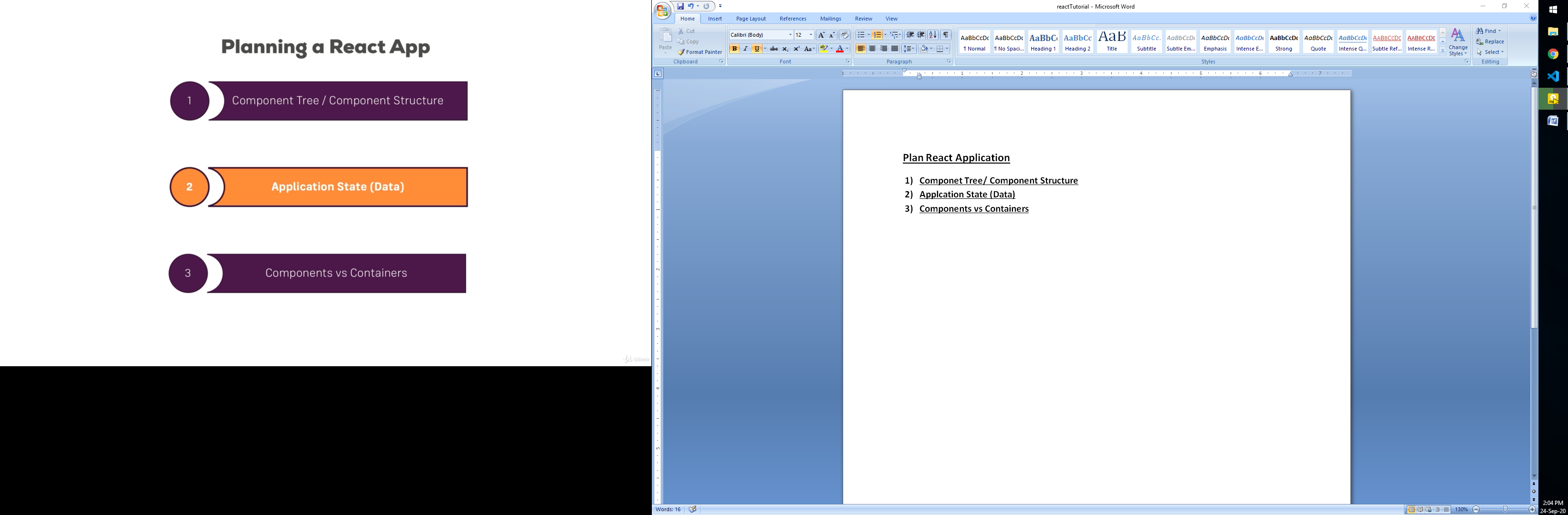
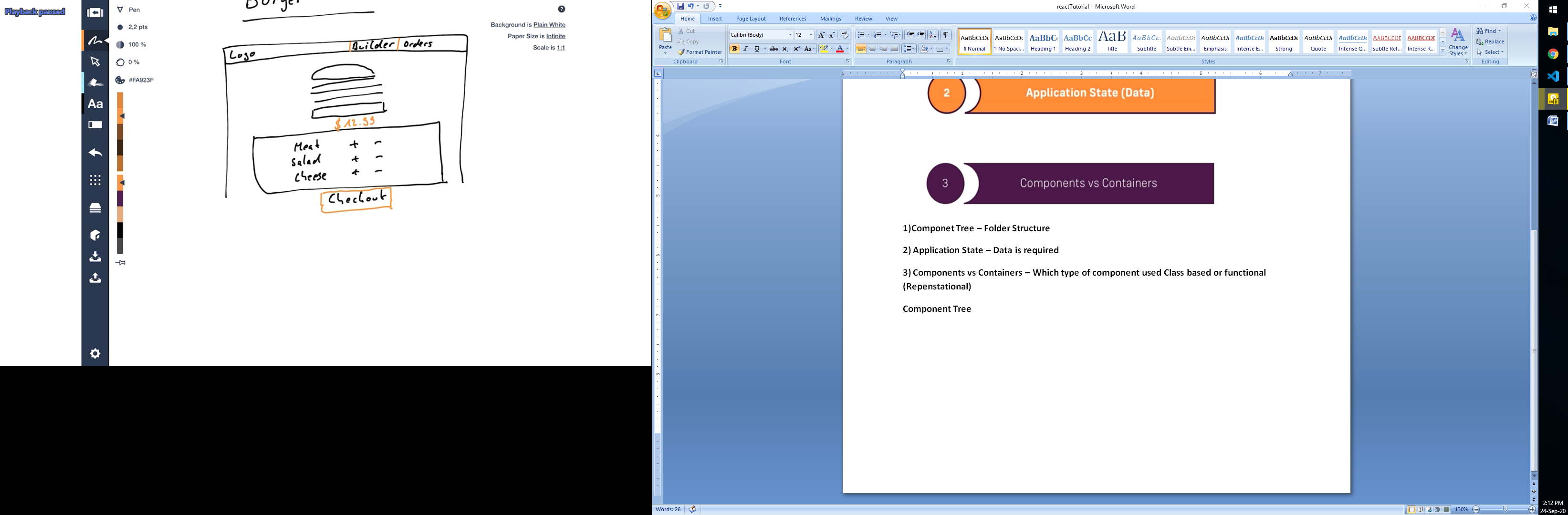
****

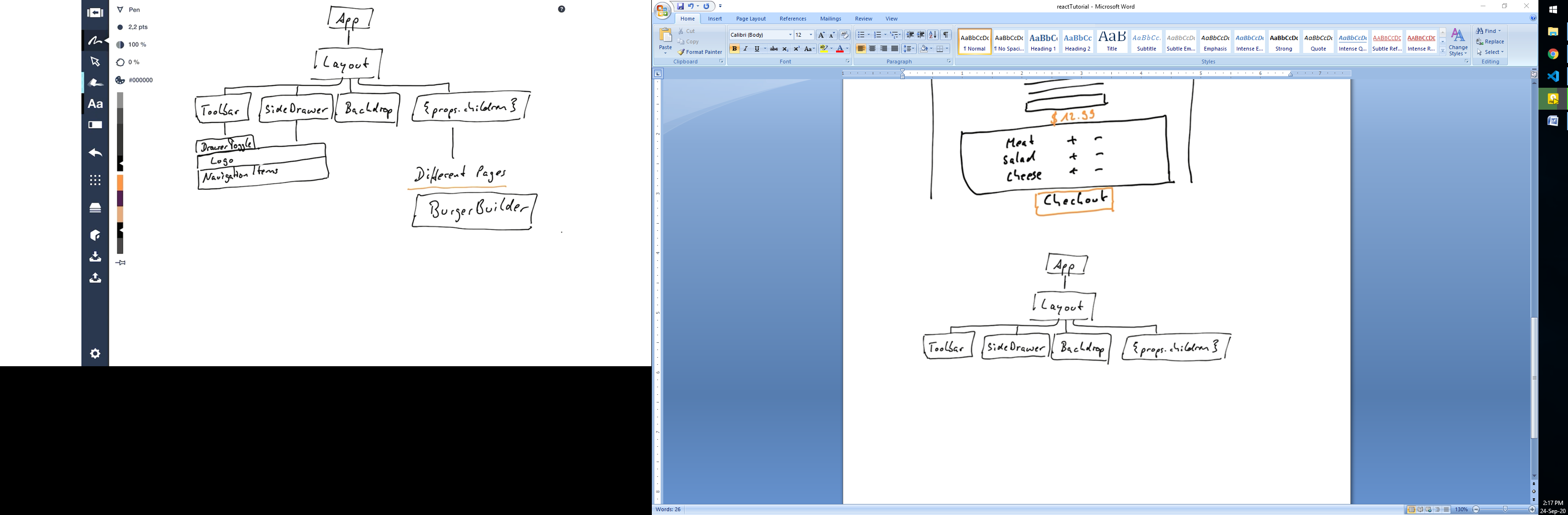
**1)Componet Tree – Folder Structure**

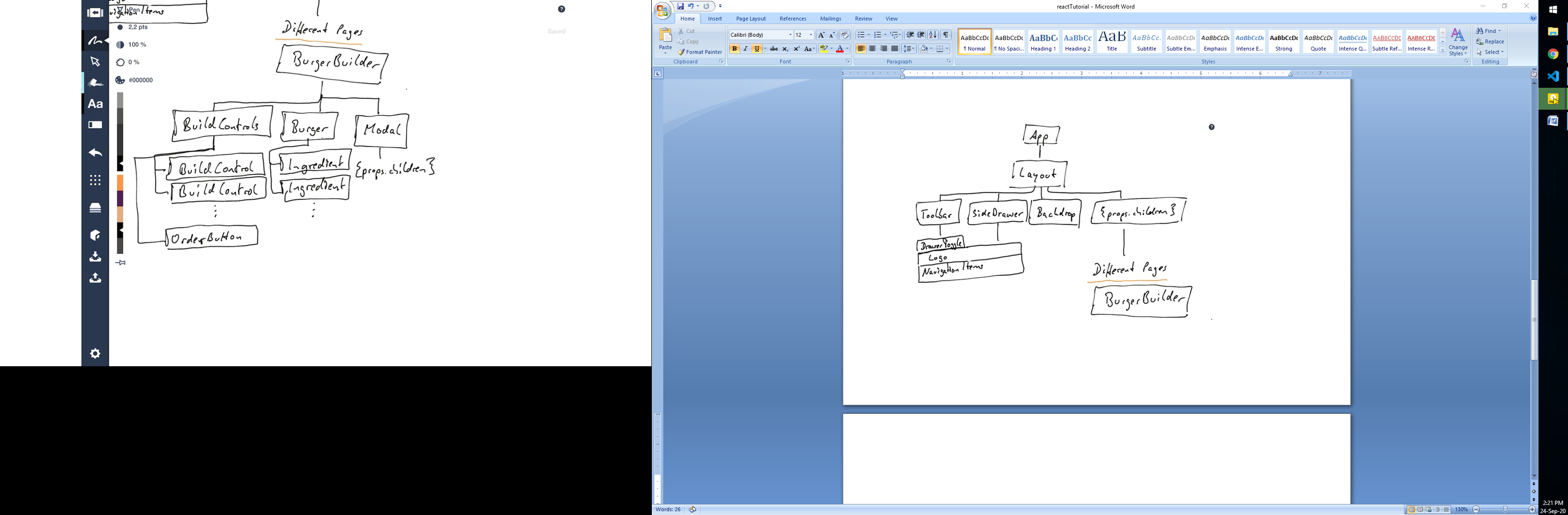
**2) Application State – Data is required**

