Day 01 - JS Basics

Monday, January 18, 2021   12:25 PM

JS Basics

Arrow functions

- var sumA = (a,b) =>{return a+b;}
  console.log("sum A "+sumA(2,5));
- var sumB = (a,b) =>(a+b);
  console.log("sum B "+sumB(2,9));
- function addition (a) {return function(b){return a+b}};
  //addition(a)(b);
  console.log("addition:"+addition(4)(67));
- var additionA = a => b => (a+b);
  console.log("additionA:"+additionA(4)(8));
- // see => used three times
  var additionB = a => b => c => (a+b);

- Arrow functions cannot be used as classes

  -
    ```
    var Person = () => {
            this.name = 'asd';
            this.getName = function() {
                    return this.name;
            }
    }
    ```

Iterators

```
function makeRangeIterator(start = 0, end = Infinity, step = 1) {
   let nextIndex = start;
   let iterationCount = 0;
const rangeIterator = {
      next: function() {
         let result;
         if (nextIndex < end) {
            result = { value: nextIndex, done: false }
            nextIndex += step;
            iterationCount++;
            return result;
         }
         return { value: iterationCount, done: true }
      }
   };
   return rangeIterator;
}
const it = makeRangeIterator(1, 10, 2);
it.next();
```

```
it.next();
it.next();
console.log(it.next());
it.next();
```

Generators

```
function* range (start, end, step) {
   while (start < end) {
      yield start
      start += step
   }
}
const it = range(0, 10, 2);
it.next();
console.log(it.next());
it.next();
it.next();
it.next();
it.next();
console.log(it.next());
/*
for (let i of range(0, 10, 2)) {
   console.log(i) // 0, 2, 4, 6, 8
}
*/
```

Day 01 - React

Monday, January 18, 2021 2:39 PM

Creating react app

- npx create-react-app react-app
