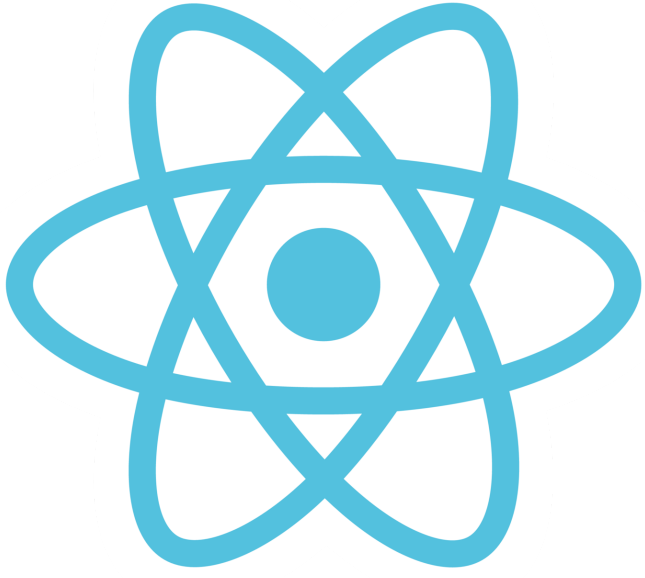


React

Introduction to React

- What is React?
- Setup react app
- Write a hello world component
- render method
- JSX
- Props
- State
- Use child components
- Access nested data with props.children
- Using ref
- React event system
- Component Life Cycle
- Manage state with Life cycle methods
- Control state when new props received
- Use map to create array of components
- Using ternary operator to render conditional JSX
- Higher order components
- Understand React.Children utilities
- Use React.cloneElement
- Use composable APIs to build reusable components
- Using react developer tools in chrome

What is React?

- Developed by Facebook
- React is a view layer library, not a framework like Backbone, Angular etc.
- You can't use React to build a fully-functional web app

Speaker notes

To make changes to HTML page, we use JavaScript and updates DOM. React only does minimal DOM changes to make application running fast.

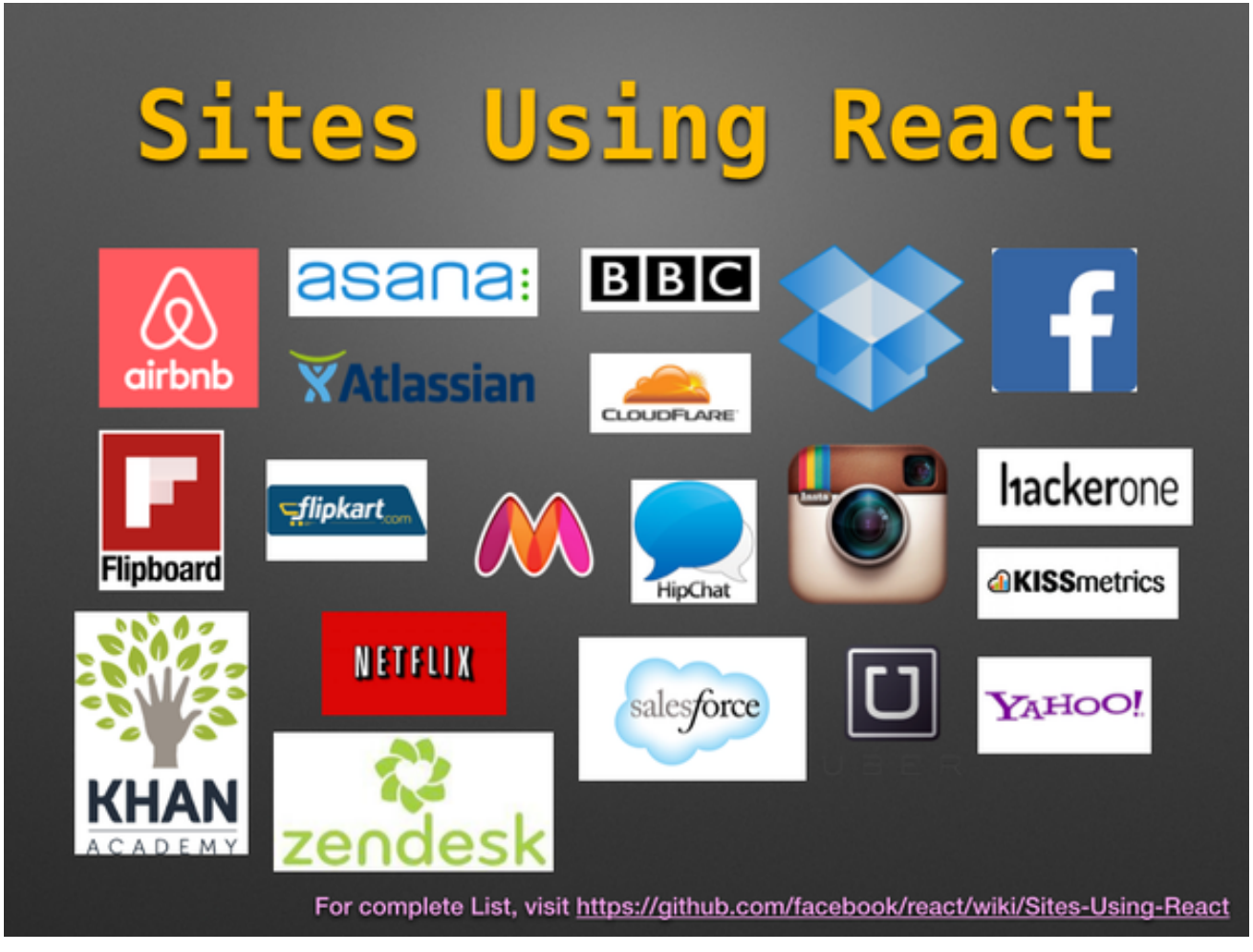
Why was React developed?