**3) Components vs Containers – Which type of component used Class based or functional (Repenstational)**

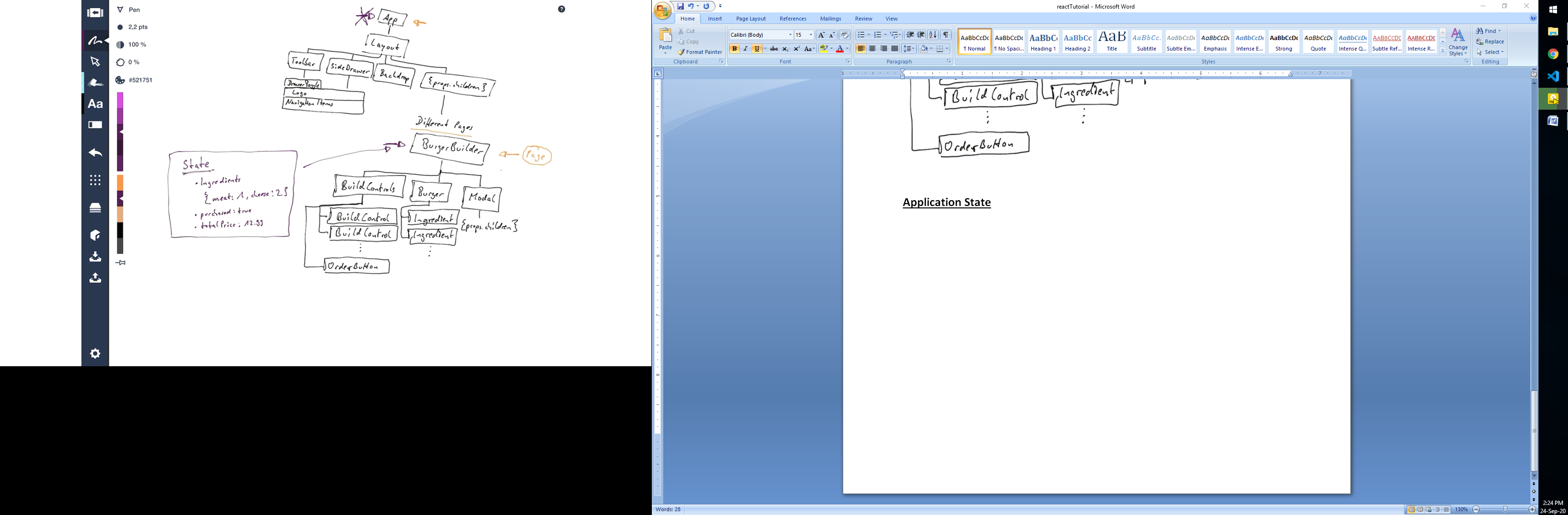
**Component Tree**

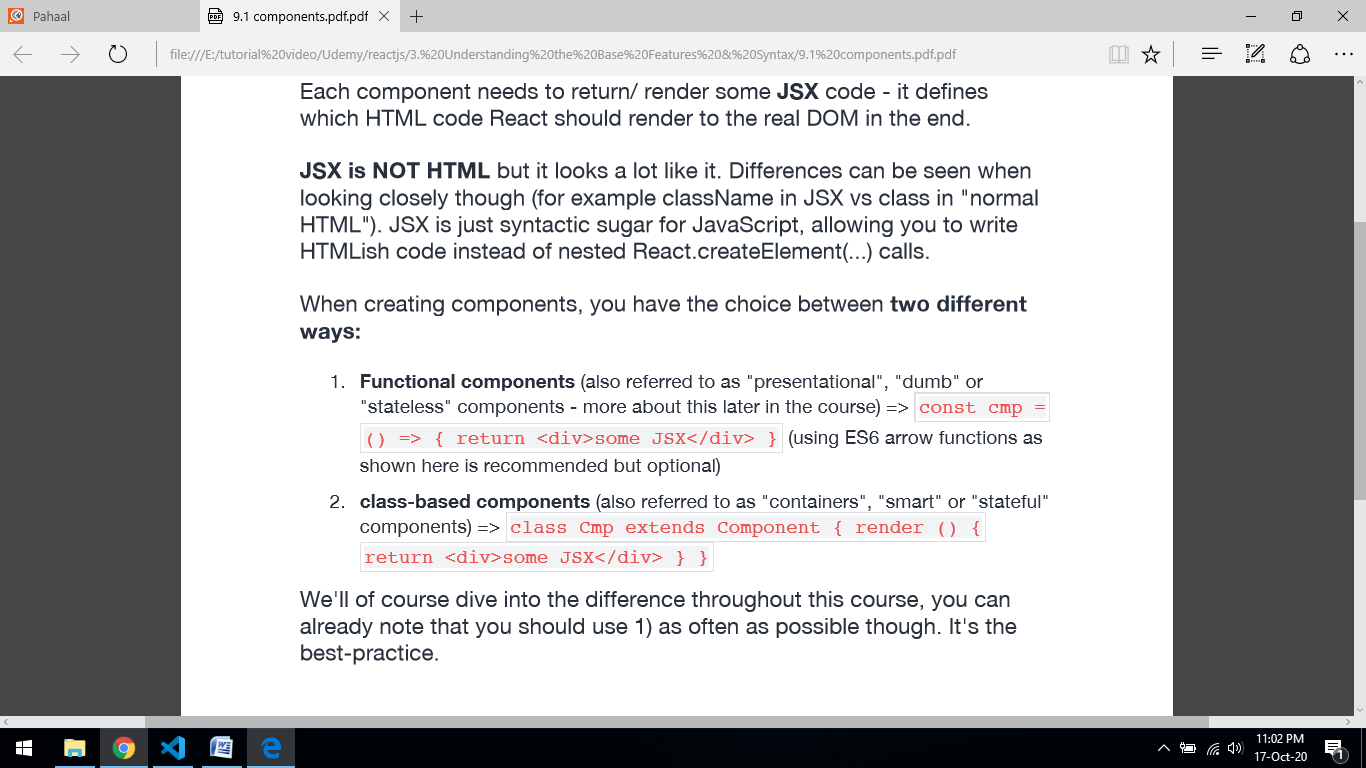
****

****

****

**Application State**

****

****

**Way of calling functions**

1. Using bind method

<button style={style} onClick = {this.switchNameHandler.bind(this,'Maximilian!!')}>Switch Name</button>

1. Using arrow function

 <button style={style} onClick={() => this.switchNameHandler('Maximilian!!')}>Switch Name</button>

**Define inline style**

render () {

    const style = {

      backgroundColor: 'white',

      font: 'inherit',

      border: '1px solid blue',

      padding: '8px',

      cursor: 'pointer'

    };

    return (

      <div className="App">

        <button

          style={style}

          onClick={() => this.switchNameHandler('Maximilian!!')}>Switch Name</button>

</div>

)

}

**Complete example**

App.js

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

class App extends Component {

  state = {

    persons: [

      { name: 'Max', age: 28 },

      { name: 'Manu', age: 29 },

      { name: 'Stephanie', age: 26 }

    ],

    otherState: 'some other value'

  }

  switchNameHandler = (newName) => {

    // console.log('Was clicked!');

    // DON'T DO THIS: this.state.persons[0].name = 'Maximilian';

    this.setState( {

      persons: [

        { name: newName, age: 28 },

        { name: 'Manu', age: 29 },

        { name: 'Stephanie', age: 27 }

      ]

    } )

  }

  nameChangedHandler = (event) => {

    this.setState( {

      persons: [

        { name: 'Max', age: 28 },

        { name: event.target.value, age: 29 },

        { name: 'Stephanie', age: 26 }

      ]

    } )

  }

  render () {

    const style = {

      backgroundColor: 'white',

      font: 'inherit',

      border: '1px solid blue',

      padding: '8px',

      cursor: 'pointer'

    };

    return (

      <div className="App">

        <button

          style={style}

          onClick={() => this.switchNameHandler('Maximilian!!')}>Switch Name</button>

  <Person

          name={this.state.persons[1].name}

          age={this.state.persons[1].age}

          click={this.switchNameHandler.bind(this, 'Max!')}

          changed={this.nameChangedHandler} >My Hobbies: Racing</Person>

      </div>

    );

  }

}

export default App;

Person.js

import React from 'react';

import './Person.css';

const person = ( props ) => {

    return (

        <div className="Person">

            <p onClick={props.click}>I'm {props.name} and I am {props.age} years old!</p>

            <p>{props.children}</p>

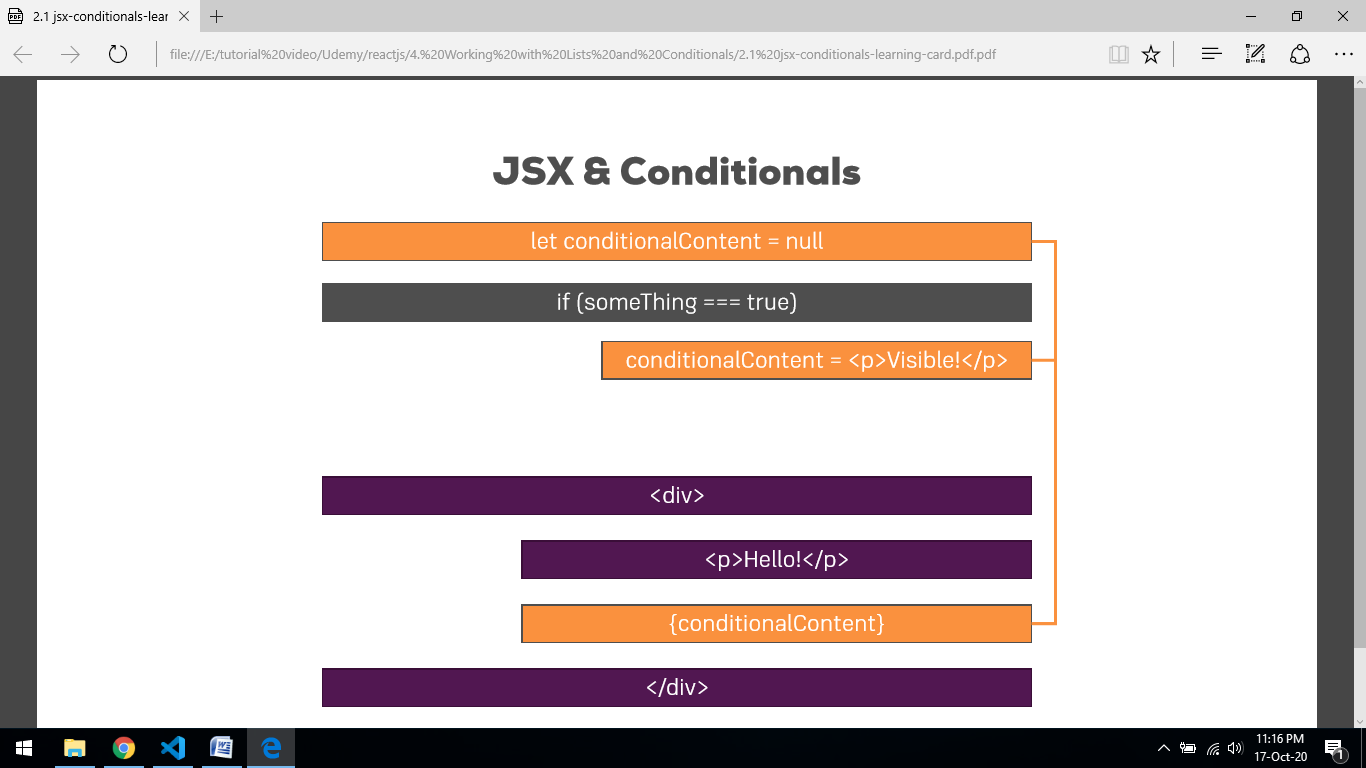
            <input type="text" onChange={props.changed} value={props.name} />

        </div>

    )

};

export default person;

****

**Looping List or conditional rendering of component**

Every list item must have unique key attribute

In react looping is done through map function

**IMP NOTE :** Do Looping of component or conditional component rendering inside the render method only.

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

class App extends Component {

  state = {

    persons: [

      { id: 'asfa1', name: 'Max', age: 28 },

      { id: 'vasdf1', name: 'Manu', age: 29 },

      { id: 'asdf11', name: 'Stephanie', age: 26 }

    ],

    otherState: 'some other value',

    showPersons: false

  }

  nameChangedHandler = ( event, id ) => {

    const personIndex = this.state.persons.findIndex(p => {

      return p.id === id;

    });

    const person = {

      ...this.state.persons[personIndex]

    };

    // const person = Object.assign({}, this.state.persons[personIndex]);

    person.name = event.target.value;

    const persons = [...this.state.persons];

    persons[personIndex] = person;

    this.setState( {persons: persons} );

  }

  deletePersonHandler = (personIndex) => {

    // const persons = this.state.persons.slice();

    const persons = [...this.state.persons];

    persons.splice(personIndex, 1);

    this.setState({persons: persons});

  }

  togglePersonsHandler = () => {

    const doesShow = this.state.showPersons;

    this.setState( { showPersons: !doesShow } );

  }

  render () {

    let persons = null;

    if ( this.state.showPersons ) {

      persons = (

        <div>

          {this.state.persons.map((person, index) => {

            return <Person

              click={() => this.deletePersonHandler(index)}

              name={person.name}

              age={person.age}

              key={person.id}

              changed={(event) => this.nameChangedHandler(event, person.id)} />

          })}

        </div>

      );

    }

    return (

      <div className="App">

        <h1>Hi, I'm a React App</h1>

        <p>This is really working!</p>

        <button

          style={style}

          onClick={this.togglePersonsHandler}>Toggle Persons</button>

        {persons}

      </div>

    );

  }

}

export default App;

**Style Components**

**1)add conditional classes**

Inside render method

 const classes = [];

    if ( this.state.persons.length <= 2 ) {

      classes.push( 'red' ); // classes = ['red']

    }

    if ( this.state.persons.length <= 1 ) {

      classes.push( 'bold' ); // classes = ['red', 'bold']

    }

 <div className="App">

 <p className={classes.join( ' ' )}>This is really working!</p>

</div>

**2)To add pesudo class in the component we use radium**

npm install radium

\*It allow us to work with inline styling, media query and pseudo class

\*Wrap the component with Radium

export default Radium( App );

\*If we use media query or any animation css then we have to wrap the entire application with special component StyleRoot

