What to learn?

1. React Basics
2. React Router
3. Firebase
4. Redux
5. Stripe
6. CSS in JS (Styled components)
7. Redux-saga (handle asynchronous actions)
8. Hooks
9. Context-api
10. GraphQL
11. PWA
12. Jest for Testing

**React Key Concepts:**

1. Why react exists?
2. What problems do it solve?
3. Why it makes front-end problem easier?

**The birth of React:**

Before we have only JS, HTML, CSS. Then came Jquery which could communicate with DOM or Document Object Module. Angular Js came in to structure larger websites. But as the complexity gets bigger due to huge number of actions happening on a webpage Angular started to face problems. So Facebook developed React.

What made react so popular?

There are 4 key concepts.

1. **Don’t touch the DOM. I’ll do it.**

Many frameworks were manipulating the DOM directly. Which is the **Imperative** way. In Imperative way, one directly changes his app in response to various user events. Overhead cost due to Repaint and Reflow.

So, **React** came with the **declarative** approach. Instead of telling what to do next if this happens, we gave the blueprint or declare what could be happen if the page is in that state and react manipulates the DOM automatically. It results in less complexity, better code, faster development.

1. **Build websites like lego blogs.**

React uses Components which can be Re-used. Components are just JS functions that receives some input or attributes and return html inside JS. **JSX** file.

1. **Unidirectional Data flow**

Virtual Dom due to its dependence on State. When state changes, it changes the virtual DOM. Data can only move down. This additional restriction gives us better code which is easier to debug.

1. **UI, the rest is up to you.**

It only works as view. Anything you need just customize the app. Learn once write anywhere. It doesn’t depend on the development stack.

**Some React Keywords:**

State, Declarative, Props, JSX, Virtual DOM, Components.

**The Job of a React Developer**

1. Decide on Components.
2. Decide the State and where it lives.
3. What changes when state changes.

**React Basics**

1. Install latest version of Node.js (<https://nodejs.org/en/>)
2. Create React app (<https://reactjs.org/docs/create-a-new-react-app.html>)

>> Check the Node and Npm version by running “node -v” and “npm -v” in the terminal.

>> “Yarn” can be used too instead of npm.

**Create React App**

**To create the react app run:**

1. npm rm -g create-react-app
2. npm install -g create-react-app
3. npx create-react-app my-app
4. cd my-app
5. npm start

React uses **WEBPACK** and **BABEL**. Babel takes all JS files and make sure that it runs on every browser. Webpack builds modular files structure creating bundles and optimize it.

**Class Components**

It has more features instead of writing html inside a function. Using Class, we can access state. A state is a JS object with properties which can be accessed at any point inside our class.

To create a class:

1. import React, {Component} from ‘react’
2. Instead of this function:

**function App() {**

return(

\*\*\*\*\*\*\*\*\* HTML code here \*\*\*\*\*\*\*\*\*\*

);

**}**

We can use:

**class App extends Component {**

**render() {**

return (

\*\*\*\*\*\*\*\*\* HTML code here \*\*\*\*\*\*\*\*\*\*

);

**}**

**}**

**To access the state, we have to create constructor inside the class.**

This component class also gives us access to a property call “set state” which lets us change the properties inside a class state.

class App extends Component {

  // To access the states,

  // We have to create constructor

  constructor() {

    super();

    this.state = {

      name: 'Saif'

    }

  }

  render() {

    return (

      <div className="App">

        <header className="App-header">

          <img src={logo} className="App-logo" alt="logo" />

          {/\* Add JS tag inside html...... \*/}

          <p>

             Hi, I'm {this.state.name}!

          </p>

          <button

            onClick={() => this.setState(

              { name: 'Sarwar Saif'} )

            }>

            Change Name

          </button>

        </header>

      </div>

    );

  }

}

\*\*\* JSX uses “className” to distinguish between an html class and class in JS.

\*\*\* JS is written inside curly braces inside a JSX file.

\*\*\* Anonymous function using () =>

\*\*\* Can’t change the state without using setState.

\*\*\* In javascript “onClick” instead of “onclick” as JS uses camel case.

**Dynamic Content**

import React, {Component} from 'react';

import logo from './logo.svg';

import './App.css';

class App extends Component {

  // To access the states,

  // We have to create constructor

  constructor() {

    super();

    this.state = {

      monsters: [

        {name: 'Vampire', id: 'asc'},

        {name: 'Dracula', id: 'asr'},

        {name: 'Zombie', id: 'ast'}

      ]

    }

  }

  render() {

    return (

      <div className="App">

        {

          this.state.monsters

            .map(monster =>

              <h1 key={monster.id}> { monster.name } </h1>

              /\* Key is used to let the react know that

                it doesn't need to re-render everything.

                It can just change the associated element \*/

            )

        }

      </div>

    );

  }

}

**Single Page Application**

Single page web applications are built around the concept of dynamically rewriting the contents of that single page. This is different from loading pre-rendered pages from the server.

And this is where the magic happens. By taking this approach, single page web applications avoid the interruption caused by rendering the pages on the server. This removes the biggest problem the web development world usually faces with regards to providing a seamless user experience.