- Complexity of two-way data binding
- Bad UX from using "cascading updates" of DOM tree
- A lot of data on a page changing over time
- Complexity of Facebook's UI architecture
- Shift from MVC mentality

Why should I use React?

- Easy to read and understand views
- Concept of components is the *future* of web development
- If your page uses a lot of fast updating data or real time data - React is the way to go
- Once you and your team is over the React's learning curve, developing your app will become a lot faster

What sites using react?



How to know if site using react?

React-Detector chrome plugin

https://chrome.google.com/webstore/detail/react-detector/jaaklebbenondhkanegppccanebkdjlh

This plugin allow you to find whether react being used in the website or not.

https://github.com/facebook/react/wiki/sites-using-react

Fundamentals

Most important terms in React

Component

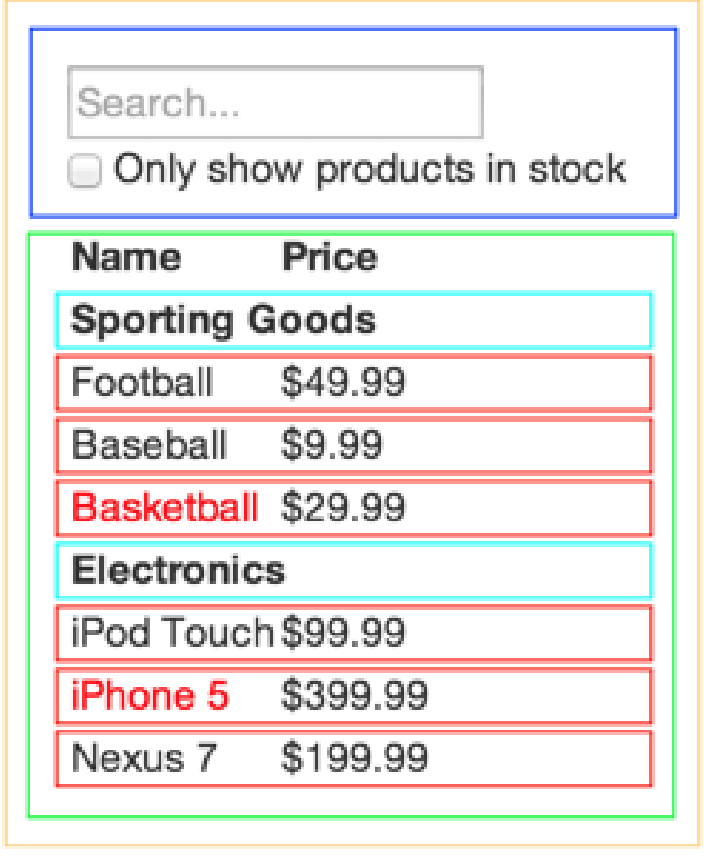
Component

Components are self-contained reusable building blocks of web application.

React components are basically just idempotent functions (same input produces same output).

They describe your UI at any point in time, just like a server-rendered app.