Person.js

import React from 'react';

import Radium from 'radium';

import './Person.css';

const person = ( props ) => {

    const style = {

        '@media (min-width: 500px)': {

            width: '450px'

        }

    };

    return (

        <div className="Person" style={style}>

            <p onClick={props.click}>I'm {props.name} and I am {props.age} years old!</p>

            <p>{props.children}</p>

            <input type="text" onChange={props.changed} value={props.name} />

        </div>

    )

};

export default Radium(person);

App.js (uses pseudo classes)

import React, { Component } from 'react';

import Radium, { StyleRoot } from 'radium';

class App extends Component {

  state = {

    persons: [

      { id: 'asfa1', name: 'Max', age: 28 },

      { id: 'vasdf1', name: 'Manu', age: 29 },

      { id: 'asdf11', name: 'Stephanie', age: 26 }

    ],

    otherState: 'some other value',

    showPersons: false

  }

  togglePersonsHandler = () => {

    const doesShow = this.state.showPersons;

    this.setState( { showPersons: !doesShow } );

  }

  render () {

    const style = {

      backgroundColor: 'green',

      color: 'white',

      font: 'inherit',

      border: '1px solid blue',

      padding: '8px',

      cursor: 'pointer',

      ':hover': {

        backgroundColor: 'lightgreen',

        color: 'black'

      }

    };

    let persons = null;

    if ( this.state.showPersons ) {

      persons = (

        <div>

          {this.state.persons.map( ( person, index ) => {

            return <Person

              click={() => this.deletePersonHandler( index )}

              name={person.name}

              age={person.age}

              key={person.id}

              changed={( event ) => this.nameChangedHandler( event, person.id )} />

          } )}

        </div>

      );

style.backgroundColor = 'red';

      style[':hover'] = {

        backgroundColor: 'salmon',

        color: 'black'

      };

    }

    const classes = [];

    if ( this.state.persons.length <= 2 ) {

      classes.push( 'red' ); // classes = ['red']

    }

    if ( this.state.persons.length <= 1 ) {

      classes.push( 'bold' ); // classes = ['red', 'bold']

    }

    return (

      <StyleRoot>

        <div className="App">

          <h1>Hi, I'm a React App</h1>

          <p className={classes.join( ' ' )}>This is really working!</p>

          <button

            style={style}

            onClick={this.togglePersonsHandler}>Toggle Persons</button>

        </div>

      </StyleRoot>

    );

  }

}

export default Radium( App );

3)Styled-component

npm I –save styled-components

import React, { Component } from 'react';

import styled from 'styled-components';

import './App.css';

const StyledButton = styled.button`

  background-color: ${props => props.alt ? 'red' : 'green'};

  color: white;

  font: inherit;

  border: 1px solid blue;

  padding: 8px;

  cursor: pointer;

  &:hover {

    background-color: ${props => props.alt ? 'salmon' : 'lightgreen'};

    color: black;

  }

`;

class App extends Component {

  state = {

    otherState: 'some other value',

    showPersons: false

  };

  togglePersonsHandler = () => {

    const doesShow = this.state.showPersons;

    this.setState({ showPersons: !doesShow });

  };

  render() {

    const style = {

      backgroundColor: 'green',

      color: 'white',

      font: 'inherit',

      border: '1px solid blue',

      padding: '8px',

      cursor: 'pointer',

      ':hover': {

        backgroundColor: 'lightgreen',

        color: 'black'

      }

    };

    return (

      <div className="App">

        <h1>Hi, I'm a React App</h1>

        <p > This is really working!</p>

        <StyledButton alt={this.state.showPersons} onClick={this.togglePersonsHandler}>

          Toggle Persons

        </StyledButton>

        {persons}

      </div>

    );

  }

}

export default App;

4)CSS module (Recommended way)

\*Create css file with name like that Person.module.css

\* import classes from “./Person.module.css”

\* <div className={classes.Red}></div>

\*If we use module approach never give classs name as string

<div className={classes.Red}></div> Correct

<div className=”Red”></div> Incorrect

**Debugging in react**

1. Use chrome debugger
2. Use react dev tools extension
3. Read the error message carefully
4. Custom ErrorBoundry component to handle error

\*ErrorBoundry component it works in production mode only

\*Wrap the component that generate error (Not always :in critical situations only)

ErroBoundry.js

import React, { Component } from 'react';

class ErrorBoundary extends Component {

    state = {

        hasError: false,

        errorMessage: ''

    }

    componentDidCatch = (error, info) => {

        this.setState({hasError: true, errorMessage: error});

    }

    render() {

        if (this.state.hasError) {

            return <h1>{this.state.errorMessage}</h1>;

        } else {

            return this.props.children;

        }

    }

}

export default ErrorBoundary;

Person.js (throw error)

import React from 'react';

import classes from './Person.css';

const person = ( props ) => {

    const rnd = Math.random();

    if ( rnd > 0.7 ) {

        throw new Error( 'Something went wrong' );

    }

    return (

        <div className={classes.Person}>

            <p onClick={props.click}>I'm {props.name} and I am {props.age} years old!</p>

            <p>{props.children}</p>

            <input type="text" onChange={props.changed} value={props.name} />

        </div>

    )

};

export default person;

App.js (wrap person with ErrorBoundry)

 persons = (

        <div>

          {this.state.persons.map( ( person, index ) => {

            return <ErrorBoundary key={person.id}>

              <Person

                click={() => this.deletePersonHandler( index )}

                name={person.name}

                age={person.age}

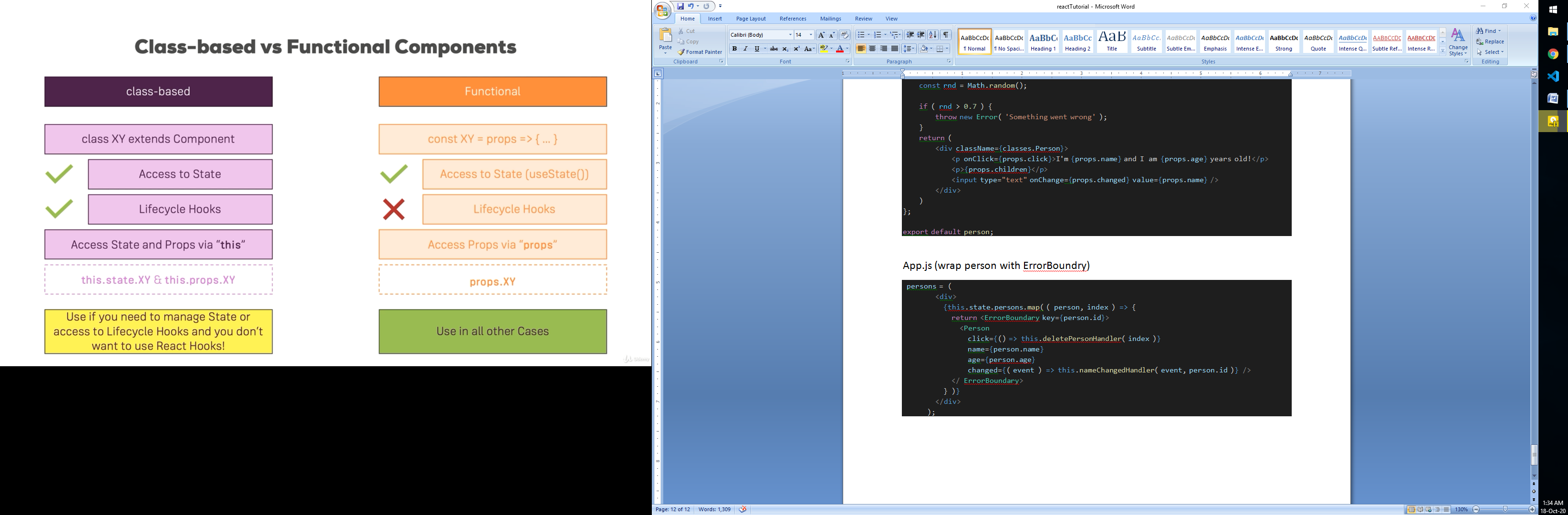
                changed={( event ) => this.nameChangedHandler( event, person.id )} />

            </ ErrorBoundary>

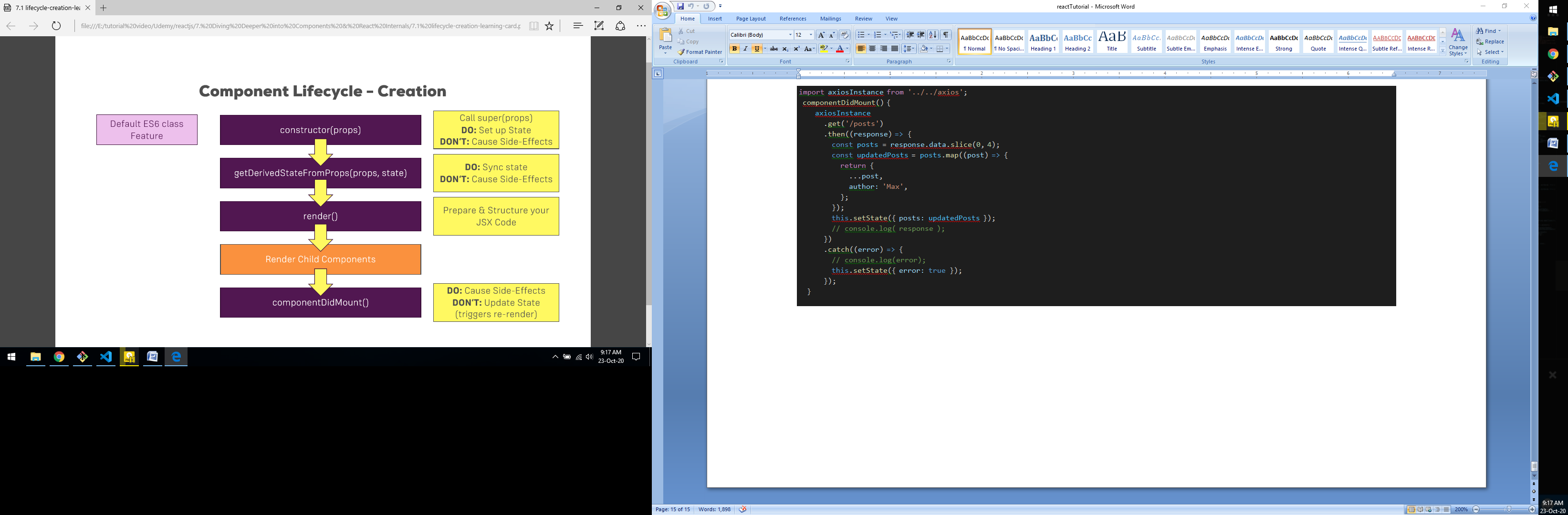
          } )}

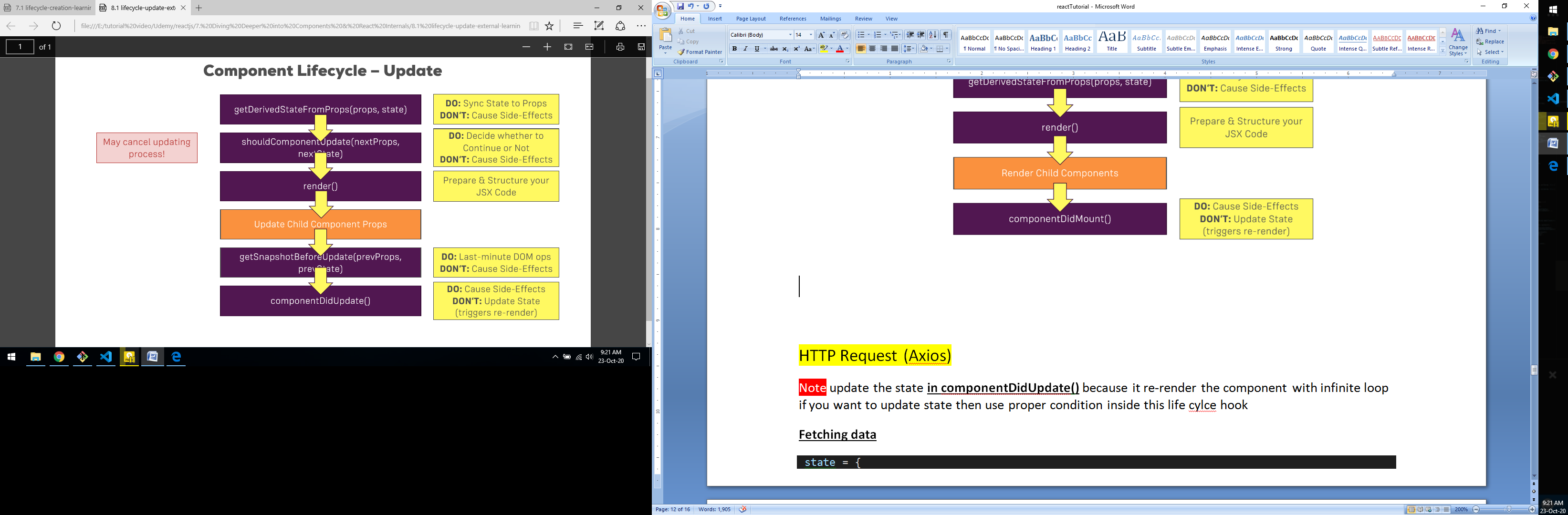
        </div>

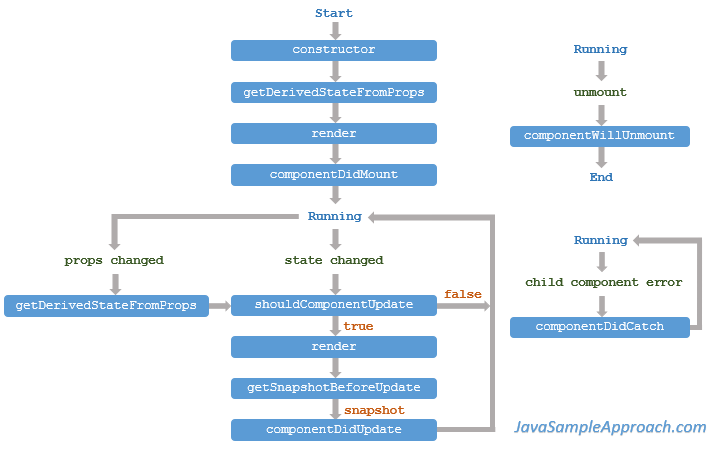
      );