- For using ts in react app
    - npx create-react-app react-app --template typescript
- yarn create react-app my-app
- cd react-app
- npm start

- The React module name and the file name should be same and the first letter should be in capitals.

- Function components are better over Class Components in terms of performance.

- Life cycle of React Class Components
    - constructor
    - componentDidMount - when component mounted successfully
    - componentWillMount - when component
    - componentDidUnmount- when component unmounted successfully
    - componentWillUnmount
    - componentWillReceiveProps

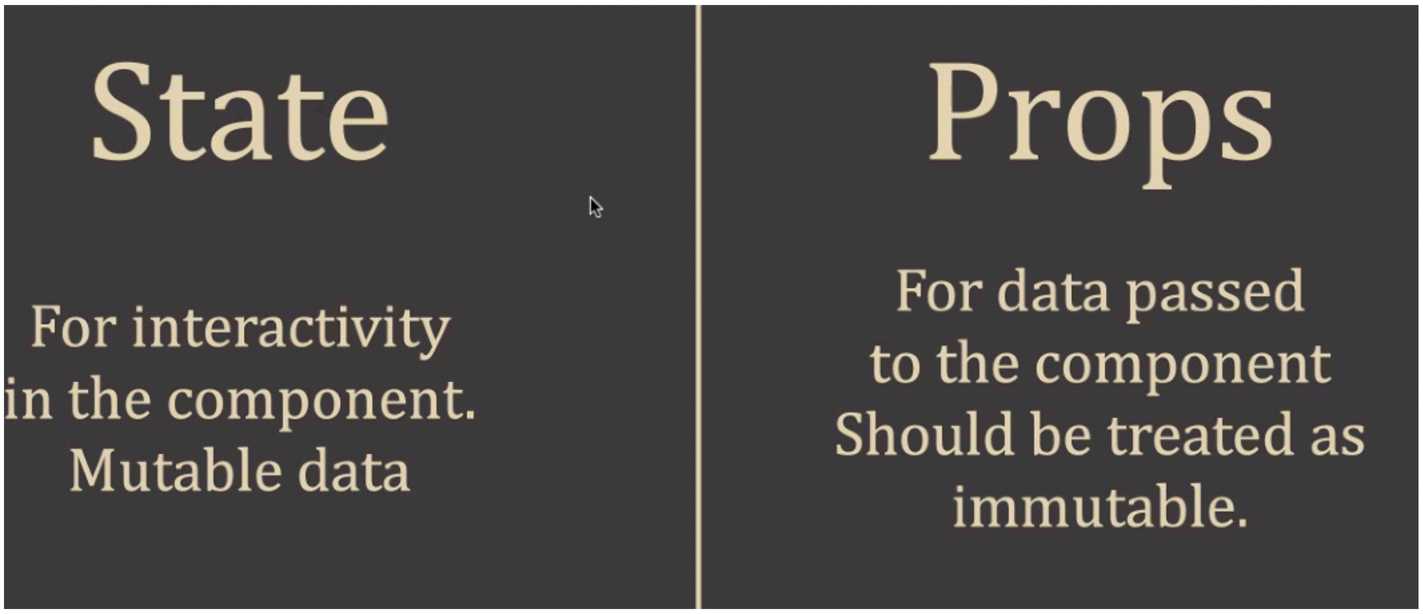
Day 02 - React

Tuesday, January 19, 2021 9:33 AM

React class component Life cycle hooks

- Do not setState in component render method as setState re-renders the component which makes it recurring.

- componentWillMount(UNSAFE_componentWillMount) -> Unsafe and very rarely used
- componentWillReceiveProps(UNSAFE_componentWillReceiveProps) -> Unsafe and very rarely used
- componentWillUpdate(UNSAFE_componentWillUpdate) -> Unsafe and very rarely used



State

For interactivity in the component. Mutable data

Props

For data passed to the component Should be treated as immutable.

- Server Side Rendering
    - Use renderToString instead of render for SEO

Day 02 - Typescript

Tuesday, January 19, 2021 12:35 PM

- npm i -g typescript
- tsc <file-name.ts> // typescript compiler

- npm i -g ts-node
- ts-node <file-name.ts>

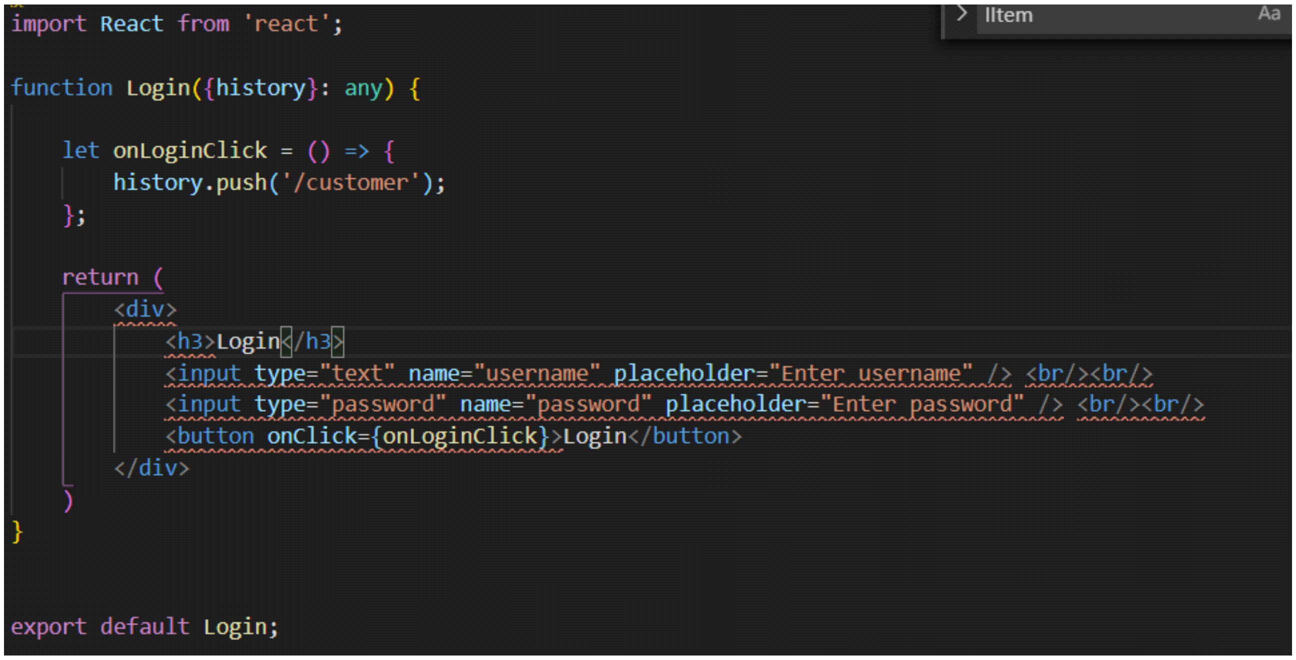
Day 03 - React

Wednesday, January 20, 2021 9:51 AM

React Router

npm i react-router --save
npm i react-router-dom
npm i @types/react-router-dom

import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom';

