1. Ilk once Babel-cli, live-server, @babel/preset-react, @babel/preset-env ‘i yuklemeliyiz projemize.

@babel/preset-react

@babel/preset-env

+Bu ikisi kodlarimizi tarayicilarin anlayacagi sekilde ayarlayacak dil paketleridir.

Babel-cli

+babel-cli ise bu iki ek yamayi kontrol ederek donusturmeleri yonetir.

+commmand’da yani terminalde babel kodlarimizi calistirmaya yarar boylelikle onlari belirledigimiz yerdeki dosyalarin icine gomeriz.

live-server

+Hazirladigimiz projeyi local’de izlememize gormemize yarayan modul’dur.

1. Daha sonra package.json doyasina asagidaki kodu script olarak tanimlariz.

babel src/app.js --out-file=public/scripts/app.js --presets=env,react –watch

**--watch =>** babel in surekli src/app.js i denetlemesini soyler ve degisiklikleri aninda guncellemesini saglar.

**--out-file =>** babel’in donusturdugu dosya icerigini nerede hangi isimde bir dosyaya cikarttigi yoldur.

**--presets=env,react =>** Bu kodda babel’in kullanacagi paketleri belirler. Tabi bunlari yuklemeyi unutmayin. (@babel/preset-react, @babel/preset-env) Bu iki paket bu yuzden onemli.

Bu kodla jsx kodlu dosyamizi tarayicilarin anlayacagi dile ceviririz otomatik olarak. Ve kodu calistirdigimizda otomatik olarak takip edip dosyamizi kontrol etmeye baslar.

1. Ve bu adimlardan sonra asagidaki gibi 2 tane script’i tanimliyoruz package.json icinde.

"serve": "live-server public/",

"build-babel": "babel src/app.js --out-file=public/scripts/app.js --presets=env,react --watch",

**NOT**

console.log('app.js is running');

// JSX - Javascript XML

//render edilem template de sadece 1 tane root elemani yani html tagi bulunabilir. Tum tagleri bir tag icinde toplamaliyiz bunun icin

var template = (

        <div>

            <h1>This is JSX from app.js</h1>

            <p>This is some info</p>

            <ol>

                <li>Item 1</li>

                <li>Item 2</li>

                <li>Item 3</li>

            </ol>

        </div>

    );

var appRoot = document.getElementById('app');

//ReactDOM.render ilk parametre olarak eklenecek degeri alir

//ikinci parametre olarak degistirilecek yolu alir.

//Degistirilecek yol yukaridaki appRoot ornegindeki gibi secilir.

ReactDOM.render(template, appRoot);

**FARKLI KULKLANIMLAR**

function getLocation (location) {

    if(location){

        return <p>Location : {location}</p>;

    } else {

        return undefined

    }

}

//Asagidaki sekillerdede kullanimi mevcuttur.

var templateTwo = (

    <div>

//TERNARY OPERATORS

//Burada username eger varsa bu true donecek

//Eger true donerce ilk deger yani user.name yazdirilacak

//Eger false donerse : ‘den sonra yazdigimiz ikinci deger ‘Anonymouse’ yazdirilacak

        <h2>{user.name ? user.name : 'Anonymouse'}</h2>

//LOGICAL OPERATORS

//JSX dosyalarinda {false} {null} {undefined} yazarsak template div icerisinde hicbir goruntu almayiz.

//Agegida && isareti su isimize yariyor. Eger && isaretinden once bir deger yazarsak ve bu dogruysa, && isaretinden sonraki degeri yazdirir. Eger degilse {false} donderir ve buda dogru olmadigi icin hicbirsey gozukmez.

        { (user.age && user.age >= 18) && <p>Age : {user.age}</p> }

        <p>{getLocation(user.location)}</p>

        {getLocation(user.location)}

    </div>

)