Component



1. **FilterableProductTable (orange):** Contains the entirety of the example
2. **SearchBar (blue):** receives all *user input*
3. **ProductTable (green):** displays and filters the *data collection* based on *user input*
4. **ProductCategoryRow (turquoise):** displays a heading for each *category*
5. **ProductRow (red):** displays a row for each *product*

Component

- Created using ES6 Class Syntax
- The only required method is render()

```
import React, { Component } from 'react';

class App extends Component {
  render(){
    return <h1>Hello World</h1>
  }
}

export default App;
```

Component

Inserted into DOM using ReactDOM.render()

```
port React from 'react';
port ReactDOM from 'react-dom';
port App from './App';


actDOM.render(<App />, document.getElementById('root'));
```

Props

Props

- Data passed down to component from parent component and represents data for the component
- accessed via **this.props**
- **props** are read only

```
render() {
    var someProp = 'bar';

    console.log('component render()', this.props);

    return <div>
        <AnotherComponent foo={someProp} model={this.props.model}
    </div>;
}
```

State

State

- Represents internal state of the component
- Accessed via **this.state**
- When a component's state data changes, **render()** method will be executed and updates UI with new data

```
render() {
    return <h3>Click count:
        <span>{this.state.clicks}</span>
    </h3>;
}
```

JSX

JSX

- Arguably, one of the coolest things in React
- XML-like syntax for generating component's HTML
- Easier to read and understand large DOM trees
- Translates to plain JavaScript

```
/** @jsx syntex */

render() {
    return (
            <h1 className="greeting">
                Hello, world!
              </h1>
           )
}

/** plain javscript syntex */

render() {
    return React.createElement('h1',
                                    {className: 'greeting'},
                                    'Hello, world!'
                                  );
}
```

Setup React Environment

Setup React Environment

In order to setup environment, we need to use create-react-app npm package to start new react app project

Setup React Environment

In order to setup environment, we need to use create-react-app npm package to start new react app project

Install create-react-app

Setup React Environment

In order to setup environment, we need to use create-react-app npm package to start new react app project

Install create-react-app

```
$ npm install -g create-react-app
```

Setup React Environment

In order to setup environment, we need to use create-react-app npm package to start new react app project

Install create-react-app

```
$ npm install -g create-react-app
```

Create new react project "helloworld"

Setup React Environment

In order to setup environment, we need to use create-react-app npm package to start new react app project

Install create-react-app

```
$ npm install -g create-react-app
```

Create new react project "helloworld"

```
$ create-react-app helloworld
```

Setup React Environment

In order to setup environment, we need to use create-react-app npm package to start new react app project

Install create-react-app

```
$ npm install -g create-react-app
```

Create new react project "helloworld"

```
$ create-react-app helloworld
```

Run npm start to run helloworld react app

Setup React Environment

In order to setup environment, we need to use create-react-app npm package to start new react app project

Install create-react-app

```
$ npm install -g create-react-app
```

Create new react project "helloworld"

```
$ create-react-app helloworld
```

Run npm start to run helloworld react app

```
$ cd helloworld && npm start
```

Setup Full Stack Environment

In order to setup full stack environment, we need to use concurrently npm package to start both Node and React's Webpack server to auto reload both applications on file changes.

DEMO

Basic Component

```
import React, { Component } from 'react';

class App extends Component {
  render(){
    return <h1>Hello World</h1>
  }
}
```

Set Props on Component

We can pass data into our components by using what's called **props**.

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <App txt="this is the prop text" />
  document.getElementById('root');
);
```

app.js

```
import React from 'react';

class App extends React.Component {
  render(){
    return <h1>{this.props.txt}</h1>
  }
}
export default App
```

PropTypes

We can define the properties that we're going to be looking for in our component by adding a property to our component called PropTypes.

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <App txt="this is the prop text" />
  document.getElementById('root');
);
```

app.js

```
import React from 'react';

class App extends React.Component {
  render(){
    return <h1>{this.props.txt}</h1>
  }
}

App.propTypes = {
  txt: React.PropTypes.string,
  cat: React.PropTypes.number
}

export default App
```

PropTypes

PropTypes allow you to declare the "type" (string, number, function, etc) of each prop being passed to a component. Then, if a prop passed in isn't of the declared type, you'll get a warning in the console.

