**REACT-ROUTER**

2 type Routing vardir. Server side ve client side routingler.

Server Side Rote =

+Geleneksel routing dir.

+Server’a bir request yapariz ve sunucunun durumuna gore hizli veya yavas geri donus response alarak gerceklesen routing islemleridir.

Client Side Route = Modern Routeing dir. Dinamik degisimleri saglar Angular React gibi populer librarylerde.

+Hizlidir.

+Sayfaya birkere girdikten sonra yapabilecegimiz istekleri ayarladiktan sonra tum istekleri hemen aninda cevap verir. Cunku daha onceden ayarlanmistir.

//Ilk basta bu ikisini HTML sayfasina import ediyorduk ama simdi direk dosyaya import ediyoruz. Bu yuzden daha sonra bunlari dosyalarimiza ekledik. Ilk basta 2 sininde CDN kodu index.html de idi

import React from 'react';

import ReactDOM from 'react-dom';

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

import 'normalize.css/normalize.css';

import './styles/styles.scss'

const ExpenseDashboardPage = () => (

    <div>

        This is from my dashboard components

    </div>

)

const AddExpensePage = () => (

    <div>

        This is from my expense components

    </div>

)

const EditExpensePage = () => (

    <div>

        This is from my Edit Expense component

    </div>

)

const HelpPage = () => (

    <div>

        This is from my Help Component

    </div>

)

//Link Client side route s ler yapmamizi saglar bu sayede full page refesh ler yapmadan route islemlerimizi yapabiliriz.

//to ile nereye hangi route'a yonlendirecegini belirleriz.

const NotFoundPage = () => (

    <div>

        404 - <Link to="/">Go Home</Link>

    </div>

)

//NavLink ' ler ekstra bir kac ozellik daha alabilirler Linklere gore

//Ekstra active class I alir navlinkler active path hangisiyse buda bize onlara yeni ozzellikler verebilmemizi saglar.

//Navbar lar icin en uygun route seklidir.

//activeClassName sadece biz o sayfadayken uygulanacak bir class alir. Yani icinde bir class tanimlariz kendimizin belirledigi bir adda ve ona style.scss icinde kalin olsun dersek. Sadece gittigimzi route un sayfasinda baslik kalin olur digerleri ince olur. Baska bir sayfaya gidersek az once kalin olan sayfa ince olur diger gittigimiz sayfa kalin olur.

//exact true niye? Cunku / oldugu icin gittigi tum route larda isActive in ozelligini bir gittigimiz route birde / route u aliyordu. exact true yaparak sadece / noktasina gidildiginde ozelligi almasini sagladik

//Veya activeStyle ile style verebiliriz active olanlara.

const Header = () => (

    <header>

        <h1>Expensify App</h1>

        <NavLink to="/" activeClassName="isActive" exact={true}>Home Page</NavLink>

        <NavLink to="/creat" activeClassName="isActive">Create Expense</NavLink>

        <NavLink to="/edit" activeClassName="isActive">Edit</NavLink>

        <NavLink to="/help" activeClassName="isActive">Help</NavLink>

    </header>

)

//BrowserRouter => indivudual route dur. Sadece 1 tane gereklidir bundan.

//Route => instance dir. route 2 props alir. path=> Hangi url yi kullanmak istedigimizi. component=> path deki url ile eslestigimizde ne gormek istedigimizi belirleriz.

//exact={true} 'yu yazmadan once /create 'e gidincede ExpenseDashboardPage sayfamiz gozukuyordu fakat artik ust route lerde gozukmez bu kod sayesinde. Sadece / ana path e yazmamiz ustekiler icinde yeterli olacaktir.

//<Route component={NotFoundPage} /> => Bu sekilde yaptiktan sonra tum linkler notFound page ye yada component yazdirilacaktir. Burada bir sorun ortaya cikiyor 404 u diger route larimizdan sonra yani altina da yazdiriyor olacak. Bunu onlemek icin Switch yapisini kullanacagiz. Bu yuzden Route 'larimizi switch icine yazdiracagiz cunku SWitch eslesme yaptiktan sonra duruyor ve baska eslesme (404 route u gibi) olmasina msuade etmiyor.

const routes = (

    <BrowserRouter>

        <div>

            <Header />

            <Switch>

                <Route path="/" component={ExpenseDashboardPage} exact={true} />

                <Route path="/creat" component={AddExpensePage} />

                <Route path="/edit" component={EditExpensePage} />

                <Route path="/help" component={HelpPage} />

                <Route component={NotFoundPage} />

            </Switch>

        </div>

    </BrowserRouter>

)

ReactDOM.render( routes, document.getElementById('app'));

Yada asagidaki gibi baska bir yontem

<Route path="/" exact render={() => <Posts />} />

İcinde h1 yazsa onu render eder

<Route render={() => <Posts />} />

Eger path vermezsek tanimlanmamis tum route lari yakalar ve bu kodu 404 sayfasi olarak kullanabiliriz.