- 

```
import React from 'react';

function Login({history}: any) {

    let onLoginClick = () => {
        history.push('/customer');
    };

    return (
        <div>
            <h3>Login</h3>
            <input type="text" name="username" placeholder="Enter username" /> <br/><br/>
            <input type="password" name="password" placeholder="Enter password" /> <br/><br/>
            <button onClick={onLoginClick}>Login</button>
        </div>
    )
}


export default Login;
```

- Router implemented in App.tsx by default gives us access to history api in the components under router

Day 03 - Unit Testing - Jasmine

Wednesday, January 20, 2021 2:27 PM

Understanding Unit test and Jasmine
----
1. install jasmine

```
$> npm install -g jasmine
$> mkdir hello-jasmine
$> cd hello-jasmine
```

create file
appLogic.js >

```
---
var app = {};
app.name = function() {
    return "Hello, testing!";
  }

app.sum = function(a,b) {
    return a+b;
  }

app.mul = function(a,b) {
    return a*b;
  }
module.exports = app;
---
```

```
$> jasmine init
$> cd spec
```

create file
appLogic.spec.js >

```
---
var app = require("../appLogic");
describe("AppLogic test", function() {
  it("test of AppLogic name", function() {
    expect(app.name()).toEqual('Hello, testing!');
  });

  it("test of sum", function() {
    expect(app.sum(2,3)).toEqual(6);
  });

  it("test of mul", function() {
    expect(app.mul(2,3)).toEqual(6);
  });
});
```

```
$> cd ..
```

run jasmine to test

```
$> jasmine ./spec/appLogic.spec.js
```

Output

```
----
Started
...


3 specs, 0 failures
Finished in 0.009 seconds
----
```

Day 03 - Unit Testing - Jest

Wednesday, January 20, 2021 2:32 PM

Jest, Mocha, Jasmine are just test runners

Node -> Jasmine runner
Angular -> e2e => ng-test Karma
react Jest | enzymes
express | RESTAPI | mocha | chai

- Global -> npm install -g jest
- Local to project -> npm install jest --save

- Example
  - sum.js

```
function sum(a, b) {
  return a + b;
}
module.exports = sum;
```

  - sum.test.js

```
const sum = require('./sum');
test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

  - Configure package.json

```
"scripts": {
  "test": "jest"
}
```

  - npm run start

- Snapshot testing
  - Creates a snapshot of the html and runs the jest tests
  - libraries required for jest snapshots

- Enzyme library testing
  - Jest, Mocha, Jasmine are just test runners
  - We can use enzyme like libraries to do Shallow rendering/Full rendering/Static rendering of the components