- string
- number
- func (for functional prop)
- bool

isRequired

On each of these PropTypes, we can add an isRequired to it to make sure props are passed to component otherwise react throw error in console

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <App txt="this is the prop text" />
  document.getElementById('root');
);
```

app.js

```
import React from 'react';

class App extends React.Component {
  render(){
    return <h1>{this.props.txt}</h1>
  }
}

App.PropTypes = {
  txt: React.PropTypes.string,
  cat: React.PropTypes.number.isRequired
}

export default App
```

Add Custom propType Validation to React Components

In addition to the types built into React.propTypes we can also define our own custom propType validator

```
class App extends React.Component {
  render(){
    return <Title text="12345678" />
  }
}

const Title = (props) => <h1>Title: {props.text}</h1>

Title.propTypes = {
  text(props, propName, component){
    if(!(propName in props)){
      return new Error(`missing ${propName}`)
    }
    if(props[propName].length < 6){
      return new Error(`${propName} was too short`)
    }
  }
}

export default App;
```

defaultProps

On each of these PropTypes, we can add an isRequired to it to make sure props are passed to component otherwise react throw error in console

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <App txt="this is the prop text" />
  document.getElementById('root');
);
```

app.js

```
import React from 'react';

class App extends React.Component {
  render(){
    return <h1>{this.props.txt}</h1>
  }
}

App.PropTypes = {
  txt: React.PropTypes.string,
  cat: React.PropTypes.number.isRequired
}

App.defaultProps = {
  txt: "this is the default txt"
}

export default App;
```

state

State is used for properties on a component that *will change*, versus static props that are passed in.

```
import React from 'react';

class App extends React.Component {
  constructor(){
    super();
    this.state = {
      txt: 'this is the state txt',
      cat: 0
    }
  }
  update(e){
    this.setState({txt: e.target.value})
  }
  render(){
    return (
      <div>
        <input type="text"
          onChange={this.update.bind(this)} />
          <h1>{this.state.txt} - {this.state.cat}</h1>
      </div>
    )
  }
}

export default App
```

Children

React component can output or render other React components

```
import React from 'react';

class App extends React.Component {
  constructor(){
    super();
    this.state = {
      txt: 'this is the state txt',
      cat: 0
    }
  }
  update(e){
    this.setState({txt: e.target.value})
  }
  render(){
    return (
      <div>
          <h1>{this.state.txt} - {this.state.cat}</h1>
          <Widget update="this.update(this).bind(this)"/>
          <Widget update="this.update(this).bind(this)"/>
      </div>
    )
  }
}
//Stateless functional component
const Widget = (props) =>
  <input type="text" onChange={props.update} />

export default App
```

React Stateless Functional Components

React 0.14 introduced a simpler way to define components called stateless functional components. These components use plain JavaScript functions

- Easy to write
- Preferable for only UI/Presentational components where state is not needed
- Improved performance

React Stateless Functional Components

```jsx
import React from 'react';

class HelloWorld extends React.Component {
  constructor(props) {
    super(props);
  }

  sayHi(event) {
    alert(`Hi ${this.props.name}`);
  }

  render() {
    return (
      <div>
        <a
          href="#"
          onClick={this.sayHi.bind(this)}>Say Hi</a>
      </div>
    );
  }
}

HelloWorld.propTypes = {
  name: React.PropTypes.string.isRequired
};

export default HelloWorld;
```

```jsx
import React from 'react';

const HelloWorld = ({name}) => {
  const sayHi = (event) => {
    alert(`Hi ${name}`);
  };

  return (
    <div>
      <a
        href="#"
        onClick={sayHi}>Say Hi</a>
    </div>
  );
};

HelloWorld.propTypes = {
  name: React.PropTypes.string.isRequired
};

export default HelloWorld;
```

Access Nested Data with Reacts props.children

In order to access nested values or components in a component, we can use props.children.

```
class App extends React.Component {
  render(){
    return <Button>I <Heart /> React</Button>
  }
}

const Button = (props) =>
  <button>{props.children}</button>

class Heart extends React.Component {
  render(){
    return <span>♥</span>
  }
}
```

Creating lists in with .map

You can build collections of elements and include them in JSX using curly braces {}

```
constructor(){
  super();

  this.state={
      name:"Varma Bhupatiraju",
      friends:['Ram', 'Robert', 'Rahim']
  }
}

render() {
  return (
    <div>
      <h3> Name: {this.state.name} </h3>
      <ul>
        {
            this.state.friends.map((friend)=>{
              return <li> {friend} </li>;
            })

        }
      </ul>
    </div>
```