**NOT**

//Javascriptte

//!'' true demektir

//!'Mucahid' false demektir

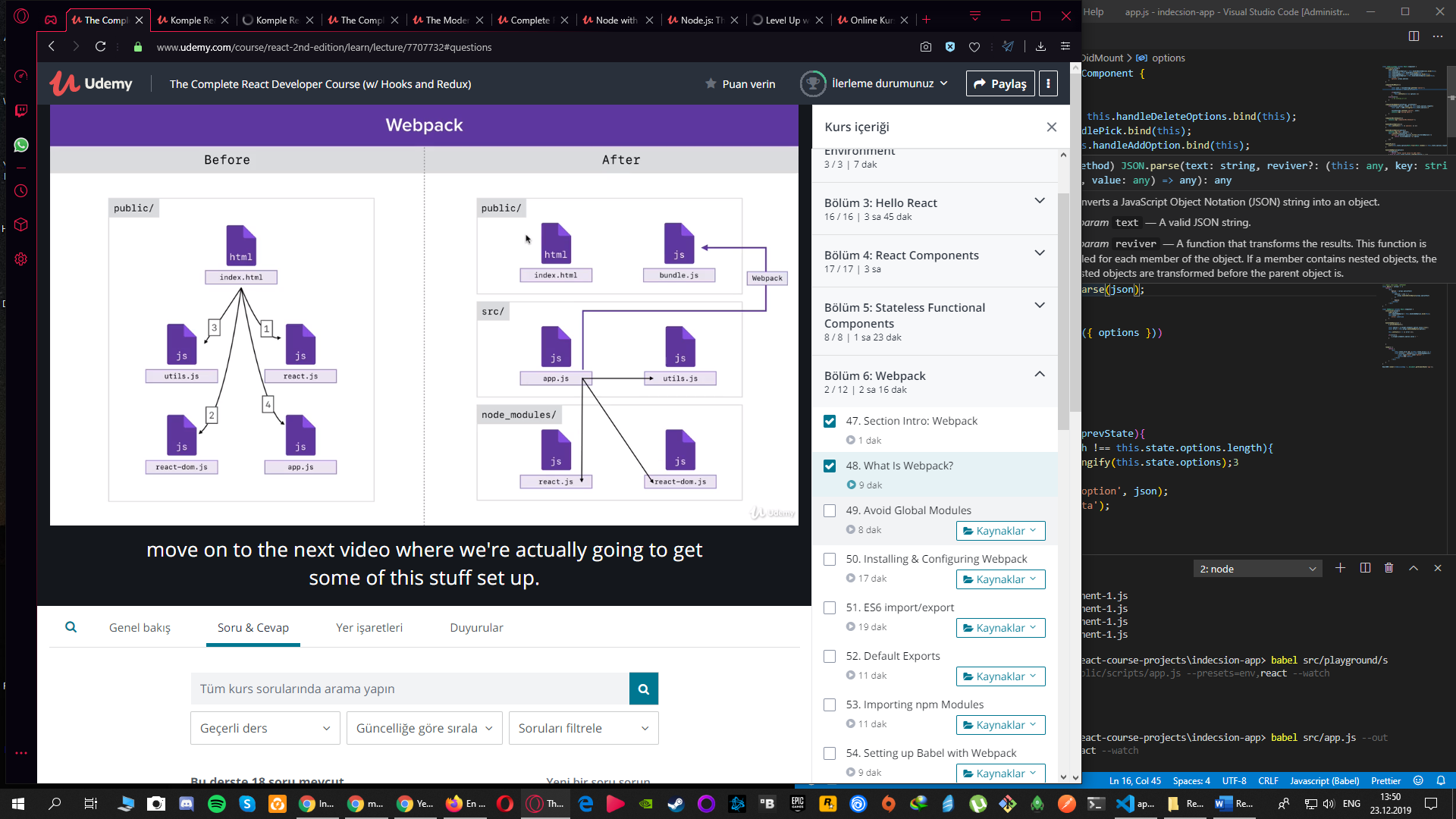
//!!'' false demektir

//!!'Mucahid' true demektir

//!undefined true demektir

//!!undefined false demektir

**WEBPACK CALISMA MANTIGI / ONCESI VE SONRASI**



**Webpack**

+Butun dosyalari tek dosyaya baglayip onu indexe baglamamiza yarayacak

+daha hizli olacak projeler

+daha duzenli tutulacak

Dosyalar saglar bize.

1. Once webpack ’i yukluyoruz projemize. Ve yukledikten sorna asagidaki sekildeki gibi dahil edip scriptlere ekliyoruz.

**--watch =>** webpack ’in surekli denetleme yapmasini soyler ve degisiklikleri aninda guncellemesini saglar.

"serve": "live-server public/",

"build": "webpack --watch",

"build-babel": "babel src/app.js --out-file=public/scripts/app.js --presets=env,react --watch",

**NOT**

Webpack ‘i yukledikten sonra build’deki babel komutunu build-babel yapiyoruz. (Artik ihtiyacimiz kalmadi aslinda isterseniz silebilirsinizde.) Ve build scripti olarak webpack ‘i yukaridaki gibi tanimliyoruz.