Debugging with Jest

1. In file package.json

add
```
  "test:debug": "react-scripts --inspect-brk test --runInBand --no-cache "
```
2. App.test.tsx
```
-------
test(' sample learn react link', () => {
  debugger;
  render(<App />); //This is full rendering
  const linkElement = screen.getByText(/Customer App/i);
  expect(linkElement).toBeInTheDocument();
});
-------
```
3. terminal
```
$> yarn test:debug
```
4. chrome://inspect/#devices

  open dedicated devtool

Async Jest Debugging example
```
import React from 'react';
import Login from './';
import $ from 'jquery';
import { shallow } from 'enzyme';
describe('Login', () => {
  afterEach(() => {
    jest.restoreAllMocks();
  });
  test('should get data', async () => {
    const ajaxSpy = jest.spyOn($, 'ajax');
    const wrapper = shallow(<Login></Login>);
    const instance = wrapper.instance();
    (instance as any).getData(); //doLogin
    expect(wrapper.text()).toBe('Login');
    expect(ajaxSpy).toBeCalledWith({
      type: 'GET',
      url: 'https://github.com/mrdulin',
      // tslint:disable-next-line: no-string-literal
      success: instance['handleSuccess'],
      // tslint:disable-next-line: no-string-literal
      error: instance['handleError']
    });
  });
test('handleSuccess', () => {
    const logSpy = jest.spyOn(console, 'log');
    const wrapper = shallow(<Login></Login>);
    const instance = wrapper.instance();
    // tslint:disable-next-line: no-string-literal
    instance['handleSuccess']('some data');
    expect(logSpy).toBeCalledWith('some data');
  });
test('handleError', () => {
    window.alert = jest.fn();
    const wrapper = shallow(<Login></Login>);
    const instance = wrapper.instance();
    // tslint:disable-next-line: no-string-literal
    instance['handleError']();
    expect(window.alert).toBeCalledWith('ERROR');
  });
});
```

Day 03 - Unit Testing - Mocha

Wednesday, January 20, 2021 3:32 PM

Start with Mocha and Chai (for Async Test)
1. Create a directory for the application:
```
$> mkdir mocha-app && cd mocha-app
```
2.
```
$> npm init
name: hello-world
entry point: app.js
test command: ./node_modules/.bin/mocha
```
We shall use this framework to test the application
3. add express
```
$> npm install express --save
$> npm install request mocha chai--save
```
4. create app.js
```
-----
//Load express module with `require` directive
var express = require('express')
var app = express()
//Define request response in root URL (/)
app.get('/', function (req, res) {
  res.send('Hello World')
})
//Launch listening server on port 8080
app.listen(8080, function () {
  console.log('App listening on port 8080!')
})
------
```
5. Run the app

The application is ready to launch:
```
$ nodemon app.js
```
----
6. Time to define our first test. We shall keep all testing files in a separate /test directory (orndung muss sein):
```
$> mkdir test
```
7. Now, add the first testing file:
```
$> touch test/test-pages.js
```
8. The test verfies the content of the websit. For that, we need an HTTP client: https://npm.io/package/request
```
$> npm install request --save-dev
```
The file should look like this now:
9. update test/test-pages.js
```
---------
var expect  = require('chai').expect;
var request = require('request');
it('Main page content', function(done) {
    request('http://localhost:8080' , function(error, response, body) {
        expect(body).to.equal('Hello World');
        done();
    });
});
----------
```

10. Run the file to trigger the tests:

```
$> npm test
```

11. Let's add some more tests that will check the status of the homepage and /about page:
 update test/test-pages.js

```
-----
var expect  = require('chai').expect;
var request = require('request');
it('Main page content', function(done) {
    request('http://localhost:8080' , function(error, response, body) {
        expect(body).to.equal('Hello World');
        done();
    });
});
it('Main page status', function(done) {
    request('http://localhost:8080' , function(error, response, body) {
        expect(response.statusCode).to.equal(200);
        done();
    });
});
it('About page content', function(done) {
    request('http://localhost:8080/about' , function(error, response, body) {
        expect(response.statusCode).to.equal(404);
        done();
    });
});
-------
```

Run npm test again and see the results. The /about page is not ready yet so it will return a 404:
Expanded tests results