Reacts Synthetic Event System

React has its own event handling system which is called **Synthetic Events**. Synthetic Events is a cross-browser wrapper of the browser's native event. It works the same way as the event system that you find on browsers, the only difference is that the same code will work across all browsers.

```
class App extends React.Component {
  constructor(){
    super();
    this.state = {currentEvent: '---'}
    this.update = this.update.bind(this)
  }
  update(e){
    this.setState({currentEvent: e.type})
  }
  render(){
    return (
      <div>
        <textarea
          onKeyPress={this.update}
          onCopy={this.update}
          onCut={this.update}
          onPaste={this.update}
          onFocus={this.update}
          onBlur={this.update}
          onDoubleClick={this.update}
          onTouchStart={this.update}
          onTouchMove={this.update}
          onTouchEnd={this.update}
          cols="30"
          rows="10" />
        <h1>{this.state.currentEvent}</h1>
      </div>
    )
  }
}
export default App;
```

Understand the React Component Lifecycle Methods

When our component is added to the DOM, this is called **mounting**, and when our component is removed from the DOM, this is called **unmounting**.

```
class App extends React.Component {
 constructor(){
    super();
    this.state = { val: 0 };
    this.update = this.update.bind(this);
  }
  update(){
    this.setState({val: this.state.val + 1 })
  }
  componentWillMount(){
    console.log('mounting')
  }
  render(){
    console.log('rendering!')
    return <button onClick={this.update}>{this.stat
  }
  componentDidMount(){
    console.log('mounted')
  }
  componentWillUnmount(){
    console.log('bye!')
  }
}
export default App;
```

Manage React Component State with Lifecycle Methods

In componentWillMount, we have access to our state and our props but we do not have access to the DOM representation of our component yet because it has not been placed into the DOM. In componentDidMount, we have access to our component in the DOM

```
class App extends React.Component {
 constructor(){
    super();
    this.state = { val: 0 };
    this.update = this.update.bind(this);
  }
  update(){
    this.setState({val: this.state.val + 1 })
  }
  componentWillMount(){
    console.log('mounting')
  }
  render(){
    console.log('rendering!')
    return <button onClick={this.update}>{this.state.val}</button>
  }
  componentDidMount(){
    console.log('mounted')
  }
  componentWillUnmount(){
```

Control React Component Updates When New Props Are Received

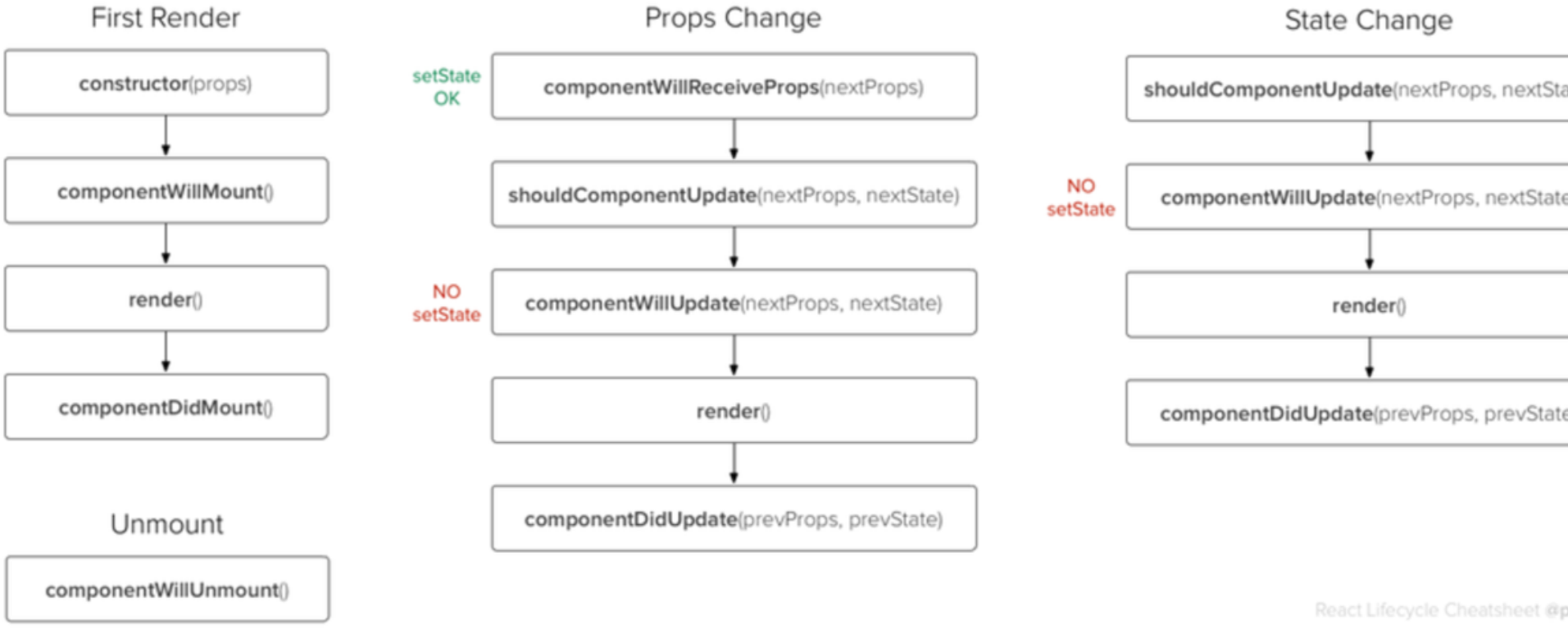
The React component lifecycle will allow you to update your components at runtime.