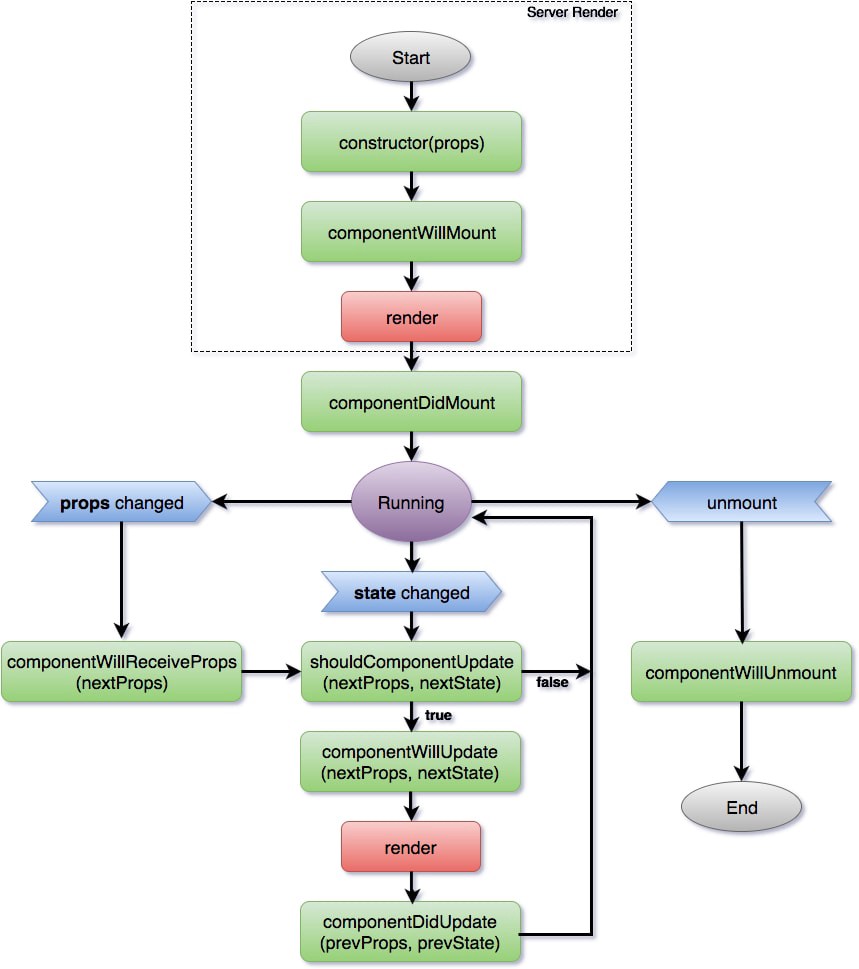
Life cycle hooks of component based class







Lifecycle hook for old version of hooks



Remove class based component (cleanup work)

componentWillUnmount(){

}

HTTP Request (Axios)

Note Do not update the state **in componentDidUpdate()** because it re-render the component with infinite loop if you want to update state then use proper condition inside this life cylce hook

**Fetching data**

 state = {

        posts: [],

        selectedPostId: null

    }

    componentDidMount () {

        axios.get( 'https://jsonplaceholder.typicode.com/posts' )

            .then( response => {

                const posts = response.data.slice(0, 4);

                const updatedPosts = posts.map(post => {

                    return {

                        ...post,

                        author: 'Max'

                    }

                });

                this.setState({posts: updatedPosts});

                // console.log( response );

            } );

    }

Cancel Previous Http Request using axios

useEffect(() => {

    const CancelToken = axios.CancelToken;

    const source = CancelToken.source();

    axios

      .get("https://jsonplaceholder.typicode.com/posts", {

        cancelToken: source.token,

      })

      .then((res) => {

        setPosts(res.data);

      })

      .catch(function (thrown) {

        if (axios.isCancel(thrown)) {

          console.log("Request canceled", thrown.message);

        } else {

          console.log(thrown);

        }

      });

    return () => {

      source.cancel("Request Cancel Manually");

    };

  }, []);

**Fetching single data inside component and note the componentDidUpdate**

Put condition inside componentDidUpdate to component rerendring

class FullPost extends Component {

    state = {

        loadedPost: null

    }

    componentDidUpdate () {

        if ( this.props.id ) {

            if ( !this.state.loadedPost || (this.state.loadedPost && this.state.loadedPost.id !== this.props.id) ) {

                axios.get( 'https://jsonplaceholder.typicode.com/posts/' + this.props.id )

                    .then( response => {

                        // console.log(response);

                        this.setState( { loadedPost: response.data } );

                    } );

            }

        }

    }

    render () {

        let post = <p style={{ textAlign: 'center' }}>Please select a Post!</p>;

        if ( this.props.id ) {

            post = <p style={{ textAlign: 'center' }}>Loading...!</p>;

        }

        if ( this.state.loadedPost ) {

            post = (

                <div className="FullPost">

                    <h1>{this.state.loadedPost.title}</h1>

                    <p>{this.state.loadedPost.content}</p>

                    <div className="Edit">

                        <button className="Delete">Delete</button>

                    </div>

                </div>

            );

        }

        return post;

    }

}

export default FullPost;

**Post Request**

state = {

        title: '',

        content: '',

        author: 'Max'

    }

    postDataHandler = () => {

        const data = {

            title: this.state.title,

            body: this.state.content,

            author: this.state.author

        };

        axios.post('https://jsonplaceholder.typicode.com/posts', data)

            .then(response => {

                console.log(response);

            });

    }

Delete Data

 deletePostHandler = () => {

        axios.delete('https://jsonplaceholder.typicode.com/posts/' + this.props.id)

            .then(response => {

                console.log(response);

            });

    }

Error Handling

 state = {

        posts: [],

        selectedPostId: null,

        error: false

    }

    componentDidMount () {

        axios.get( 'https://jsonplaceholder.typicode.com/posts' )

            .then( response => {

                const posts = response.data.slice(0, 4);

                const updatedPosts = posts.map(post => {

                    return {

                        ...post,

                        author: 'Max'

                    }

                });

                this.setState({posts: updatedPosts});

                // console.log( response );

            } )

            .catch(error => {

                // console.log(error);

                this.setState({error: true});

            });

    }

    postSelectedHandler = (id) => {

        this.setState({selectedPostId: id});

    }

render(){

let posts = <p style={{textAlign: 'center'}}>Something went wrong!</p>;

        if (!this.state.error) {

            posts = this.state.posts.map(post => {

                return <Post

                    key={post.id}

                    title={post.title}

                    author={post.author}

                    clicked={() => this.postSelectedHandler(post.id)} />;

            });

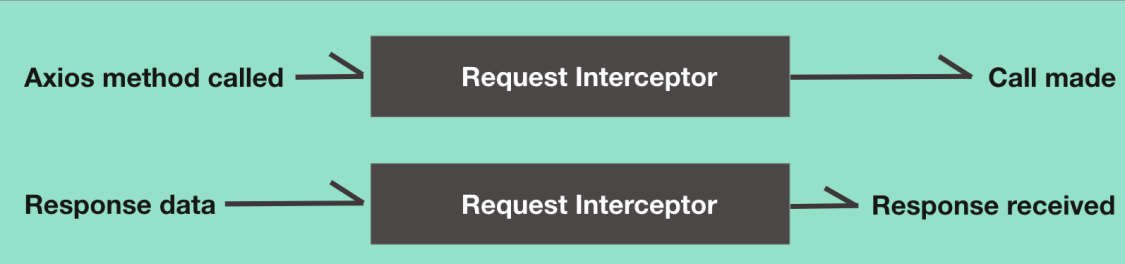
        }

}

Interceptors

Interceptors are methods which are triggered before the main method. There are two types of interceptors:

* **request interceptor**: this is called before the actual call to the endpoint is made.
* **response interceptor**: this is called before the promise is completed and the data is received by the then callback.

Think of the interceptor as a tunnel between the request/response and actual promise.

**Global Interceptors**

It is call on every request call and response recieved

**Index.js**

import axios from 'axios';

import './index.css';

import App from './App';

import registerServiceWorker from './registerServiceWorker';

axios.interceptors.request.use(

  (request) => {

    console.log('Request Interceptor');

    console.log(request);

    return request;

  },

  (error) => {

    console.log(error);

    return Promise.reject(error);

  }

);

axios.interceptors.response.use(

  (response) => {

    console.log('Response Interceptor');

    console.log(response);

    return response;

  },

  (error) => {

    console.log(error);

    return Promise.reject(error);

  }

);

**Setting a Default global configuration for axios**

Index.js

axios.defaults.baseURL = 'https://jsonplaceholder.typicode.com';

axios.defaults.headers.common['Authorization'] = 'Auth Token';

axios.defaults.headers.post['Content-Type'] = 'application/json';

**Axios Instance**

It overrides the default global configuration

Create axios.js file

import axios from 'axios';

const instance = axios.create({

  baseURL: 'https://jsonplaceholder.typicode.com',

  headers: {

    common: {

      Authorization: 'AUTH\_TOKEN\_FROM\_INSTANCE',

    },

  },

});

instance.interceptors.request.use(

  (resquest) => {

    console.log('Instance Request');

  },

  (error) => {

    return Promise.reject(error);

  }

);

export default instance;

import axiosInstance from '../../axios';

 componentDidMount() {

    axiosInstance

      .get('/posts')

      .then((response) => {

        const posts = response.data.slice(0, 4);

        const updatedPosts = posts.map((post) => {

          return {

            ...post,

            author: 'Max',

          };

        });

        this.setState({ posts: updatedPosts });

        // console.log( response );

      })

      .catch((error) => {

        // console.log(error);

        this.setState({ error: true });

      });

  }

**Higher component with axios**

**axios-order.js**

import axios from 'axios';

const instance = axios.create({

  baseURL: 'https://react-burger-7fe72.firebaseio.com/',

});

export default instance;

**burgerBuilder.js**

if any error regarding axios is occurred inside burgerBuilder will be handle by withErrorHanlder component

import React, { Component } from 'react';

import Aux from '../../hoc/Auxiliary/Auxiliary';

import Burger from '../../components/Burger/Burger';

import BuildControls from '../../components/Burger/BuildControls/BuildControls';

import Modal from '../../components/UI/Modal/Modal';

import OrderSummary from '../../components/OrderSummary/OrderSummary';

import Spinner from '../../components/UI/Spinner/Spinner';

import withErrorHandler from '../../hoc/withErrorHandler/withErrorHandler';

import axios from '../../axios-order';

const INGREDIENT\_PRICES = {

  salad: 0.5,

  cheese: 0.4,

  meat: 1.3,

  bacon: 0.7,

};

class BurgerBuilder extends Component {

  state = {

    ingredients: null,

    totalPrice: 4,

    purchasable: false,

    purchasing: false,

    loading: false,

    error: false,

  };

  componentDidMount() {

    axios

      .get('https://react-burger-7fe72.firebaseio.com/ingredients.json')