Day 03 - React

Wednesday, January 20, 2021 5:16 PM

- adding bootstrap to react
    - npm i react-bootstrap@1.0.1 @types/react-bootstrap --save
    - To use bootstrap in react
        - import Button from 'react-bootstrap/Button
        - import Form from 'react-bootstrap/Form'
        - import Navbar from 'react-bootstrap/Navbar'
    - https://react-bootstrap.netlify.app/components/alerts
    - Example:

```
import React, {useState, useEffect} from 'react';
import ICustomer from '../interfaces/ICustomer';

import customerService from '../services/customerService';

import Button from "react-bootstrap/Button";
import Form from "react-bootstrap/Form";

function AddCustomer({history, match}: any) {
    let [customer, setCustomer] = useState<ICustomer | undefined>({
        id: '',
        name: '',
        mail: '',
        phone: '',
        address: ''
    });
    let nameInputRef: any = React.createRef<HTMLInputElement>();
    let emailInputRef: any = React.createRef<HTMLInputElement>();
    let phoneInputRef: any = React.createRef<HTMLInputElement>();
    let addressInputRef: any = React.createRef<HTMLInputElement>();
    let [buttonLabel, setButtonLabel] = useState<string | undefined>("Add Customer");
    useEffect(()=>{
        console.log("CustomerId :"+match.params.id);
        if (!match.params.id) return;
        let customerObj: ICustomer = customerService.getCustomerById(match.params.id);
        setCustomer(customerObj);
        nameInputRef.current.value = customerObj.name;
        emailInputRef.current.value = customerObj.mail;
        phoneInputRef.current.value = customerObj.phone;
        addressInputRef.current.value = customerObj.address;
        setButtonLabel("Update Customer");
    },[]);
    let addUpdateCustomer = () => {
        if (nameInputRef?.current?.value === "") return;
        const id: (string | undefined) = customer?.id;
        const customerObj = {
          name: nameInputRef?.current?.value,
          mail: emailInputRef?.current?.value,
          phone: phoneInputRef?.current?.value,
          address: addressInputRef?.current?.value,
          id: id !== '' ? customer?.id : Date.now().toString()
        };
        id === '' ? customerService.addCustomer(customerObj) : customerService.updateCustomer(customerObj);
        setCustomer({
            id: '',
            name: '',
            mail: '',
            phone: '',
            address: ''
        });
        setButtonLabel('Add Customer');
        history.push('/customer');
    };
    return(
        <Form>
            <h3> Add Customer </h3> <br/>
            <Form.Group controlId="formName">
                <Form.Label>Name</Form.Label>
                <Form.Control ref={nameInputRef} type="text" placeholder="Enter name" />
            </Form.Group>
            <Form.Group controlId="formName">
                <Form.Label>Email</Form.Label>
                <Form.Control ref={emailInputRef} type="email" placeholder="Enter email-id" />
                <Form.Text className="text-muted">
                    We'll never share your email with anyone else.
                </Form.Text>
            </Form.Group>
            <Form.Group controlId="formName">
                <Form.Label>Phone</Form.Label>
                <Form.Control ref={phoneInputRef} type="text" placeholder="Enter phone" />
            </Form.Group>
            <Form.Group controlId="formName">
                <Form.Label>Address</Form.Label>
                <Form.Control ref={addressInputRef} type="text" placeholder="Enter address" />
            </Form.Group>
            <Button onClick={addUpdateCustomer} variant="primary">
                {buttonLabel}
            </Button>
        </Form>
    )
}
export default AddCustomer;
```

Day 04 - Hooks

Thursday, January 21, 2021 9:37 AM

Hooks in react

useEffect

- When using useEffect we pass an empty [] to make sure it is called only on mounting the component
- If empty array has not been passed on every state change the component gets re-rendered again

```
const [count, setCount] = useState(0);
 // addUpdateCustomer handle based on customer.id
   /*
     useEffect(
       ()=>{

      },[]);
            mount / unmount / props / events
          */
          useEffect( () => {
            console.log(" >> useEffect");
           } return(()=>{
                      // used for cleaning up
             })
             },[])
```