Webpack ’i yukleyip Config dosyasini ayarladiktan sonra artik package.json dosyasindaki babel-build scriptine ihtiyac kalmiyor webpack bunu bizim icin zaten yapiyor olacak

**WEBPACK CONFIG DOSYASI**

1. Webpack’i yukledikten sonra webpack.config.js dosyasini olusturuyoruz.

Kullanmak icin node ile birlikte gelen path modulunu REQUIRE yapiyoruz asagidaki gibi.

const path = require('path');

+Ve module.exports yaparak ayarlarimizi icinde EXPORT ederek ayarliyoruz.

module.exports = {

//Burasi ya development olur yada production olur. Gelistirme asamasinda development yayinlama asamasinda productiondur.

    mode: 'development',

    entry: './src/app.js',

    //outputta 2 sey ayarlamamiz tanimlamamiz gerekiyor. Path ve filename

    output: {

        //(./public) bu sekilde tanimlayamayiz burada onun icin asagidaki sekilde tanimlkiyoruz.

        //Bu path yoluna webpack bizim icin tum projeyi bundle.js adinda tek bir dosyada toplayacak.

        path: path.join(\_\_dirname, 'public'),

        //filename istedigimiz herhangi birsey olabilir. Genelde bundle.js kullaniliyor onun icin bende onu kullanacagim

        filename: 'bundle.js'

    },

**IMPORT and EXPORT**

**USER.JS**

//EXPORT ve IMPORT ORNEKLERI

//import './utils'; // CIKTI => 'utils.js is running'

//Bu sekilde import ile sadece dosyayi calistiririz icindekilere ulasamayiz.

//Icindekilere ulasmamiz icin isimlendirilmis yani NAMED EXPORT'lari isimleriyle import etmek gerekmektedir.

//2. 3. 4. gibi daha fazla IMPORT'larda da EXPORT'ta oldugu gibi virgul ve sonrasinda eklenir.

import { square, add, minus } from './utils';

console.log(square(5)); // CIKTI => 25

//CHALLENGE

//DEFAULT EXPORT'lar {} suslu parantezler disinda cagirilirlar ASAGIDAKI isSenior gibi

import isSenior, { isAdult, canDrink } from './utils';

console.log(isAdult(45)); // CIKTI => true

console.log(canDrink(30)); // CIKTI => true

console.log(isSenior(25)); // CIKTI => true

//DEFAULT EXPORT'lar yada asagidaki gibi de IMPORT edilebilir.

import isSenior from './utils';

//Veya dEFAULT EXPORT yerine istedigimiz her hangi bir isimde kullanabiliriz asagidaki gibi. Cunku program biliyor DEFAULT EXPORT hangisi sadece 1 TANE oldugu icin.

import istedigimHerHangiBirSey from './utils';

**IMPORT and EXPORT**

**UTILS.JS**

console.log('utils.js is running');

//Buna user.js'den ulasilmasi icin once asagidaki gibi EXPORT etmemiz gerekiyor.

const square = (x) => x \* x;

// 2 Type EXPORT turu vardir

// DEFAULT EXPORT ve NAMED EXPORT

//Ikinci bir export icin virgul koyulup yanina gondermek istedigimiz degiskeni gonderebiliriz. Asagida 2. olarak bir fonksiyon gonderiyoruz ornek olarak. 3. 4. gibi islemlerdede aynisi gecerli.

export  { square, add, minus };

const add = (a, b) => a + b;

//Ayrica basina EXPORT yazarak da export edebiliriz degiskenleri.

export const minus = (a, b) => a - b;

//Tabi STRING'ler buna dahil degil

//export 'YANLIS KULLANIM'

//Degiskenlerin baslarina export yazarakda EXPORT islemini yapabiliriz.

//Fakat export default degiskenlerin basina yazilmaz. FAKAT CLASS'larin basina yazilabilir.

export const sayi1 = 5; //DOGRU

export const topla = (a, b) => a + b; //DOGRU

export default sayi2 = 7; //YANLIS

export default sayi2; //DOGRU => sayi2 once bir degisken olarak tanimlanir sonra bu sekilde EXPORT edilebilir.

export default topla = (a, b) => a + b; //YANLIS

export default topla; //DOGRU => topla once bir degisken olarak tanimlanir sonra bu sekilde EXPORT edilebilir.

export default class topla { } //DOGRU

//CHALLENGE

const isAdult = (age) => age>=18

const canDrink = (age) => age>=18

const isSenior = (age) => age>=65

export {isAdult, canDrink, isSenior as default};

//DEFAULT EXPORT'lar 1 tane olabilir sadece bir dosyada.