      .then((response) => {

        this.setState({ ingredients: response.data });

      })

      .catch((error) => {

        this.setState({ error: true });

      });

  }

  render() {

    let orderSummary = null;

    let burger = this.state.error ? (

      <p>Ingredients can't be loaded!</p>

    ) : (

      <Spinner />

    );

    if (this.state.ingredients) {

      burger = (

        <Aux>

          <Burger ingredients={this.state.ingredients} />

          <BuildControls

          />

        </Aux>

      );

      orderSummary = (

        <OrderSummary

          ingredients={this.state.ingredients}

        />

      );

    }

    // {salad: true, meat: false, ...}

    return (

      <Aux>

        <Modal

          show={this.state.purchasing}

          modalClosed={this.purchaseCancelHandler}

        >

          {orderSummary}

        </Modal>

        {burger}

      </Aux>

    );

  }

}

export default withErrorHandler(BurgerBuilder, axios);

**withErrorHandler.js (HOC)**

import React, { Component } from 'react';

import Modal from '../../components/UI/Modal/Modal';

import Aux from '../Auxiliary/Auxiliary';

const withErrorHandler = (WrappedComponent, axios) => {

  return class extends Component {

    state = {

      error: null,

    };

    componentWillMount() {

      this.reqInterceptor = axios.interceptors.request.use((req) => {

        this.setState({ error: null });

        return req;

      });

      this.resInterceptor = axios.interceptors.response.use(

        (res) => res,

        (error) => {

          this.setState({

            error: error,

          });

        }

      );

    }

    componentWillUnmount() {

      axios.interceptors.request.eject(this.reqInterceptor);

      axios.interceptors.response.eject(this.resInterceptor);

    }

    errorConfirmHandler = () => {

      this.setState({

        error: null,

      });

    };

    render() {

      return (

        <Aux>

          <Modal show={this.state.error} modalClosed={this.errorConfirmHandler}>

            {this.state.error ? this.state.error.message : null}

          </Modal>

          <WrappedComponent {...this.props} />

        </Aux>

      );

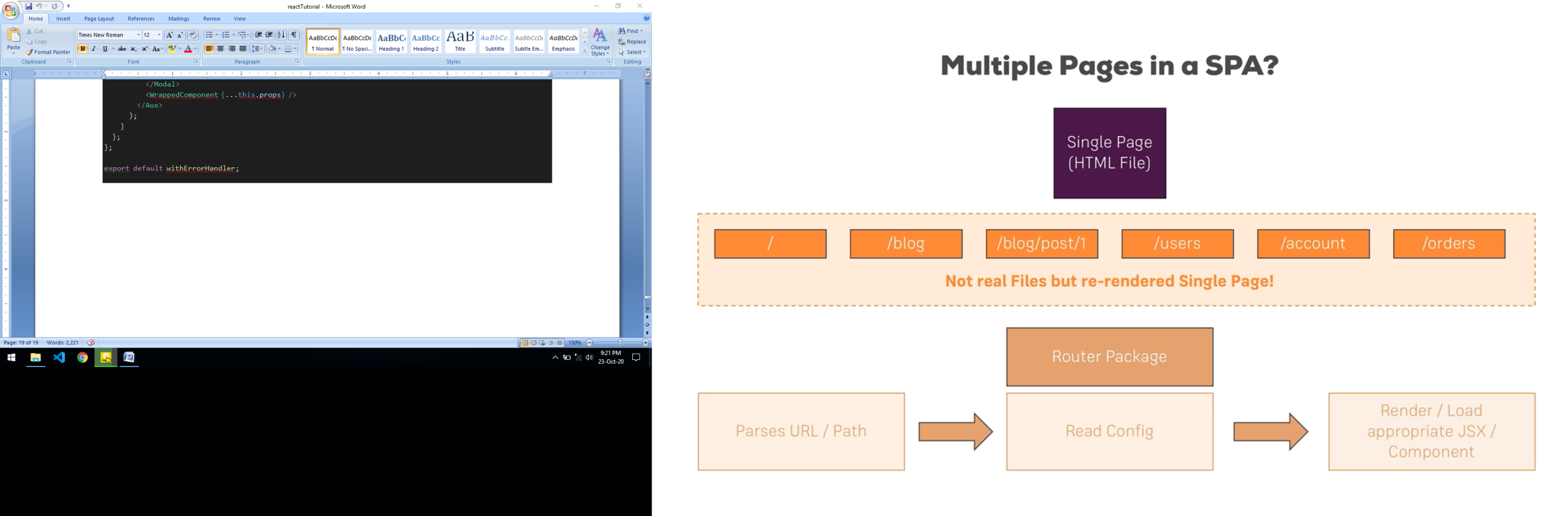
    }

  };

};

export default withErrorHandler;

**Routing**

****

**Install routing package**

**npm install --save react-router react-router-dom**

**Steps to configure Routes**

**1)To enable routing in our app wrap the application with router (app.js)**

import React, { Component } from 'react';

import { BrowserRouter as Router } from 'react-router-dom';

import Blog from './containers/Blog/Blog';

class App extends Component {

  render() {

    return (

      <Router>

        <div className='App'>

          <Blog />

        </div>

      </Router>

    );

  }

}

export default App;

**Blog.js**

import React, { Component } from 'react';

// import axios from 'axios';

import { Route, Link } from 'react-router-dom';

import './Blog.css';

import Posts from './Posts/Posts';

import NewPost from './NewPost/NewPost';

class Blog extends Component {

  render() {

    return (

      <div className='Blog'>

            <ul>

              <li>

                <Link to='/'>Home</Link>

              </li>

              <li>

                <Link

                  to={{

                    pathname: '/new-post',

                    search: '?quiz-submit=true',

                    hash: '#submit',

                  }}

                >

                  New Post

                </Link>

              </li>

            </ul>

        {/\* <Route path='/' exact render={() => <h1>Home</h1>} />

        <Route path='/' render={() => <h1>Home 2</h1>} /> \*/}

        <Route path='/' exact component={Posts} />

        <Route path='/new-post' exact component={NewPost} />

      </div>

    );

  }

}

export default Blog;

**2)withRouter**

\*When any component load with Route component it gets extra props from routing

<Route path='/' exact component={Posts} />

match,history,location and staticContext

**\***When any component load without route component it do not get extra routing props to access this extra routing props we use hoc to wrap component

Import { withRouter } from ‘react-router-dom’;

export default withRouter(Post);

**3)Absolute and relative path**

<Link

                  to={{

                    pathname: this.props.match.url+'/new-post', //relative path

                    search: '?quiz-submit=true',

                    hash: '#submit',

                  }}

                >

                  New Post

                </Link>

<Link

                  to={{

                    pathname:  '/new-post', //absolute path – ‘new-post’, ‘/new-post’

                    search: '?quiz-submit=true',

                    hash: '#submit',

                  }}

                >

                  New Post

                </Link>

**4)Adding active class on link**

import { Route, NavLink } from 'react-router-dom';

  <li>

                <NavLink to='/' exact>

                  Home

                </NavLink>

              </li>

              <li>

                <NavLink

                  to={{

                    pathname: '/new-post',

                    search: '?quiz-submit=true',

                    hash: '#submit',

                  }}

                >

                  New Post

                </NavLink>

              </li>

\*Use NavLink instead of Link to add active class

\*To rename active class

 <NavLink to='/' exact activeClassName='my-active'>

                  Home

                </NavLink>

\*Inline styling of active link

 <NavLink

                  to='/'

                  exact

                  activeStyle={{

                    textDecoration: 'underline',

                    color: '#ff0000',

                  }}

                >

                  Home

                </NavLink>

5)**Parametric Route**

<Route path='/:id' component={FullPost} />

\*define all the dynamic routes below all the static routes

\* get the parameter from url

this.props.match.params.id

componentDidMount() {

    if (this.props.match.params.id) {

      if (

        !this.state.loadedPost ||

        (this.state.loadedPost &&

          this.state.loadedPost.id !== this.props.match.params.id)

      ) {

        axios.get('/posts/' + this.props.match.params.id).then((response) => {

          // console.log(response);

          this.setState({ loadedPost: response.data });

        });

      }

    }

  }

6)**To match exact route we use Switch**

 <Switch>

          <Route path='/' exact component={Posts} />

          <Route path='/new-post' exact component={NewPost} />

          <Route path='/:id' component={FullPost} />

        </Switch>

\*switch stop matching after selecting first route

\* we can also mix routes with switch

 <Route path='/' exact component={Posts} />

        <Switch>

          <Route path='/new-post' exact component={NewPost} />

        </Switch>

        <Route path='/:id' component={FullPost} />

7)**Programatic Navigation**

postSelectedHandler = (id) => {

    this.props.history.push(`/${id}`);

    //this.props.history.push({ pathname: `/${id}` });

  };

 purchaseContinueHandler = () => {

    const queryParams = [];

    for (const i in this.state.ingredients) {

      queryParams.push(`${encodeURIComponent(i)}=${this.state.ingredients[i]}`);

    }

    queryParams.push(`price=${this.state.totalPrice.toFixed(2)}`);

    const queryString = queryParams.join('&');

    this.props.history.push({

      pathname: '/checkout',

      search: `?${queryString}`,

    });

  };

8)**Nested Routes**

Blog.js (Don’t put exact on /posts route)

 <Switch>

          <Route path='/new-post' component={NewPost} />

          <Route path='/posts' component={Posts} />

        </Switch>

Posts.js

 render() {

    return (

      <div>

        <section className='Posts'>{posts}</section>

        <Route

          path={`${this.props.match.url}/:id`}

          exact

          component={FullPost}

        />

      </div>

    );

  }

9)**Redirect Route**

import { Route, NavLink, Switch, Redirect } from 'react-router-dom';

<Switch>

          <Route path='/new-post' component={NewPost} />

          <Route path='/posts' component={Posts} />

          <Redirect from='/' to='/posts' exact />

                  </Switch>

\*it redirect / to /posts

\*Note from attribute of Redirect is only work inside Switch

\*to attribute of Redirect outside Switch redirect the route (Used in Conditional redirect)

\*Redirect also replace the current state of browser url (We can not go back by browser)

10)**Conditional redirect**

import { Route, NavLink, Switch, Redirect } from 'react-router-dom';

render() {

    let redirect = null;

    if (this.state.submitted) {

      redirect = <Redirect to='/posts' />;

    }

    return (

      <div className='NewPost'>

        {redirect}

</div>)

}

11) **Difference b/w history.push and history.replace**

\*this.props.history.push(‘/posts’) push another location in the stack (we can go back by browser)

\*this.props.history.replace(‘/posts’) replace the current location (we can not go back by browser) similar to Redirect

\*this.props.history.goBack() it is used to go back to previous route

12)**404 page not found**

Define route below all the routes

<Route render={() => <h1>Not Found</h1>} />

Or

<Redirect from='\*' to='/page-not-found' />

<Switch>

          <Route path='/new-post' component={NewPost} />

          <Route path='/posts' component={Posts} />

          <Route path='/page-not-found' component={PageNotFound} />

          <Redirect from='/' to='/posts' exact />

          <Redirect from='\*' to='/page-not-found' />

        </Switch>

13)**Lazy loading or code splitting (Old react version)**

Create HOC asynchComponent.js

Step1

import React, { Component } from 'react';

const asynchComponent = (importFun) => {

  return class extends Component {

    state = {

      component: null,

    };

    componentDidMount() {

      importFun().then((cmp) => {

        this.setState({ component: cmp.default });

      });

    }

    render() {

      const C = this.state.component;

      return C ? <C {...this.props} /> : null;

    }

  };

};

export default asynchComponent;

Blog.js

Step2 define function inside

import asynchComponent from '../../hoc/asynchComponent';

const AsyncNewPost = asynchComponent(() => {

  return import('./NewPost/NewPost');

});

Step 3

<Switch>

          <Route path='/new-post' component={AsyncNewPost} />

          <Route path='/posts' component={Posts} />

          <Redirect from='/' to='/posts' exact />

          <Route render={() => <h1>Page Not Found</h1>} />

        </Switch>

14) **Lazy loading react version above 16.6**

Step1

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

import { BrowserRouter, Route, NavLink } from 'react-router-dom';

const Posts = React.lazy(() => import('./containers/Posts'));

step2

 <Route

            path='/posts'

            render={() => (

              <Suspense fallback={<div>Loding...</div>}>

                <Posts />

              </Suspense>

            )}

          />

If Posts is functional component then we need to pass props inside the component render