useContext

- we can create context using React.createContext()
- and we have to create a provider through which we can use the context

```
const themes = {
   light: {
     foreground: "#000000",
     background: "#eeeeee",
     name:"Light theme"
   },
   dark: {
     foreground: "#ffffff",
     background: "#222222",
     name:"Dark theme"
   }
 };

 const ThemeContext:any = React.createContext(themes.light);// create with default Value

 function ContextApp() {
   return (
    <ThemeContext.Provider value={themes.light}>
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar() {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

function ThemedButton() {
  const theme:any = useContext(ThemeContext);
  return (
    <div>
      <h4>The theme is {theme.name}</h4>
      <button style={{ background: theme.background, color: theme.foreground }}>
        I am styled by theme context!
      </button>
    </div>
);
}
```

Day 04 - Protected Routes

Thursday, January 21, 2021 2:27 PM

Implement Protected Routes in customer-app
----
1. add components/UserContext.js

```
import { createContext } from "react";
export default createContext();
```
---
2. add components/Protected.jsx (component)

```
import React from "react";
import { Route, Redirect } from "react-router-dom";
const Protected = ({ isLoggedIn, children }) => (
  <Route
    render={() =>
      isLoggedIn ? (
        children
      ) : (
        <Redirect
          to={{
            pathname: "/login"
          }}
        />
      )
    }
  />
);
export default Protected;
```
----
3. Change in App.js

```
.
.
import UserCtx from './components/UserContext'
.
.
.
function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  //const theme = useContext(ThemeContext);
  return (
    <Router>
      <UserCtx.Provider
        value={{
          isLoggedIn,
          doLogin: code =>
            code ? setIsLoggedIn(true) : setIsLoggedIn(false)
        }}
      >
    <div>
      <div className="App">
        <h2>Customer Management</h2>
       </div>
       <Switch>
          <Route exact path='/' component={Login} />
          <Route exact path='/home' component={Home} />
          <Route exact path='/reducer' component={ReducerExample} />
          <Route exact path='/home' component={Home} />
          <Route exact path='/timer' component={() => <TimerF name="One" startCount={11} />}/>
          <Route exact path='/todo' component={ToDoApp} />
          <Protected isLoggedIn={isLoggedIn} path="/customers">
              <Customers />
          </Protected>
          <Route exact path='/customer/add' component={AddEditCustomer} />
          <Route exact path='/customer/edit/:id' component={AddEditCustomer} />
          <Route exact path='/login' component={Login} />
       </Switch>
    </div>
    </UserCtx.Provider>
 </Router>
  );
}
export default App;
```
4. Change in Login.js

```
------
.
.
import UserContext from "./UserContext";
.
.
// function Login(props) {
    const userContext = useContext(UserContext);
.
.
.
            //.then(response => response.json())
    //      .then(response => {
             //console.log(JSON.stringify(response));
                    //if(response.result == "success"){
  userContext.doLogin(true);
             //  props.history.push('/home');
            //          }else{
            //                  alert(response.msg);
            //          }
```

Day 04 - HOC, Composition

Thursday, January 21, 2021 4:14 PM

HOC

- Passing component as param to another component
- Used when common code to be shared across multiple components

Day 04 - Custom Events

Thursday, January 21, 2021 5:05 PM

Actions a event (listened by multiple component)

```
npm install react-custom-events --save
```

https://www.npmjs.com/package/react-custom-events
===============
Custom Event in React
------
1. Implement Listener in App.tsx

```
--
.
.
.
import { useCustomEventListener } from 'react-custom-events';
.
.
.
//inside component
function App() {
.
.
.
   useCustomEventListener('my-event', data => {
     console.log("my-event received in App.tsx "+JSON.stringify(data));
   });
.
.
.
```

2. Implement Listener in AddCustomer.tsx

```
--
..
.
import { emitCustomEvent } from 'react-custom-events';
.
.  // inside addUpdateCustomer
    var addUpdateCustomer = async () => {
        emitCustomEvent('my-event', "TEST DATA");
```

