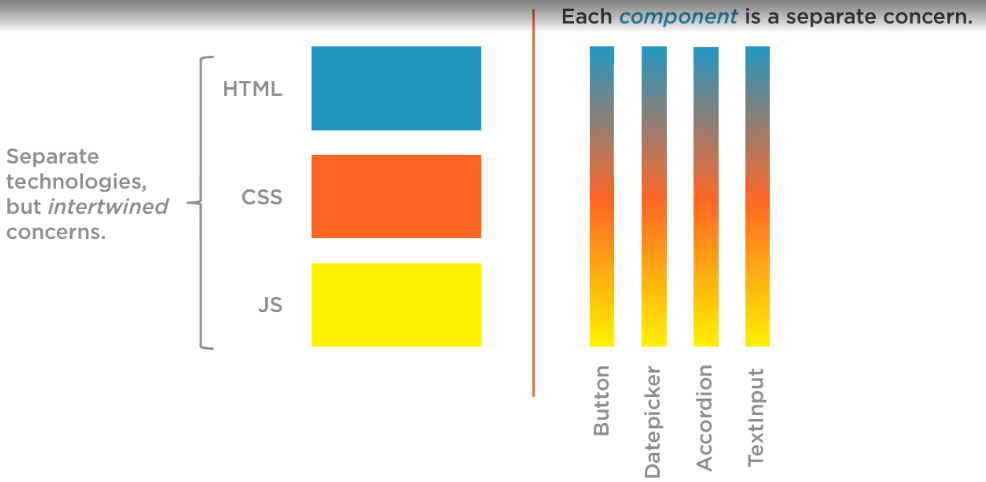
*React Js is Javascript Centric but other frameworks have their own unique syntax:*

* *Modules*
* *Let and Const*
* *Enhanced object literals*
* *Default Parameters*
* *Template Strings*
* *Classes*
* *Arrow Functions*
* *Promises*
* *Destructuring*
* *Spread Operator*

**

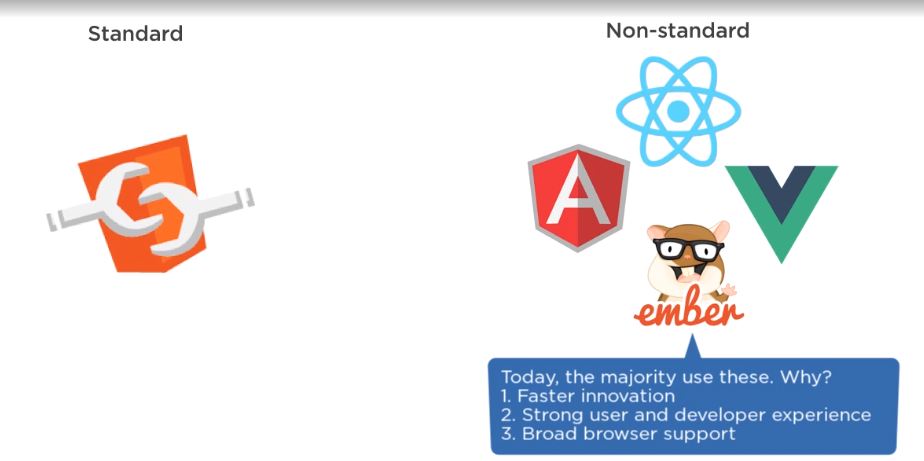
**

1. **Seperate Template vs Single File**

**Components can be composed eg: Navigation, Book List, Book Description etc.

1. **Standard vs non Standard**

React is one of many non-standard component libraries



1. **Community vs Corporate Backing**

React is driven by Facebooks needs, although other libraries are community driven but other than that, has:

1. Full time development staff
2. Over 1000 contributors
3. FB: World's 5th most valuable company
4. 30 K components in production

So With React We are,

***Getting...*** Explicit/ Javascript centric /Single file component/ Non Standard /Corporate Backed /Library ***and Giving up..***.concise /Template centric/Seperate template/standard/Community based framework

**4] Potential Issues with React**

**1) JSX and HTML Difference**

Options to Convert HTML to JSX

1. find/replace
2. Online Compiler
3. htmltoJSX on npm for large files

**2) Build Step required**

As we need to compile down JSX to JS but no matter what JS framework we use for web apps a build step is critical

1. minify the code to save band width
2. Transpile the code so that we can use modern JS features
3. Lint our code and run automated tests

What are 2 transpilers Compile JSX?