<Route

            path='/posts'

            render={(props) => (

              <Suspense fallback={<div>Loding...</div>}>

                <Posts {…props} />

              </Suspense>

            )}

          />

15) **Conditional Lazy loading**

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

import { BrowserRouter, Route, NavLink } from 'react-router-dom';

const Posts = React.lazy(() => import('./containers/Posts'));

{this.state.show && (

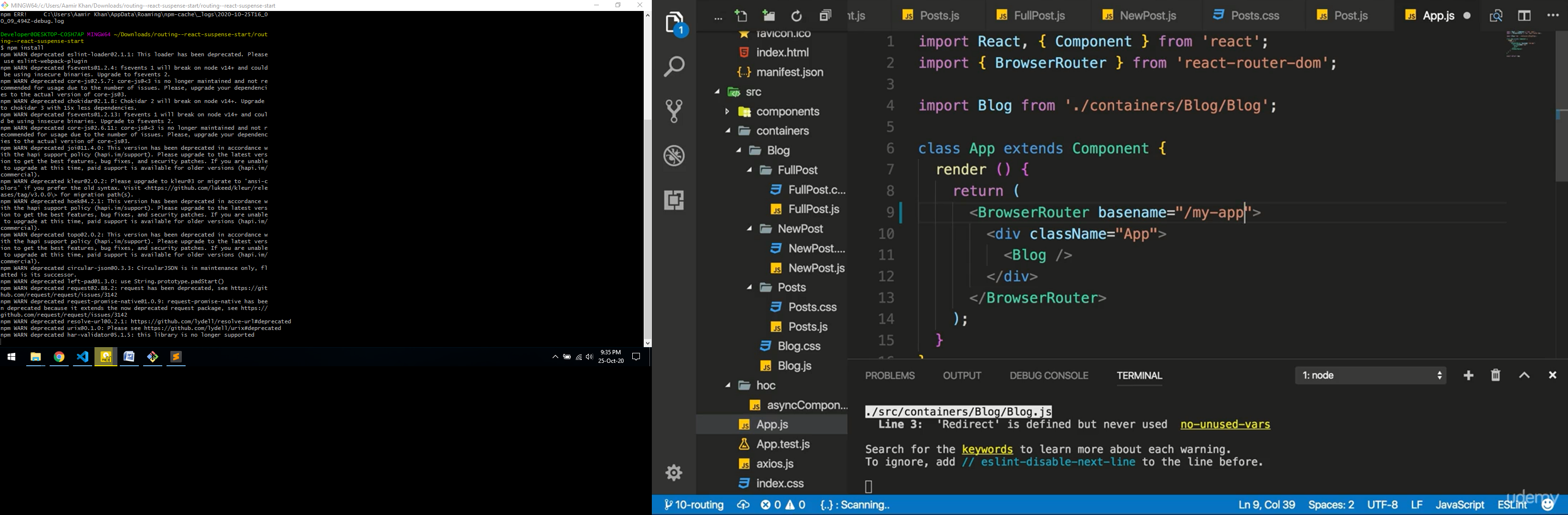
            <Suspense fallback={<div>Loading...</div>}>

              <Posts />

            </Suspense>

          )}

16) **If our react application is define inside the other folder then we have to define basename on BrowserRouter (it works on client side only)**



17) var params = new URLSearchParams("foo=1&bar=2");

URLSearchParams it takes query string convert it into iterator

 componentDidMount() {

    //?bacon=1&cheese=0&meat=0&salad=1

    //this.props.history.location.search ===?bacon=1&cheese=0&meat=0&salad=1

    const query = new URLSearchParams(this.props.history.location.search);

    const ingredients = {};

    for (const param of query.entries()) {

      //['salad','1']

      ingredients[param[0]] = +param[1];

    }

    this.setState({

      ingredients: ingredients,

    });

  }

Form validation in react

Custom form validation

Contactdata.js

import Input from '../../../components/UI/Input/Input';

export class ContactData extends Component {

//step 1 create structure of form

  state = {

    orderForm: {

      name: {

        elementType: 'input',

        elementConfig: {

          name: 'name',

          type: 'text',

          placeholder: 'Your name',

        },

        validation: {

          required: true,

        },

        valid: false,

        touched: false,

        value: '',

        label: 'Name',

      },

      email: {

        elementType: 'input',

        elementConfig: {

          name: 'email',

          type: 'email',

          placeholder: 'Your E-mail',

        },

        validation: {

          required: true,

          isEmail: true,

        },

        valid: false,

        touched: false,

        value: '',

        label: 'E-mail',

      },

      street: {

        elementType: 'input',

        elementConfig: {

          name: 'address',

          type: 'text',

          placeholder: 'Your Street Address',

        },

        validation: {

          required: true,

        },

        valid: false,

        touched: false,

        value: '',

        label: 'Street Address',

      },

      zipCode: {

        elementType: 'input',

        elementConfig: {

          name: 'zipcode',

          type: 'text',

          placeholder: 'Your zipcode',

        },

        validation: {

          required: true,

          minLength: 6,

          maxLength: 6,

          isNumeric: true,

        },

        valid: false,

        touched: false,

        value: '',

        label: 'Postal Code',

      },

      country: {

        elementType: 'input',

        elementConfig: {

          country: 'country',

          type: 'text',

          placeholder: 'Your Country',

        },

        validation: {

          required: true,

        },

        valid: false,

        touched: false,

        value: '',

        label: 'Country',

      },

      deliveryMethod: {

        elementType: 'select',

        elementConfig: {

          name: 'delivery',

          options: [

            { value: 'cheapest', displayValue: 'Cheapest' },

            { value: 'fastest', displayValue: 'Fastest' },

          ],

        },

        validation: {},

        valid: true,

        touched: false,

        value: 'fastest',

        label: 'Delivery',

      },

    },

    formIsValid: false,

    loading: false,

  };

  orderHandler = (event) => {

    event.preventDefault();

    this.setState({ loading: true });

    const formData = {};

    for (const inputKey in this.state.orderForm) {

      formData[inputKey] = this.state.orderForm[inputKey].value;

    }

    const order = {

      ingredients: this.props.ingredients,

      price: this.props.totalPrice,

      orderData: formData,

    };

    axios

      .post('/orders.json', order)

      .then((response) => {

        this.setState({ loading: false });

        this.props.history.replace('/');

      })

      .catch((error) => {

        this.setState({ loading: false });

      });

  };

//step 5

  checkValidity(value, rules) {

    if (!Object.keys(rules).length) {

      return true;

    }

    let isValid = true;

    if (rules.required) {

      isValid = value.trim() !== '' && isValid;

    }

    if (rules.minLength) {

      isValid = value.length >= rules.minLength && isValid;

    }

    if (rules.maxLength) {

      isValid = value.length <= rules.maxLength && isValid;

    }

    if (rules.isEmail) {

      const pattern = /[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?/;

      isValid = pattern.test(value) && isValid;

    }

    if (rules.isNumeric) {

      const pattern = /^\d+$/;

      isValid = pattern.test(value) && isValid;

    }

    return isValid;

  }

//step 4

  inputChangeHandler = (event, inputIdentifier) => {

    let formData = { ...this.state.orderForm };

    let updateForm = { ...formData[inputIdentifier] };

    updateForm.value = event.target.value;

    updateForm.touched = true;

    updateForm.valid = this.checkValidity(

      updateForm.value,

      updateForm.validation

    );

    formData[inputIdentifier] = updateForm;

    let formIsValid = true;

    for (let inputKey in formData) {

      formIsValid = formData[inputKey].valid && formIsValid;

    }

    this.setState({ orderForm: formData, formIsValid: formIsValid });

  };

  render() {

//step 2

    let formElements = [];

    for (const key in this.state.orderForm) {

      formElements.push({

        id: key,

        config: this.state.orderForm[key].elementConfig,

        value: this.state.orderForm[key].value,

        label: this.state.orderForm[key].label,

        type: this.state.orderForm[key].elementType,

        touched: this.state.orderForm[key].touched,

        valid: this.state.orderForm[key].valid,

        validation: this.state.orderForm[key].validation,

      });

    }

//step 3

    let form = (

      <form onSubmit={this.orderHandler}>

        {formElements.map((ele) => (

          <Input

            key={ele.id}

            elementType={ele.type}

            elementConfig={ele.config}

            value={ele.value}

            label={ele.label}

            inValid={!ele.valid}

            touched={ele.touched}

            shouldValidate={ele.validation}

            changed={(event) => this.inputChangeHandler(event, ele.id)}

          />

        ))}

        <Button disabled={!this.state.formIsValid} btnType='Success'>

          Order

        </Button>

      </form>

    );

    if (this.state.loading) {

      form = <Spinner />;

    }

    return (

      <div className={classes.ContactData}>

        <h2>Enter your Contact Data</h2>

        {form}

      </div>

    );

  }

}

export default ContactData;

input.js

import React from 'react';

import classes from './Input.module.css';

const Input = (props) => {

  let inputElement = null;

  let inputClasses = [classes.InputElement];

  if (

    props.inValid &&

    Object.keys(props.shouldValidate).length &&

    props.touched

  ) {

    inputClasses.push(classes.Invalid);

  }

  switch (props.elementType) {

    case 'input':

      inputElement = (

        <input

          onChange={props.changed}

          className={inputClasses.join(' ')}

          {...props.elementConfig}

          defaultValue={props.value}

        />

      );

      break;

    case 'select':

      inputElement = (

        <select

          onChange={props.changed}

          name={props.elementConfig.name}

          className={inputClasses.join(' ')}

          defaultValue={props.value}

        >

          {props.elementConfig.options.map((op) => (

            <option key={op.value} value={op.value}>

              {op.displayValue}

            </option>

          ))}

        </select>

      );

      break;

    case 'textarea':

      inputElement = (

        <textarea

          onChange={props.changed}

          className={inputClasses.join(' ')}

          {...props.elementConfig}

        >

          {props.value}

        </textarea>

      );

      break;

    default:

      inputElement = (

        <input

          onChange={props.changed}

          className={inputClasses.join(' ')}

          {...props.elementConfig}

          defaultValue={props.value}

        />

      );

      break;

  }

  return (

    <div className={classes.Input}>

      <label className={classes.Label}>{props.label}</label>

      {inputElement}

    </div>

  );

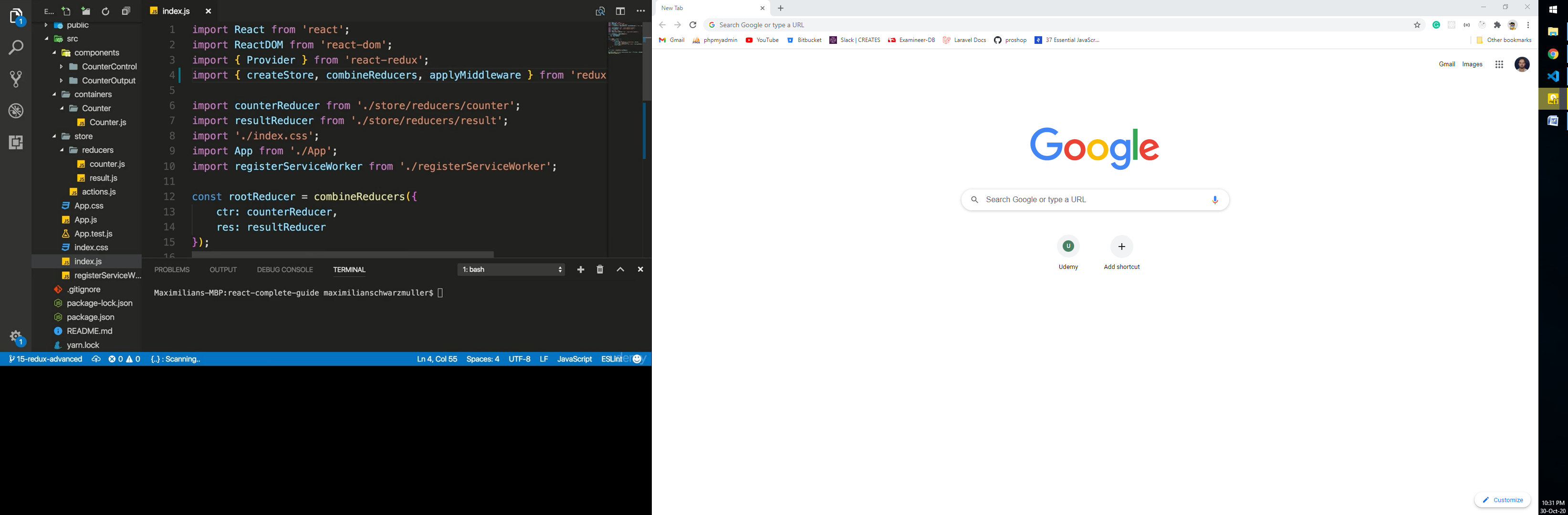
};