componentWillReceiveProps gives us an opportunity to update state by reacting to a prop transition before the render() call is made.

shouldComponentUpdate allows us to set conditions on when we should update a component so that we are not rendering constantly.

componentDidUpdate lets us react to a component updating.

React Life Cycle Flow Chart



Fetching data with Fetch API

The fetch API is a promise-based API that returns a response object. In order to get to the actual JSON content, you need to invoke the json() method of the response object.

```
fetch(QUOTE_SERVICE_URL)
      .then(response => response.json())
      .then((result) => {
            this.setState({quotes: result, isFetching: false}
       }
      .catch(e => console.log(e));
```

Forms

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with setState().

*An input form element whose value is controlled by React in this way is called a "**controlled component**"*

```
//------
handleChange(event) {
  this.setState({value: event.target.value});
}

handleSubmit(event) {
  alert('A name was submitted: ' + this.state.value);
  event.preventDefault();
}

render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Name:
        <input type="text" value={this.state.value} onChange={this.handleChange} /
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
```

```
//------
handleChange(event) {
  this.setState({value: event.target.value});
}

handleSubmit(event) {
  alert('A name was submitted: ' + this.state.value);
  event.preventDefault();
}

render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Name:
        <input type="text" value={this.state.value} onChange={this.handleChange} /
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

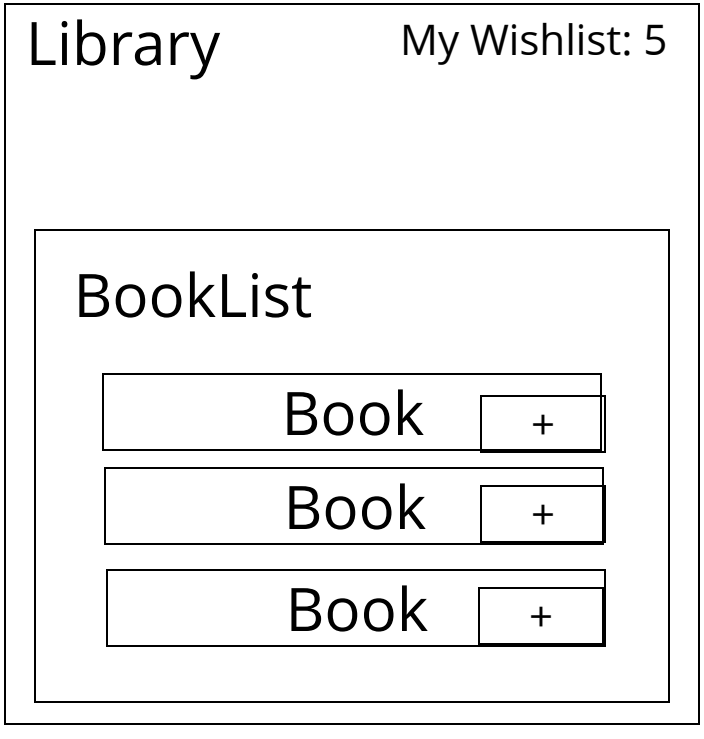
React Data Flow

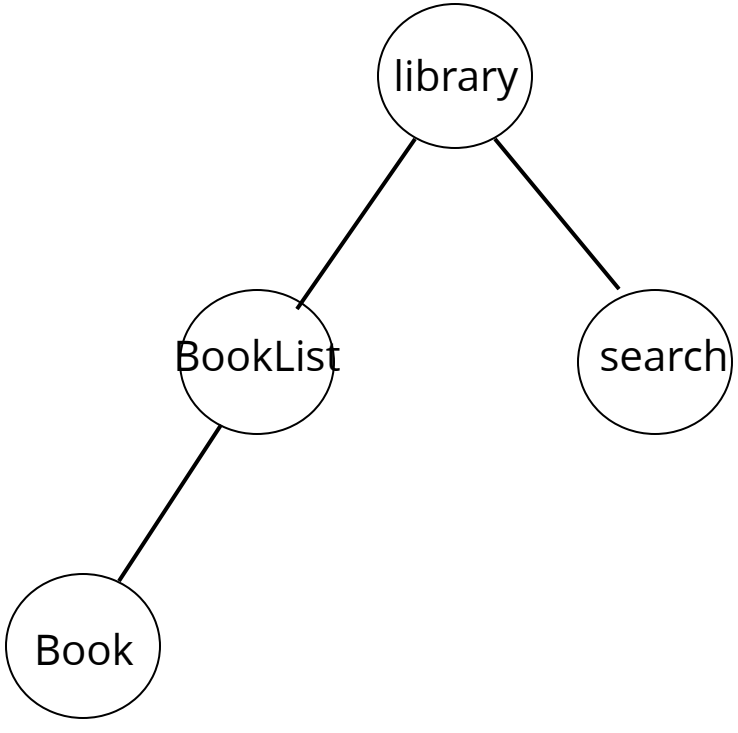
Data Down

Actions Up

Data Flow in React

Passing data between components





Parent to Child – Use Prop

Library to BooksList