1. Babel
2. TypeScript

*There is a variety of React -boiler-plates to get started and to have build steps built in -automatically-to transpile JSX for us ,create-react-app is the most popular option.*

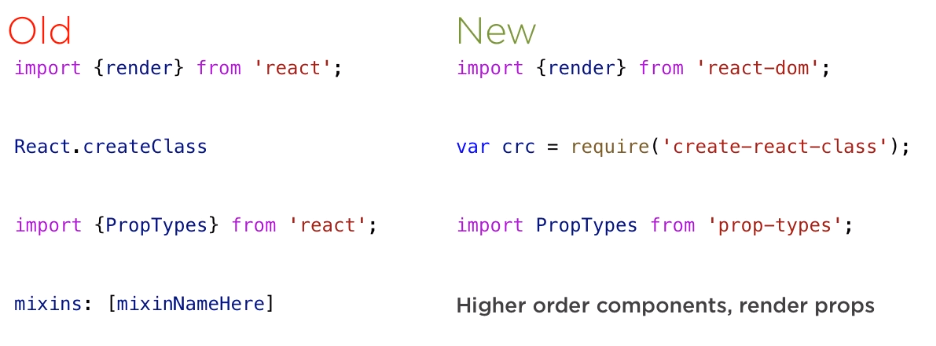
**3) Version Conflicts**

* In React JS there is a runtime and we cannot have 2 versions of react on the same page
* We will be using other libraries like react-router with react (eg: React router needs React 15+)
* FB is consistent about releasing codemods when breaking releases occur -upgrades to existing react components can be easily automated

**How to Avoid Conflicts:?**

* Standardize on a version
* Upgrade React when upgrading libraries
* Upgrade as a team

**4) Old stuff on Searches (outdated Resources)**

****

**5) Decision Fatigue**

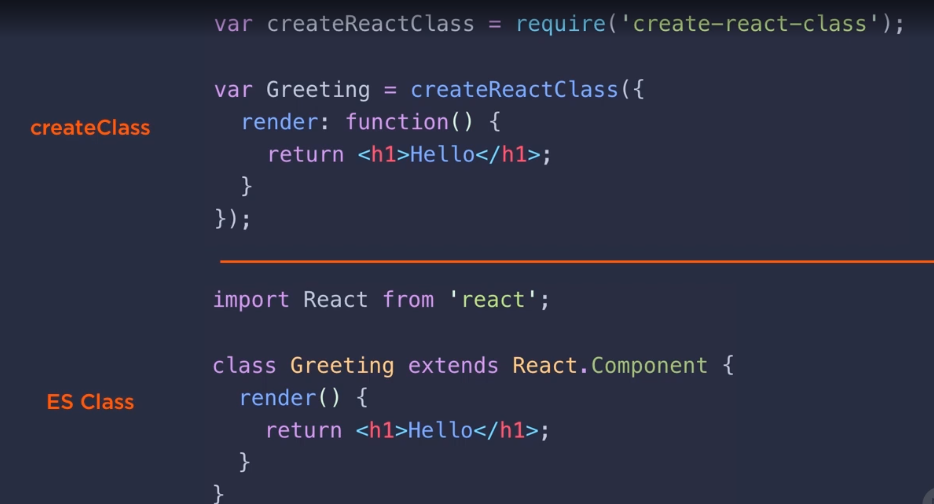
1) Dev environment

We can visit andrewhfarmer.com/starter-project to select one of the dev environment

create-react-app boiler plate-popular

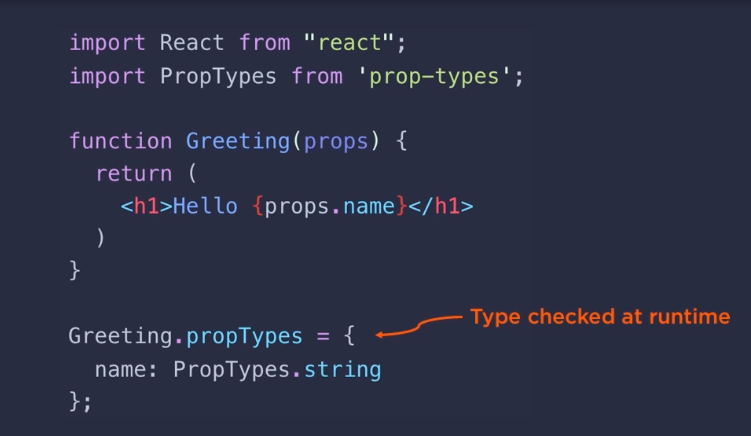
React-router

2) ES class or Create Class



3) Types

***Prop types***



***Type Script***



***Flow - A different way-we add annotations***

4) State ( App's data)

***Plain React***

***Flux***

***Redux (most popular )-centralized state***

***Mobx (observable state)***

5) Styling