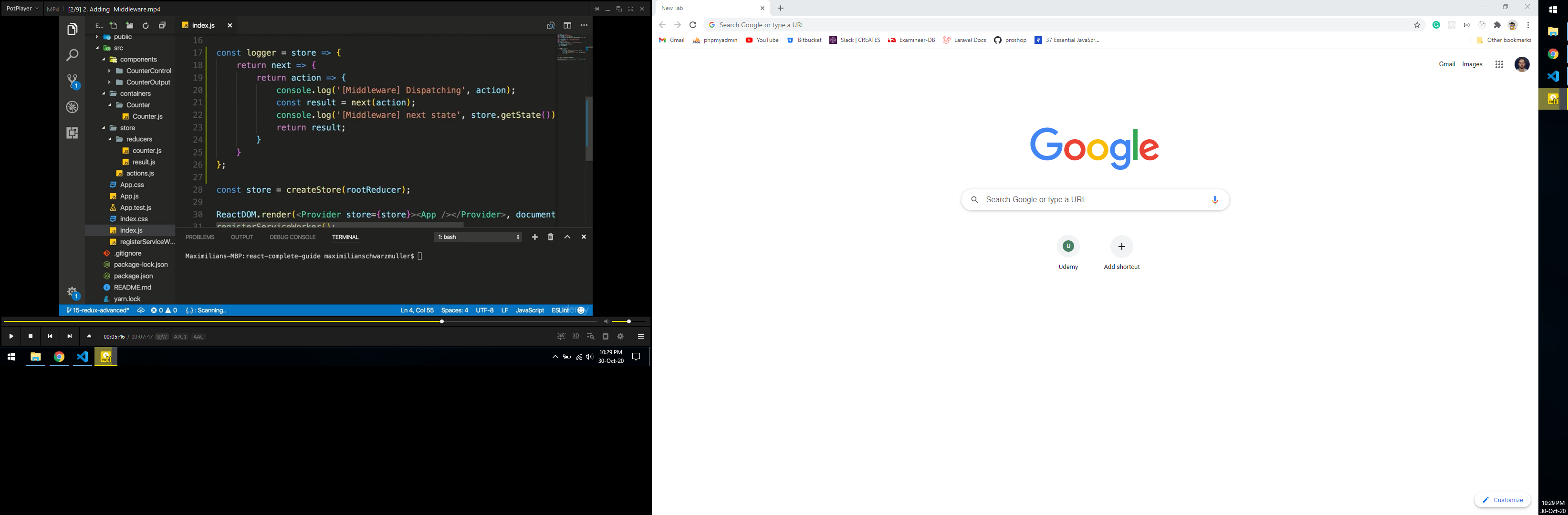
export default Input;

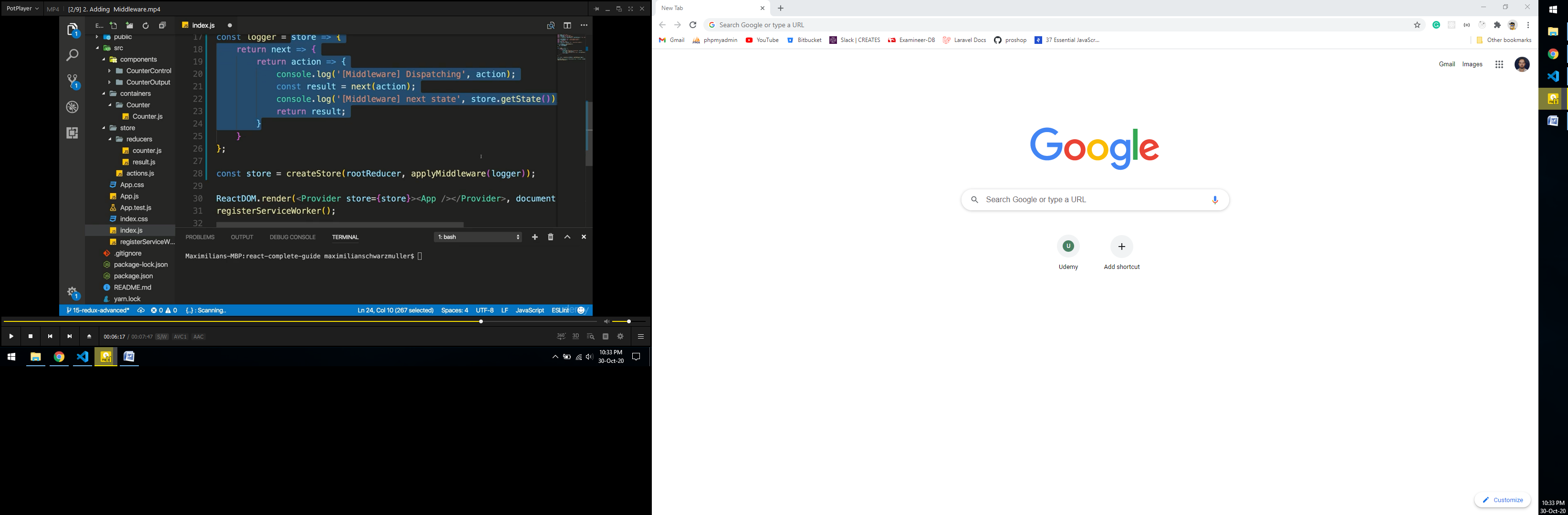
Redux

IMP NOTE: To update state of array inside the store we must use state.arr.concat(val). Don’t use push method it change the mutability of state.

Custom middleware pass through reducer



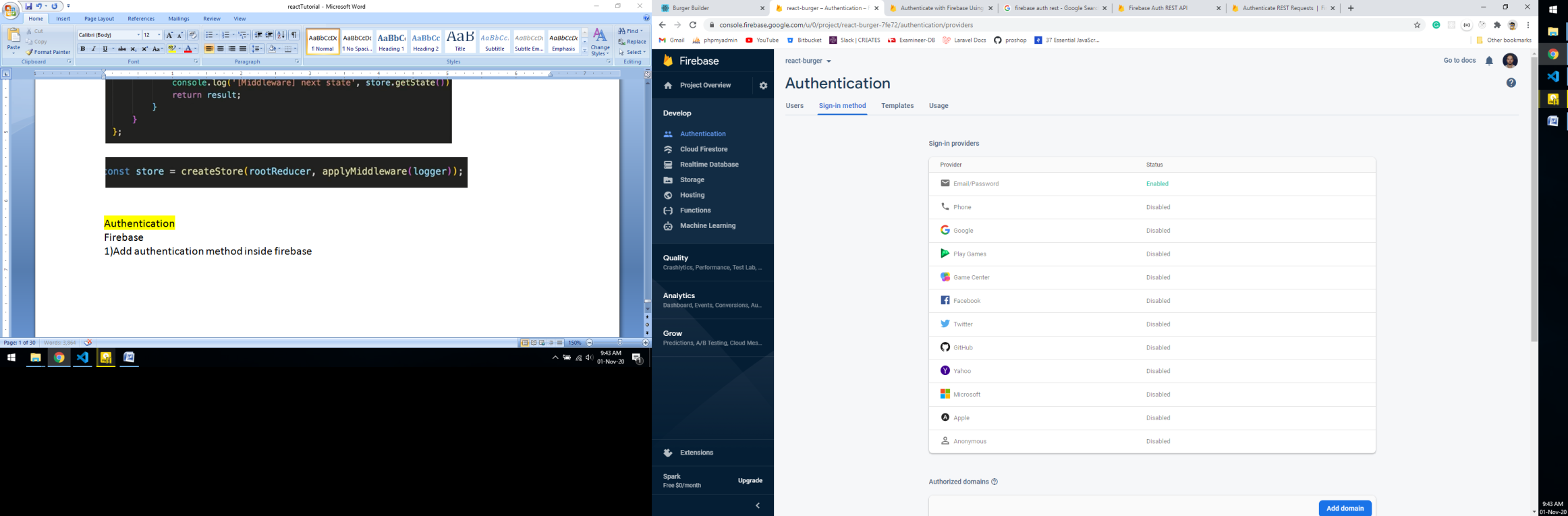




Authentication

Firebase

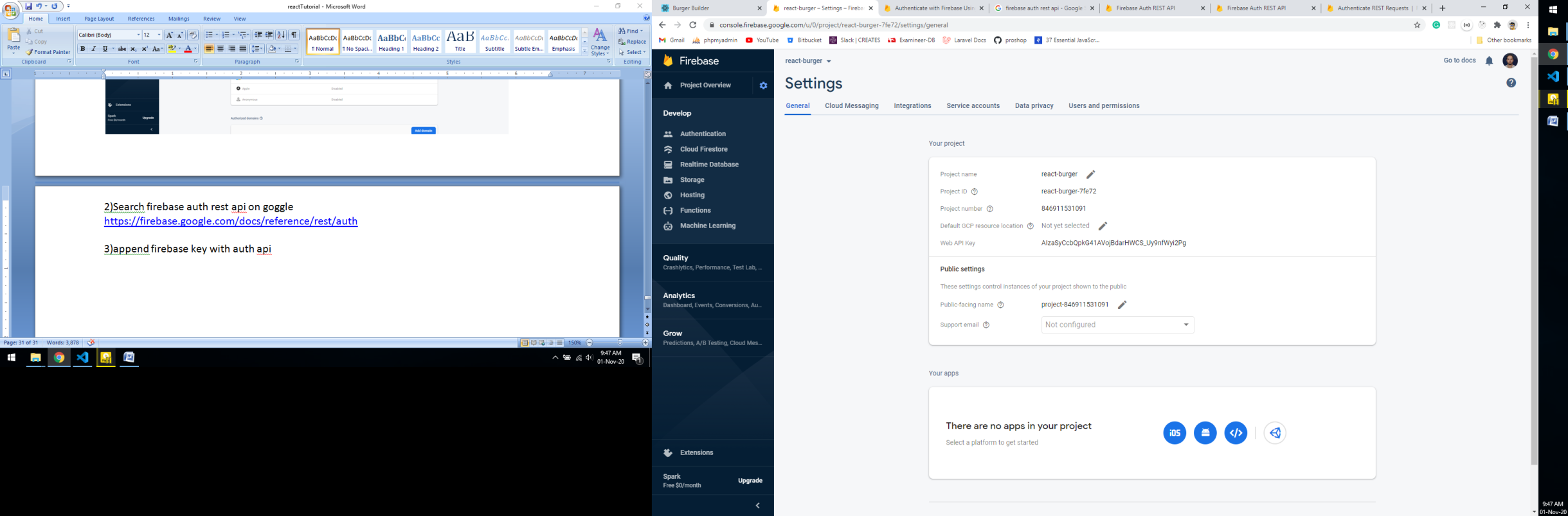
1)Add authentication method inside firebase