```
//.....
class Library extends Component {
    //.....
    this.state={
        books
    }
    render(){
        return (
            <BookList books={this.state.books}/>
        )
    }
}
```

Parent to Child – Use Prop

BookList to Book

```
//.....
class BookList extends Component {
    //.....
    this.state={
        books: this.props.books
    }
    render(){
        return (
            {
                this.state.books.map ( (book)=>{
                    return <Book book={book}/>
                });
            }
        )
    }
}
```

Child to Parent

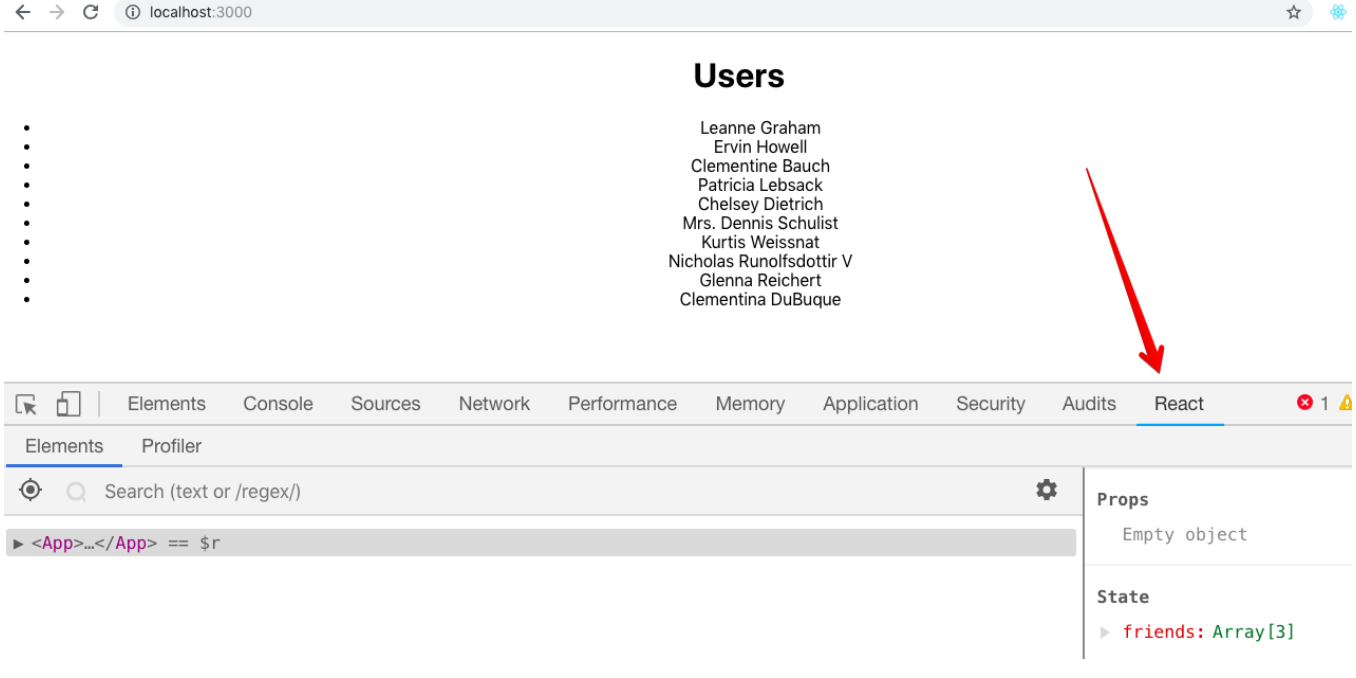
Child to Parent—Use a callback and states

1. Define a callback in parent which takes the data you need in as a parameter.
2. Pass that callback as a prop to the child
3. Call the callback using this.props.[callback] in the child, and pass in the data as the argument.

React Developer Toos

Install react developer tools via chrome extension

https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi



Virtual DOM

https://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/

React Router

What is Routing & React Router?

Routing is a way of telling your app which part of your code should handle a particular request.

To do this in React, you can use **React Router**: a library that lets you handle routing. With it you can map URLs to components or group of components. And specify how to change way your app renders when some one clicks a link on your website.

Install react router v4

