# REACTJS (NPM – VSCode)

Single Page Application (SPA)

Facebook tarafından geliştirilmiş bir javascript web framework’üdür.

Index.html (Projenin tek web sayfası)

Index.js (Projenin script konfigürasyon dosyası)

React projesini oluşturmak için (proje isimlerinde büyük harf kullanmıyoruz.)

>npx create-react-app intro (yoksa, npm install create-react-app ile kuruyoruz.)

Çalıştırmak için,

>npm start

# JSX (Javascript XML) ( { … } )

Jsx elementleri mutlaka kapsayıcı bir etikete (tag) sahip olmalıdır. Bunun için,

<React.Fragment>…</React.Fragment> (sayfaya react yüklenmelidir (import))

veya <> … </> kullanılabilir.

Css tanımlarını etiketlere class yerine className ile ekliyoruz.

Eğer inline css (style) eklemek istersek { {object} } şeklinde tanımlıyoruz.

<div style={{ width:’200px’, height:’200px’, backgrounColor: ‘blue’ }} ></div>

background-color şeklindeki css’ler birleştirilerek camelCase yapısında tanımlanıyor.

# Components

React component mimarisiyle çalışır. Önce componentler yüklenir sonra render işlemi yapılır.

Component isimlerini büyük harfle başlatıyoruz.

Aynı seviyedeki componentler arasında veri transferi yapılamıyor.

App.js parent component, Rfcxxxx.js ve Rccxxxx.js child component’ler olarak kabul edilir. Veri transferi parent component’lerden child component’lere doğru yapılır.

# Props

Parent componentten (App.js) Child componente (Card.js) veri aktarabiliyoruz. Child component bu verileri props olarak alıp kullanabiliyor.

React içinde aynı seviyedeki componentler arasında veri taşıyamıyoruz.

# State

Aynı component içinde kullanmak istediğimiz verileri state ile tanımlıyor ve değişen değerlerinin geçerli olmasını istiyoruz.

Class component mimarisinde constructor içinde tanımlanıp, gerektiğinde setState fonkisyonuyla değeri değiştiriliyordu.

Şimdi fonksiyonel componentlerde useState hooks yapısı kullanılarak state değerlerini değiştiriyoruz.

const [first, setFirst] = useState(second)

first, içinde state değerini tutacağımız değişken.

Second, state değişkeninin başlangıç değeri.

setFirst, state değişkeninin değerini değiştirecek fonksiyon.

Reactstrap

>npm i reactstrap

>npm i bootstrap

# React-Router-Dom

Öncelikle react-router-dom paketini yüklüyoruz.

>npm i react-router-dom

Ardından index.js sayfasında BrowserRouter yükleyerek App.js’i içine alıyoruz.

  <BrowserRouter>

    <App />

  </BrowserRouter>

App.js sayfasında Routes ve Route componentlerini yükleyerek route tanımlarımızı routes içinde yapıyoruz.

        <Routes>

          <Route path="/" element={<Home />} />

          <Route path="/about" element={<About />} />

          <Route path="/contact" element={<Contact />} />

        </Routes>

Eğer projemizde link vermek istersek sayfamıza Link yükledikten sonra,

        <ul>

          <li><Link to="/">Home</Link></li>

          <li><Link to="/about">About</Link></li>

          <li><Link to="/contact">Contact</Link></li>

        </ul>

# Json-server

Projelerimizde test amaçlı fake restful apilere ihtiyaç duyuyoruz. Bu durumda kullanabileceğimiz pekçok fake api örneği bulabiliriz.

<https://jsonplaceholder.typicode.com>,

https://reqres.in/api/users

Ayrıca json-server kullanarak kendi ihtiyaçlarımıza göre şekillendirebileceğimiz bir db.json fake api kullanabiliriz.

Bunun için projemize json-server kuruyoruz (istersek global olarak kurabiliriz).

>npm i -g json-server

Proje root klasörümüzde bir api klasörü oluşturup, içine db.json ekliyoruz.

Aktif hale getirmek için,

>json-server --watch api/db.json --port 3002

Datalara ulaşmak için;

<http://localhost:3002/posts> GET method ile tüm postları çekiyoruz.

<http://localhost:3002/posts/2> GET method ile id=2 olan postun bilgilerini çekiyoruz.

<http://localhost:3002/posts> POST method ile body üzerinde tanımladığımız yeni post bilgilerini db.json’a kayıt ediyoruz. id otomatik olarak json-server tarafından veriliyor.

const getPostsWithFetch = async () => {

  const response = await fetch('http://localhost:3002/posts')

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

}

Yeni kayıt eklemek istersek;

const requestOptions = {

method: “POST”,

headers: { ‘Content-Type’: ‘application-json’ },

body: JSON.stringify({ id:1, title:’React’, author:’ali uçar’ })

};

await fetch(‘http://localhost:3002/posts’, requestOptions)

# AXIOS

Öncelikle projemize yüklüyoruz.

>npm i axios

Ardından sayfamıza tanıtıyoruz.

import axios from ‘axios’

Tüm dataları çekmek için;

const [posts, setPosts] = useState([])

const getPostsWithAxios = async () => {

  const response = await axios.get('http://localhost:3002/posts')

  console.log(response.data);

  setPosts(response.data)

}

Kayıt ekleme, güncelleme ve silme işlemlerinde post, put ve delete metodlarını kullanıyoruz.

const deletePostWithAxios = async (id) => {

  id=5;

  const response = await axios.delete(`http://localhost:3002/posts/${id}`)

  console.log(response.data);

}

# React Yaşam Döngüsü (life cycle)

React 16.x.x.x versiyona kadar genelde class component mimarisiyle çalışıyordu ve yaşam döngüsünü componentDidMount, xxxWillMount, xxxUnMount gibi metodlarla kontrol ediyorduk.

16.x.x.x versiyondan itibaren Hooks paketinin fonksiyonel componentleri desteklemesiyle, react yaşam döngüsünü useEffect() hook metodlarıyla kontrol ediyoruz.

  useEffect(() => {

    console.log('her zaman çalışır.');

    //alert('değişiklik oldu') her zaman çalışır.

  })

  useEffect(() => {

    console.log('sadece component ilk yüklendiğinde çalışır.');

    setTek(5); //sadece component yüklendiğinde tek değerini 5 yapar.

  }, [])

  useEffect(() => {

    console.log("component ilk yüklendiğinde ve tek isimli state her değiştiğinde çalışır.");

  }, [tek])

# Context API – useContext

React projelerinde state’ler prop’lar yardımıyla parent (üst) component’lerden child (alt) componentlere aktarılır. Projeler büyüdükçe props drilling ismi verilen sorunla karşılaşılır ve projenin yönetilmesi zorlaşır.

Context-api ile componentler arasında istediğimiz data, method gibi verileri kolaylıkla taşıyabiliyoruz.

State’leri componentler arasında paylaşabilmek için öncelikle bir context oluşturuyoruz.

Bu işlem için React altında createContext kullanıyoruz.

Oluşturduğumuz context’i kullanmak için Context.Provider ve Context.Consumer olmak üzere iki adet componente ihtiyacımız var.

## Axios – Jwt Bearer Token

tokenPayload() {

let config = {

headers: {

'Authorization': 'Bearer ' + validToken()

}

}

Axios.post(

'http://localhost:8000/api/v1/get\_token\_payloads',

config

)

.then( ( response ) => {

console.log( response )

} )

.catch()

}