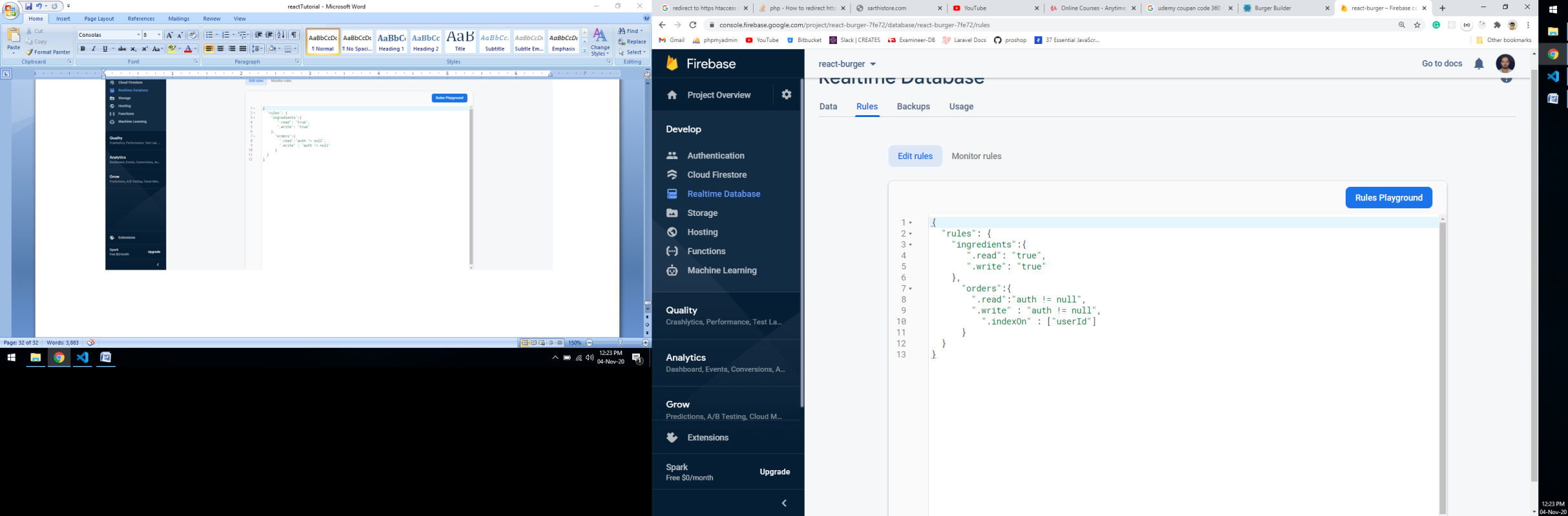
2)Search firebase auth rest api on goggle

<https://firebase.google.com/docs/reference/rest/auth>

3)append firebase key with auth api



Set Rules for authentication



"orders":{

".read":"auth != null", //not allow read without auth

".write" : "auth != null", //not allow write without auth

".indexOn" : ["userId"] //it filter orders on the basis of userId field (used in fetch records of user orders)

}

Auth.js component (Login and registration)

import React, { Component } from "react";

import { connect } from "react-redux";

import { Redirect } from "react-router-dom";

import Button from "../../components/UI/Button/Button";

import Input from "../../components/UI/Input/Input";

import Spinner from "../../components/UI/Spinner/Spinner";

import \* as authActions from "../../store/actions";

import { checkValidity } from "../../store/utility";

import classes from "./Auth.module.css";

export class Auth extends Component {

  state = {

    controls: {

      email: {

        elementType: "input",

        elementConfig: {

          type: "email",

          name: "email",

          placeholder: "Mail Address",

        },

        value: "",

        validation: {

          required: true,

          isEmail: true,

        },

        valid: false,

        touched: false,

      },

      password: {

        elementType: "input",

        elementConfig: {

          type: "password",

          name: "password",

          placeholder: "Your Password",

        },

        value: "",

        validation: {

          required: true,

          minLength: 5,

        },

        valid: false,

        touched: false,

      },

    },

    isSignup: true,

  };

  switchAuthModeHandler = () => {

    this.setState((preState) => {

      return { isSignup: !preState.isSignup };

    });

  };

  inputChangedHandler = (event, inputIdentifier) => {

    const updatedForm = {

      ...this.state.controls,

      [inputIdentifier]: {

        ...this.state.controls[inputIdentifier],

        value: event.target.value,

        valid: checkValidity(

          event.target.value,

          this.state.controls[inputIdentifier].validation

        ),

        touched: true,

      },

    };

    this.setState({ controls: updatedForm });

  };

  submitHandler = (event) => {

    event.preventDefault();

    this.props.onAuth(

      this.state.controls.email.value,

      this.state.controls.password.value,

      this.state.isSignup

    );

  };

  componentDidMount() {

    if (!this.props.buildingBurger && this.props.authRedirectPath !== "/") {

      this.props.onSetAuthRedirectPath();

    }

  }

  render() {

    const formElementsArray = [];

    for (let key in this.state.controls) {

      formElementsArray.push({

        id: key,

        config: this.state.controls[key],

      });

    }

    let form = (

      <form onSubmit={this.submitHandler}>

        <b>

          {this.state.isSignup ? "Create Account" : "Sign With Your Account"}

        </b>

        {formElementsArray.map((formElement) => (

          <Input

            key={formElement.id}

            elementType={formElement.config.elementType}

            elementConfig={formElement.config.elementConfig}

            value={formElement.config.value}

            invalid={!formElement.config.valid}

            shouldValidate={formElement.config.validation}

            touched={formElement.config.touched}

            changed={(event) => this.inputChangedHandler(event, formElement.id)}

          />

        ))}

        <Button btnType='Success'>Submit</Button>

      </form>

    );

    if (this.props.loading) {

      form = <Spinner />;

    }

    let errorMessage = null;

    if (this.props.error) {

      errorMessage = <p className={classes.Auth\_Error}>{this.props.error}</p>;

    }

    let authRedirect = null;

    if (this.props.isAuth) {

      authRedirect = <Redirect to={this.props.authRedirectPath} />;

    }

    return (

      <div className={classes.Auth}>

        {authRedirect}

        {errorMessage}

        {form}

        <Button btnType='Danger' clicked={this.switchAuthModeHandler}>

          Switch to {this.state.isSignup ? "Login" : "Register"}

        </Button>

      </div>

    );

  }

}

const mapStateToProps = (state) => {

  return {

    loading: state.auth.loading,

    error: state.auth.error,

    isAuth: state.auth.token !== null,

    buildingBurger: state.burgerBuilder.building,

    authRedirectPath: state.auth.authRedirectPath,

  };

};

const mapDispatchToProps = (dispatch) => {

  return {

    onAuth: (email, password, isSignup) =>

      dispatch(authActions.auth(email, password, isSignup)),

    onSetAuthRedirectPath: () => dispatch(authActions.setAuthRedirectPath("/")),

  };

};

export default connect(mapStateToProps, mapDispatchToProps)(Auth);

Auth Action.js

import axios from "axios";

import \* as actionTypes from "./actionTypes";

export const authStart = () => {

  return {

    type: actionTypes.AUTH\_START,

  };

};

export const authSuccess = (token, userId) => {

  return {

    type: actionTypes.AUTH\_SUCCESS,

    idToken: token,

    localId: userId,

  };

};

export const authFail = (error) => {

  return {

    type: actionTypes.AUTH\_FAIL,

    error: error,

  };

};

export const loggout = () => {

  localStorage.removeItem("token");

  localStorage.removeItem("userId");

  localStorage.removeItem("expirationDate");

  return {

    type: actionTypes.AUTH\_LOGGOUT,

  };

};

export const checkAuthTimeOut = (expiresIn) => (dispatch) => {

  setTimeout(() => {

    dispatch(loggout());

  }, expiresIn);

};

export const auth = (email, password, isSignup) => async (dispatch) => {

  dispatch(authStart());

  try {

    const authData = {

      email,

      password,

      returnSecureToken: true,

    };

    let url =

      "https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=AIzaSyCcbQpkG41AVojBdarHWCS\_Uy9nfWyi2Pg";

    if (!isSignup) {

      url =

        "https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyCcbQpkG41AVojBdarHWCS\_Uy9nfWyi2Pg";

    }

    const { data } = await axios.post(url, authData);

    const expirationDate = new Date(

      new Date().getTime() + data.expiresIn \* 1000

    );

    localStorage.setItem("token", data.idToken);

    localStorage.setItem("expirationDate", expirationDate);

    localStorage.setItem("userId", data.localId);

    dispatch(authSuccess(data.idToken, data.localId));

    dispatch(checkAuthTimeOut(data.expiresIn \* 1000));

  } catch (error) {

    dispatch(authFail(error.response.data.error.message));

  }

};

export const setAuthRedirectPath = (path) => {

  return {

    type: actionTypes.SET\_AUTH\_REDIRECT\_PATH,

    path: path,

  };

};

export const authCheckState = () => (dispatch) => {

  const token = localStorage.getItem("token");

  if (!token) {

    dispatch(loggout());

  } else {

    const expirationDate = new Date(localStorage.getItem("expirationDate"));

    if (expirationDate > new Date()) {

      const userId = localStorage.getItem("userId");

      dispatch(authSuccess(token, userId));

      dispatch(

        checkAuthTimeOut(expirationDate.getTime() - new Date().getTime())

      );

    } else {

      dispatch(loggout());

    }

  }

};

Auth reducer

import \* as actionTypes from "../actions/actionTypes";

import { updateObject } from "../utility";

const initialState = {

  token: null,

  userId: null,

  error: null,

  loading: false,

  authRedirectPath: "/",

};

const authStart = (state) => {

  const updateState = {

    loading: true,

    error: null,

  };

  return updateObject(state, updateState);

};

const authSucces = (state, action) => {

  const updateState = {

    token: action.idToken,

    userId: action.localId,

    loading: false,

    error: null,

  };

  return updateObject(state, updateState);

};

const authFail = (state, action) => {

  const updateState = {

    loading: false,

    error: action.error,

  };

  return updateObject(state, updateState);

};

const authLoggout = (state, action) => {

  const updateState = {

    userId: null,

    token: null,

  };

  return updateObject(state, updateState);

};

const setRedirectPath = (state, action) => {

  return updateObject(state, { authRedirectPath: action.path });

};

const reducer = (state = initialState, action) => {

  switch (action.type) {

    case actionTypes.AUTH\_START:

      return authStart(state);

    case actionTypes.AUTH\_SUCCESS:

      return authSucces(state, action);

    case actionTypes.AUTH\_FAIL:

      return authFail(state, action);

    case actionTypes.AUTH\_LOGGOUT:

      return authLoggout(state, action);

    case actionTypes.SET\_AUTH\_REDIRECT\_PATH:

      return setRedirectPath(state, action);

    default:

      return state;

  }

};

export default reducer;

React Animation

1)Using css transition and animation

2)npm I react-transition-group

import React from "react";

import { Transition } from "react-transition-group";

import "./Modal.css";

const animationTiming = {

  enter: 400,

  exit: 600,

};

const modal = (props) => {

  return (

    <Transition

      in={props.show}

      timeout={animationTiming}

      unmountOnExit

      mountOnEnter

    >

      {(state) => {

        const cssClasses = [

          "Modal",

          state === "entering"

            ? "ModalOpen"

            : state === "exiting"

            ? "ModalClosed"

            : null,

        ];

        return (

          <div className={cssClasses.join(" ")}>

            <h1>A Modal</h1>

            <button className='Button' onClick={props.closed}>

              Dismiss

            </button>

          </div>

        );

      }}

    </Transition>

  );

};

export default modal;

in = {props.show} to show and hide

timing = timing for different state enter,exit,

Transition Events

 <Transition

          in={this.state.showBlock}

          timeout={1000}

          mountOnEnter

          unmountOnExit

          onEnter={() => console.log("onEnter")}

          onEntering={() => console.log("onEntering")}

          onEntered={() => console.log("onEntered")}

          onExit={() => console.log("onExit")}

          onExiting={() => console.log("onExiting")}

          onExited={() => console.log("onExited")}

        >

          {(state) => (

            <div

              style={{

                background: "red",

                width: "100px",

                height: "100px",

                margin: "auto",

                transition: "opacity 1s ease-in-out",

                opacity: state == "exiting" ? 0 : 1,

              }}

            ></div>

          )}

        </Transition>

CssTransition

import React from "react";

import { CSSTransition } from "react-transition-group";

import "./Modal.css";

const animationTiming = {

  enter: 400,

  exit: 600,

};

const modal = (props) => {

  return (

    <CSSTransition

      in={props.show}

      timeout={animationTiming}

      unmountOnExit

      mountOnEnter

      classNames='fade-slide'

    >

      <div className='Modal'>

        <h1>A Modal</h1>

        <button className='Button' onClick={props.closed}>

          Dismiss

        </button>

      </div>

    </CSSTransition>

  );

};

export default modal;

.fade-slide-enter {

}

.fade-slide-enter-active {

  animation: openModal 0.4s ease-out forwards;

}

.fade-slide-exit {

}

.fade-slide-exit-active {

  animation: closeModal 0.6s ease-out forwards;

}

Customizing ClassNames

import React from "react";

import { CSSTransition } from "react-transition-group";

import "./Modal.css";

const animationTiming = {

  enter: 400,

  exit: 600,

};

const modal = (props) => {

  return (

    <CSSTransition

      in={props.show}

      timeout={animationTiming}

      unmountOnExit

      mountOnEnter

      classNames={{

        enterActive: "ModalOpen",

        exitActive: "ModalClosed",

      }}

    >

      <div className='Modal'>

        <h1>A Modal</h1>

        <button className='Button' onClick={props.closed}>

          Dismiss

        </button>

      </div>

    </CSSTransition>

  );

};

export default modal;

<https://reactcommunity.org/react-transition-group/>