===================
App has lot of actions *that create
addCustomer A => C,
------
all events must be only listened at one destination Reducer ==> state (globalState)
Redux = customEvent (Action) + reducer + state

From <http://training.pyther.com/deloitte-reactjs/04-day/17-CustomEvents.txt>

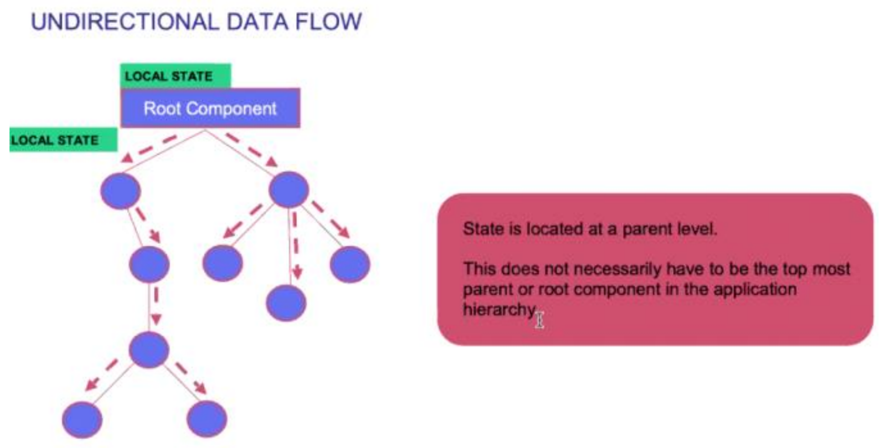
Day 04 - Error Boundary

Thursday, January 21, 2021 5:07 PM

```
import React, { Component } from 'react';
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { error: null, errorInfo: null };
  }

  componentDidCatch(error, errorInfo) {
    // Catch errors in any components below and re-render with error message
    this.setState({
      error: error,
      errorInfo: errorInfo
    })
    // You can also log error messages to an error reporting service here
  }

  render() {
    if (this.state.errorInfo) {
      // Error path
      return (
        <div>
          <h2>Something went wrong.</h2>
          <details style={{ whiteSpace: 'pre-wrap' }}>
            {this.state.error && this.state.error.toString()}
            <br />
            {this.state.errorInfo.componentStack}
          </details>
        </div>
      );
    }
    // Normally, just render children
    return this.props.children;
  }
 }

 class BuggyCounter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { counter: 0 };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(({counter}) => ({
      counter: counter + 1
    }));
  }

  render() {
    if (this.state.counter === 5) {
      // Simulate a JS error
      throw new Error('I crashed!');
    }
    return <h1 onClick={this.handleClick}>{this.state.counter}</h1>;
  }
}

function ExApp() {
  return (
    <div>
      <p>
        <b>
          This is an example of error boundaries in React 16.
          <br /><br />
          Click on the numbers to increase the counters.
          <br />
          The counter is programmed to throw when it reaches 5. This simulates a JavaScript error in a component.
        </b>
      </p>
      <hr />
      <ErrorBoundary>
        <p>These two counters are inside the same error boundary. If one crashes, the error boundary will replace both of them.</p>
        <BuggyCounter />
        <BuggyCounter />
      </ErrorBoundary>
      <hr />
      <p>These two counters are each inside of their own error boundary. So if one crashes, the other is not affected.</p>
      <ErrorBoundary><BuggyCounter /></ErrorBoundary>
      <ErrorBoundary><BuggyCounter /></ErrorBoundary>
    </div>
  );
}
export default ExApp;
```

Day 05 - Redux

Friday, January 22, 2021 9:42 AM

- React has unidirectional data flow (top-down)
- React View has only a single source of data truth through state
- 

State is located at a parent level.

This does not necessarily have to be the top most parent or root component in the application hierarchy

- Disadvantages of Redux
    - Component are reusable but the container reusablility will be effected as they use connect() to connect with a specified reducer

Trainer

Friday, January 22, 2021 5:36 PM

Vivek Singhwal
https://www.linkedin.com/in/viveksinghwal/