Bu Component’i

const routes = (

    <BrowserRouter>

        <Switch>

            <Route path="/" component={ExpenseDashboardPage} exact={true} />

            <Route path="/creat" component={AddExpensePage} />

            <Route path="/edit" component={EditExpensePage} />

            <Route path="/help" component={HelpPage} />

            <Route component={NotFoundPage} />

        </Switch>

    </BrowserRouter>

)

Eger alttaki sekile getirirsek her sayfada Header Componentini en ustte gormus olacagiz.

Switchin ustundeki hersey her sayfada gozukur.

const routes = (

    <BrowserRouter>

        <div>

            <Header />

            <Switch>

                <Route path="/" component={ExpenseDashboardPage} exact={true} />

                <Route path="/creat" component={AddExpensePage} />

                <Route path="/edit" component={EditExpensePage} />

                <Route path="/help" component={HelpPage} />

                <Route component={NotFoundPage} />

            </Switch>

        </div>

    </BrowserRouter>

)

//Link Client side route s ler yapmamizi saglar bu sayede full page refesh ler yapmadan route islemlerimizi yapabiliriz.

//to ile nereye hangi route'a yonlendirecegini belirleriz.

const NotFoundPage = () => (

    <div>

        404 - <Link to="/">Go Home</Link>

    </div>

)

const Header = () => (

    <header>

        <h1>Expensify App</h1>

        <Link to="/">Home Page</Link>

        <Link to="/creat">Create Expense</Link>

        <Link to="/edit">Edit</Link>

        <Link to="/help">Help</Link>

    </header>

)

Gelismis Link kurulumu

<Link

to={{

pathname: "/new-post",

hast: "#submit", //Atlamalar yapmamiza yarar

search: "?quick-submit=true" //Query olusturmamiza yarar

}}>

New Post

</Link>

**WITHROUTER - HISTORY LOCATION MATCH ‘e ERISMEK**

withRouter ile erismek istedigimiz componenti asagidaki gibi export etsek yeterli oluyor.

export default withRouter(post);

**ABSOLUTE AND RELATIVE PATH**

Absolute Path => Sadece gidilecek yol olur.

pathname: "/new-post"

Relative Path => Tum link olur. Mucahidyazar.io/new-post bir absolute path dir.

pathname: this.props.match.url + "/new-post"

**SAYFA YENILEMEDEN PUSH ETMEK YONLENDIRMEK**

this.props.history.push(‘/’) ile sayfa yenilemeden yonlendirmeler yapabiliriz.

This.props.history.push(pathname: ‘/’ + id)

**404 NOT FOUND SAYFASI TAKTIGI**

<Route render={() => <Posts />} />

Eger path vermezsek tanimlanmamis tum route lari yakalar ve bu kodu 404 sayfasi olarak kullanabiliriz.

Ve Route tanimlamalarinda en alta yazin.

Redirect ile ayni hizadayken calismaz

**REDIRECT**

Tabi once react-router-dom dan import ediyoruz

<Redirect from="/" to="/posts" />

**REDIRECT --- CONDITIONAL --- KOSULLU**

Herhangi bir componentin icine bunu yazarsak, bu komponent render edildiginde bu sayfaya redirect et anlamina gelir.

<Redirect to="/posts" />

**ORNEK**

import React, { Component } from "react";

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

…

class NewPost extends Component {

  state = {

    …

  };

  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.data);

        this.setState({ submitted: true });

//Yada tum redirectler yerine sadece bu alttaki push satirini ekleriz.

this.props.history.push("/");

      })

      .catch(err => {

        console.error(err);

      });

  };

  render() {

    return (

      <div className="NewPost">

        {this.state.submitted ? <Redirect to="/posts" /> : null}

        <h1>Add a Post</h1>

        …

        <button onClick={() => this.postDataHandler()}>Add Post</button>

      </div> )}}

export default NewPost;

**HISTORY PUSH**

Yada tum redirectler yerine sadece bu alttaki push satirini, axiosda then icine ekleriz.

this.props.history.push("/");

**LAZILY LOADING** – SADECE TIKLADIGINDA SAYFANIN INDIRILMESI VE BOYLELIKLE DAHA AZ ISLEM DAHA AZ BOYUT VE DAHA FAZLA HIZ KAZANMAMIZ

* Once asagidaki gibi bir AsyncComponent.js adinda hoc yani higher order component olusturuyoruz.
* Sonra burada asagidaki veya benzeri bir sistemi yaziyoruz.

import React, { Component } from "react";

const AsyncComponent = importComponent =>

  class extends Component {

    state = {

      component: null

    };

    componentDidMount() {

      importComponent().then(cmp => {

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

      });

    }

    render() {

      const C = this.state.component;

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

    }

  };

export default AsyncComponent;

Daha sorna bu komponenti asagidaki sekilde import edilmesi uzere hazir birakmak icin yeni bir degisken tanimlayarak fonksiyon olustturuyoruz hangi komponenti lazily yapacaksak. Daha sonra laziliy olacak componentin koyulacagi yere bu tanimladigimiz fonksiyon degiskeni yerlestiriyoruz en alttaki gibi

import AsyncComponent from "../../hoc/AsyncComponent";

const AsyncNewPost = AsyncComponent(() => {

  return import("./NewPost/NewPost");

});

Daha sonra laziliy olacak componentin koyulacagi yere bu tanimladigimiz fonksiyon degiskeni yerlestiriyoruz en alttaki gibi

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

**NEW LAZY LOADING**

Sayfamiz acildiginda herseyin dowloand edilmesini istemeyiz buyuk uygulamalarda cunku uygulamalarimizi yavaslatir. Lazyloading bu yavaslamalarin onune gecmek icin sayfa yuklediginde component icindeki componentlari tiklayana kadar indirmez. React 16..6 ile super pratik bir lazy loading kullanimi gelmistir. Yukaridakine gore cok basittir.