```
$ npm install react-router-dom
```

React Router

At the core of every React Router application should be a router component.

```
import { BrowserRouter as Router, Route, browserHistory } from 'react-router-dom'

//.....

render(){
        return (
            <div>
                <Router history={browserHistory}>
                    <div>
                        <Route path="/" component={Home}/>
                        <Route path="/about" component={About}/>
                        <Route path="/contact" component={Contact}/>
                    </div>
                </Router>
            </div>
        )
}
```

Routers

- BrowserRouter
  - Use this, if you have a server that responds to requests
- HashRouter
  - Use this, if you are using static file server
  - It uses # (hash) in the url to render component
  - if you're building single page website with links to same page, this might be good fit

exact

When true, will only match if the path matches the location.pathname exactly.

```
import { BrowserRouter as Router, Route, browserHistory } from ‘react-rout

//.....

render(){
        return (
            <div>
                <Router history={browserHistory}>
                    <div>
                        <Route exact path=“/“ component={Home}/>
                        <Route path=“/about“ component={About}/>
                        <Route path=“/contact“ component={Contact}/>
                    </div>
                </Router>
            </div>
        )
}
```

Not found?

```
import { BrowserRouter as Router, Route, browserHistory } from ‘react-ro

//.....

render(){
        return (
            <div>
                <Router history={browserHistory}>
                    <Route path=“/“ component={Home}/>
                    <Route path=“/about“ component={About}/>
                    <Route path=“/contact“ component={Contact}/>
                    <Route path="*" component={NotFound}/>
                </Router>
            </div>
        )
}
```

Redirect

Rendering a <Redirect> will navigate to a new location

```
import { BrowserRouter as Router, Route, browserHistory } from ‘react-ro

//.....

render(){

        if(!this.state.userLoggedIn){
            return(
                <Redirect to="/login" />
            )
        }

        return (
            <div>
                <Router history={browserHistory}>
                    <Route path=“/“ component={Home}/>
                    <Route path=“/about“ component={About}/>
                    <Route path=“/contact“ component={Contact}/>
                    <Route path="*" component={NotFound}/>
                </Router>
            </div>
        )
```

Switch

Renders the first child <Route> that matches the location.

```
import { BrowserRouter as Router, Route, browserHistory } from ‘react-r

//.....

render(){

        return (
            <div>
                <Router history={browserHistory}>
                    <Switch>
                        <Route exact path=“/“ component={Home}/>
                        <Route path=“/about“ component={About}/>
                        <Route path=“/contact“ component={Contact}/>
                        <Route path="*" component={NotFound}/>
                    </Switch>
                </Router>
            </div>
        )
}
```

match

A match object contains information about how a <Route path> matched the URL. match objects contain the following properties:

- params - (object) Key/value pairs parsed from the URL corresponding to the dynamic segments of the path
- isExact - (boolean) true if the entire URL was matched (no trailing characters)
- path - (string) The path pattern used to match. Useful for building nested <Route>s
- url - (string) The matched portion of the URL. Useful for building nested <Link>s

You'll have access to match objects in various places

- Route component as this.props.match
- Route render as ({ match }) => ()
- Route children as ({ match }) => ()
- withRouter as this.props.match

withRouter

You can get access to the history object’s properties and the closest <Route>'s match via the withRouter higher-order component. withRouter will pass updated match, location, and history props to the wrapped component whenever it renders.

Link

If you've created several Routes within your application, you will also want to be able to navigate between them. React Router supplies a Link component that you will use to make this happen.

NavLink

A special version of the <Link> that will add styling attributes to the rendered element when it matches the current URL.

AXIOS

Redux