//Ya yukaridaki gibi as default yanina yazilarak normal EXPORT'larla birlikte EXPORT edilebilirler.

**REACT and REACT-DOM**

+React uygulamalari gelistirmemiz ve web sayfalarimizda yayinlamamiz icin kullanmamiz gereken package yani module lerdir. Ister CDN lerle istersek projemize dahil ederek yukleyerek kullanabiliriz.

+HTML sayfasinda kullandigimiz REACT ve REACT-DOM CDN lerini kullanmamak icin bu 2 modulu once yukluyoruz ve sonra asagidaki gibi IMPORT ediyoruz her REACT COMPONENTIN’a. Ve artik HTML sayfasinda ki REACT ve REACT-DOM CDNs lerini silebiliriz.

1. Once **react react-dom** ‘i projemize yukluyoruz ve dahil etmek istedigimiz jsx dosyalarina asagidaki gibi IMPORT ederek kullaniyoruz. Tabi web-pack’le birlikte kullanabilmemiz icin yapmamiz gerekn bir kac ayar daha var webpack.config.js dosyasinin icinde babel ile alakali.

import React from 'react';

import ReactDOM from 'react-dom';

1. Projemize @babel/core, babel-loader modullerini yukleyerek dahil ediyoruz

+Cunku babelimizin yaptigi renderin dogru calisip wepack ile duzenlenip ekrana atilmasi icin loader ile ayarlanmasi gerekiyor. Yani LOADER webpack in davranislarini sekillendirir customize eder ona verilmis bir dosyayi yukletmek istedigimiz zaman.

+loader, module property ’si icinde rule olarak tanimlanir. Birden fazla kural belirlenebilir ama bu projemiz icin suanlik benim sadece 1 tane rule belirlemem yeterli. Bunuda asagidaki gibi yapacagim

+@babel/core, babel-cli gibidir.

+Babel-cli Command’da yani terminalde babel’i calistirmamiza yariyor ve boylelikle onlari belirledigimiz yerdeki dosyalarin icine gomeriz. babel-core ise webpack gibi toolslardan babel i calistirmamiza yarar

+babel-loader ise  bir webpack pluginidir. Ve babelin nasil calisacagini, ve ne zaman nereye babelin cikti dosyalarini servis edecegini, hangi dosyalari sececegini, hangilerini dahil etmeyecegini ogreten bir webpack pluginidir.Babel-core da iste bunun gibi pluginleri calistirir.

    module: {

        //Biz burada kurallari belirtiyoruz. Cok fazla kural koyabiliriz ama simdilik bize burada sadece asagidaki tek kural lazimdi onuda koyduk

        rules: [{

            loader: 'babel-loader',

            //Ne type dosylara calistirmak istedigimizi burada belirtiyoruz. Ben javascript dosyalari calistirmak istedigim icin .js i asagidaki sekilde ifade ettim $ isareti harflerin son sinirini \ isareti harflerin bas sinirini belirtir ve // ifadenin arasina yazilir. Detayli bilgi icin REGULAR EXPRESSION i arayin

            test: /\.js$/,

            //Dahil etmek istemedigimiz dosyalari ayarlariz EXCLUDE'de. Ve biz tum Node\_modules'un dahil olmamasini istiyoruz cunku bu moduller zaten islem gormus ve productiona surulmus dosyalar.

            exclude: /node\_modules/

        }]

    },

+BABEL icin ayarlar boylelikle tamam bundan sonra yapmamiz gereken tek sey .babelrc yani babel config dosyasini ayarlamak.

**BABEL CONFIG DOSYASI**

1. **babelrc** adinda root dizininde yani ana dizinde bir dosya olusturuyoruz. Artik babel-build ’i kullanmayacagimiz icin, webpack ’i ayarladiktan sonra webpack, babel ’a bunu bizim icin yaptiracagi icin ve babel bunu yapmaya calisirken ayarlari otomatik alacagi bir yol ve ayar dosyasi hazirliyoruz asagidaki gibi.