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

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

import User from './containers/User';

import Welcome from './containers/Welcome';

**//Once React.lazy yazarak lazy mizi cekiyoruz ve bir degiskene tanimliyoruz. Daha sonra icine bir fonksiyon yaziyoruz ve import edecegimiz componentimizi seciyoruz.**

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

class App extends Component {

**### 1. YOL ###**

**state = { showPosts: false };**

**modeHandler = () => {**

**this.setState(prevState => {**

**return { showPosts: !prevState.showPosts };**

**});**

**};**

**render() {**

**return (**

**<React.Fragment>**

**<button onClick={this.modeHandler}>Toggle Mode</button>**

**{this.state.showPosts**

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

**<Posts />**

**</Suspense>**

**:**

**<User />**

**}**

**</React.Fragment>**

### 2. YOL ###

**<BrowserRouter>**

**<React.Fragment>**

**…**

**<Route**

**path="/posts"**

**render={() => (**

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

**<Posts />**

**</Suspense>**

**)}**

**/>**

**</React.Fragment>**

**</BrowserRouter> )}}**

export default App;

fallback => Tikladigimizda yuklenene kadar gosterilecek yazi veya component. Isteginiz herseyi koyabilirsiniz.

Cagrilacak Component Suspense icine alinir.

SET BASE PATH

class App extends Component {

  render() {

    return (

      <BrowserRouter basename="/my-app">

        <div className="App">

          <Blog />

        </div>

      </BrowserRouter>

    );

  }

}

Bir onceki sayfaya yonlendirir

  onCheckouyCancelled = () => {

    this.props.history.goBack();

  };

Olunan guncel sayfanin url sini istediginizle degistirerek yonlendirir.

  onCheckouyContinued = () => {

    this.props.history.replace("checkout/contact-data");

  };

QUERYS

state = {

  ingredients: {

   salad: 0.5,

   cheese: 0.4,

   meat: 1.3,

   bacon: 0.7

};

};

Burada i ingredientsdeki her bir key dir yani salad cheese meat bacon gibi.

    const queryParams = [];

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

      queryParams.push(

        encodeURIComponent(i) + "=" + encodeURIComponent(this.state.ingredients[i])

      );

    }

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

    this.props.history.push({

      pathname: "/checkout",

      search: "?" + queryString

    });

URL deki Query degerlerinden this.setState i degistirmek

* componentDidMountla yapmamizin nedeni urli her degistirip enter yaptigimizda sayfa yeniden yuklenerek acilir.
* İlk satirda bir degiskene urldeki querylerimizi secip koyuyoruz
* Sonra bir ingredients degikeni tanimliyoruz daha sonra this.setState state icindeki ingrediente ayarlamak icin
* For ile ingredients i tekrar bir obje haline getiriyoruz key ve value lerle birlikte
* Sonra setState

  componentDidMount() {

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

    const ingredients = {};

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

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

    }

    this.setState({ ingredients });

  }

Parsing Query Parameters & the Fragment

You learned how to extract route parameters (=> :id  etc).

But how do you extract **search** (also referred to as "**query**") **parameters**(=> ?something=somevalue  at the end of the URL)? How do you extract the **fragment** (=> #something  at the end of the URL)?

**Query Params:**

You can pass them easily like this:

<Link to="/my-path?start=5">Go to Start</Link>

or

1. <Link
2. to={‌{
3. pathname: '/my-path',
4. search: '?start=5'
5. }}
6. >Go to Start</Link>

React router makes it easy to get access to the search string: props.location.search .

But that will only give you something like ?start=5

You probably want to get the key-value pair, without the ?  and the = . Here's a snippet which allows you to easily extract that information:

1. componentDidMount() {
2. const query = new URLSearchParams(this.props.location.search);
3. for (let param of query.entries()) {
4. console.log(param); // yields ['start', '5']
5. }
6. }

URLSearchParams  is a built-in object, shipping with vanilla JavaScript. It returns an object, which exposes the entries()  method. entries()  returns an Iterator - basically a construct which can be used in a for...of...  loop (as shown above).

When looping through query.entries() , you get **arrays** where the first element is the **key name** (e.g. start ) and the second element is the assigned **value** (e.g. 5 ).

**Fragment:**

You can pass it easily like this:

<Link to="/my-path#start-position">Go to Start</Link>

or

1. <Link
2. to={‌{
3. pathname: '/my-path',
4. hash: 'start-position'
5. }}
6. >Go to Start</Link>

React router makes it easy to extract the fragment. You can simply access props.location.hash .