**Life Cycle Method Basic**

You can think of React lifecycle methods as the series of events that happen from the birth of a React component to its death.

Every component in React goes through a lifecycle of events. I like to think of them as going through a cycle of birth, growth, and death.

1. Mounting – Birth of your component
2. Update – Growth of your component
3. Unmount – Death of your component

**Fetching Content**

To fetch content, we need to use a life cycle method name “componentDidMount” which ensures running certain tasks when the page get mounted.

APP.JS

import React from 'react';

import {CardList} from './components/card-list/card-list.component'

import './App.css';

class App extends React.Component {

  // To access the states,

  // We have to create constructor

  constructor() {

    super();

    this.state = {

      monsters: []

    }

  }

  componentDidMount() {

    fetch('https://jsonplaceholder.typicode.com/users')

    /\* Api requests in the URL which returns a promise \*/

      .then(response => response.json())

      .then(users => this.setState( {monsters: users} ))

  }

  render() {

    return (

      <div className="App">

        <CardList name='Saif'> <!—this is the component -🡪

          {

            this.state.monsters

              .map(monster =>

                <h1 key={monster.id}> { monster.name } </h1>

                /\* Key is used to let the react know that

                  it doesn't need to re-render everything.

                  It can just change the associated element \*/

              )

          }

        </CardList>

      </div>

    );

  }

}

export default App;

\*\*\* Components take props. Use props and props.children.

**card-list.component.jsx**

import React from 'react'

import './card-list.styles.css'

export const CardList = (props) => {

    return(<div className='card-list'>{props.children}</div>);

}

*Adjacent JSX elements must be wrapped in an enclosing tag…*

To use different components on the same page and styling differently we need to use Fragments. React@16.2 introduces the Fragment component. Instead of creating div inception and the inability to use Semantic, Bootstrap and Material UI on different components you are rendering you now have the ability to use Fragment avoid having to wrap multiple elements in <div>’s or <spans> so you can style each one individually and not as a whole.

After adding **Fragment** on the previous code, I can use different divs.

import React, {Fragment} from 'react'

import './card-list.styles.css'

export const CardList = (props) => {

    return(

        <Fragment>

            <div className='card-list'>{props.name}</div>

            <div className='card-list'> {props.children} </div>

        </Fragment>

    );

}

**“setState” is Asynchronous**

“setState” is an asynchronous function. So when we console.log the the state it doesn’t change immediately.

<input type='search'

          placeholder='search monsters'

          onChange={e => {

            this.setState({searchField: e.target.value})

            console.log(this.state.searchField)

          }}

        />

Asynchronous vs Synchronous is a big thing in javascript development, specially in React. Synchronous action is something that we can expect to happen immediately. Javascript knows how much time it will take whereas, Asynchronous actions take indefinite amount of time. What it does after running the asynchronous action it finishes running the rest of the code and then when the asynchronous event finishes it then run that event.

So, we need to use callback to get the effect of asynchronous actions. It runs the rest of the code when setState finishes updating the state.

We can use the callback like following:

<input type='search'

          placeholder='search monsters'

          onChange={e => {

            this.setState({searchField: e.target.value},

              ()=> {

                console.log(this.state.searchField)

              })

            }}

        />

**# Object Destructuring**

Basic assignment

const o = {p: 42, q: true};

const {p, q} = o;

let a, b;

Assignment without declaration

({a, b} = {a: 1, b: 2});

Assigning to new variable names

const o = {p: 42, q: true};

const {p: foo, q: bar} = o;

const { monsters, searchField} = this.state // Obeject Destructuring

    const filteredMonsters = monsters.filter(monster =>

      monster.name.toLowerCase().includes(searchField.toLowerCase())

    )

    // Convert it into lowercase and then search if the string includes anything like searchField

    return (

      <div className="App">

        <input type='search'

          placeholder='search monsters'

          onChange={e => /\* Synthetic React Event "onChnage", this are not html events.\*/

            this.setState({searchField: e.target.value})

            }

        />

        <CardList name='Saif' monsters = {filteredMonsters}>

        </CardList>

      </div>

    );

  }

}

Search Box Component

import React from 'react'

import './search-box.styles.css'

export const SearchBox = ({placeholder, handleChange}) => {

    return(

        <input className='search' type='search'

          placeholder= {placeholder}

          onChange={handleChange}

        />

    );

}

<SearchBox

          placeholder='search monsters'

          handleChange={e => {this.setState({searchField: e.target.value})}}

        ></SearchBox>

\*\*\* To use SAAS file in React install: **npm install node-sass**

\*\*\* Functional Components like search box don’t have access to the state or life cycle methods.

\*\*\* Structuring state is one of the most important things in React.

**Class Methods and Arrow functions:**

1. **“this” is a special keyword in Javascript that references the context in which it’s being invoked.**
2. **By default constructor, life cycle methods and render function gets the Context of the App.js or root class but if we call our own function it won’t get the context. Thus will be undefined. So, we have to bind the context with our function when the code get compiled. We do this using Arrow function. [Lexical Scoping]**
3. handleChange = (e) => {
4. this.setState({searchField: e.target.value})
5. }

        <SearchBox

          placeholder='search monsters'

          handleChange={this.handleChange}

        ></SearchBox>

aa