+presets kisminda babel-cli ‘in kullanacagi 2 tane babel donusturuculerini asagidaki gibi ayarliyoruz.

SOURCE MAP // KAYNAK HARITALAMA

1. Webpack.config.js dosyasina asagidaki sekilde devtool: heap-module-eval-source-map bu yazimizi objenin bir elemani olarak ekliyoruz.

devtool: 'cheap-module-eval-source-map',

+Ne ise yarar?

+Eger bunu kullanmazsak bir hata aldigimiz da tarayici konsolunda, tarayici bize bundle.js de olan hata yerini bize gosterecek. Fakat bunu webpack olusturuyor ve duzenliyor bizim duzenlememiz gereken bu dosya olmadigi icin bu yer gosterimi bizim isimize yaramaz. Bize dogru yeri gostermesi icn konsolda bunu ayarlamamiz gerekiyor burada.  +devtool bir stringe ayarlanmasi gerekiyor bunun icin. Bir kac cesit string ayarlamasi yapilabilir. Bazilari development icin iyidir bazilari production kismi icin iyidir. webpack.js.org dev tool section dan bunlara ayrintiuli bakabilirsiniz.

+Productionda farkli kod kullanilacak. Ayrintilardan bakin. Ayrntilar icin buradan bakin https://webpack.js.org/configuration/devtool/

devSERVER

1. Once webpack-dev-server modulumuzu npm install ile projemize ekliyoruz. Ve asagidaki gibi webpack.config.js dosyamizin icinde bir obje elemani olarak ekliyoruz.

devServer: {

        //contentBase'e path vermemiz gerekiyor fakat normal './' seklinde veremeyiz.

        //Tam dizin almamiz gerekiyor loaderde yaptigimiz gibi yapip path ile birlestirerek dizini veriyoruz.

        contentBase: path.join(\_\_dirname, 'public')

    }

ES6 CLASS PROPERTIES

1. Plugini ekleyebilmek icin once babel-plugin-transform-class-properties adinda olan BABEL-PLUGIN modulunu yukluyoruz.

+plugin kisminda ayarladigimiz ise CLASS ’larda degiskenler tanimlayabilmemizi, arrow function ‘lari kullanabilmemize yarayan bir BABEL PLUGIN ‘idir. Tabi bu plugini eklemek icin once 7. Maddede yazan modulu yukluyoruz.

+Ve pluginimizi .babelrc dosyasinda asagidaki sekildeki gibi ekliyoruz.

{

    "presets": [

        "@babel/preset-env",

        "@babel/preset-react"

    ],

    "plugins": [

        "transform-class-properties"

    ]

}

+Ve artik ES6 CLASS PORPERTIES ‘lerimizi kullanabiliriz jsx dosyalarimizda. Asagida OLD SYNTAX ve NEW SYNTAX ile kullanimlarini goruyoruz. Detayli inceleme ReactJS - Aciklamalar.docx dosyasinda ayni baslik altinda mevcuttur.

class OldSyntax {

    constructor() {

        this.name = 'Mike';

        this.getGreeting = this.getGreeting.bind(this);

    }

    getGreeting(){

        return `Hi. I am ${this.name}`;

    }

}

const oldSyntax = new OldSyntax();

console.log(oldSyntax);

//Eger getGreeting'i bind yaparak this le baglamazsak constructoda calismaz. Oysa babel’e ekledigimiz bu plugin sayesinde newSyntax’de buna gerek kalmiyor artik.

console.log(oldSyntax.getGreeting());

class NewSyntax {

    //name'e herhangi bir const var let gibi degisken atamamiza gerek yoktur NEW SYNTAX'de

    name = 'Jen';

    //oldSyntaxde yukaridaki gibi DEGISKEN'de, asagidaki gibi ARROW FONKSIYONDA tanimlayamayiz ama newSyntaxde bunu yapabiliriz.

    //Ve burada ARROW FONKSIYON kullanabildigimiz icin BIND ile this i baglamamiza gerek yok. Cunku ARROW FONKSIYONlar direk parent elementini this olarak kullanirlar.

    getGreeting = () => {

        return `Hi. My name is ${this.name}`;

    }

}

const newSyntax = new NewSyntax();

console.log(newSyntax);

console.log(newSyntax.getGreeting());

//Yukaridaki iki Syntax'de bize ayni sekilde cikti sonucu verir isim farkliliklari haric.