**Ders\_1 Express JS Nedir\_ExpressJs Oluşturma ?**

Express Js aslında Node Js http ve diğer modüller üzerine kurulmuş bir javascript webFramework ü dür.Biz bu expressJs sayesinde backend API kodlayabiliyoruz.Bir sonraki bölümde questionanswer rest api kodlayacağız.ve yine bu express ile kodlama yapacız.Express kullanımı oldukça kolaydır.Anlaşılması da kolaydır Ancak burda Javascript bilgimizin iyi olması gerekmektedir.Bu derste bir tane app oluşturucaz bu appimiz bizim API ımızın app i olacak ve biz buraya get put post ve delete request atıcaz.

1. Adım klasrde terminal açıp npm init yazdık
2. Kurulum aşamasında başlangıç server ına server.js adını verdik
3. Npm install express diyerek express js kuracağız.
4. Her seferinde çalıştırmak yerine otomatik başlatacak bir paket daha indiricez.(herhagi değişiklik yaptığımızda serverımız tekrardan başlatılacak. Sadece geliştrime ortamında çalıştırıcaz
5. npm install --save-dev nodemon
6. package.json içinde script bölümünde test yanına "dev" : "nodemon server.js" ekliyoruz.
7. Script.js oluşturduk ve içine const express = require("express"); diyerek dahil etmiş olduk.
8. Sıradaki adımda ise express in constructorından bir adet app oluşturuyoruz.

const app = express();

1. Serverımız için bir port numarası atamalıyız bunun için boş bir port bulmalıyız boş port bulmak için cmd de C:\Users\cihan>netstat -an |find /i "listening" araması yada bellirli bir port numarası dolu mu boşmu görmek için

C:\Users\cihan>netstat -an |find  "portNumarası" bende 5040 uygun ben 5040 vereceğim bunun için de const PORT =5040; yazarak port numarası atadık.

Artık ben bu uygulamamın 5040 portunda çalışması için **app.listen(PORT,{}=>{console.log(“server started”);})**

1. npm run dev diyerek serverımızı çalıştırabiliriz.

C:\Users\cihan\Desktop\Web\_Egitim\_2\ExpressJS\_DersNotları\Ders-1\_ExpressJs\_Nedir\_ExpressJs\_Olusturma\express\_fundamentals>npm run dev

> express\_fundamentals@1.0.0 dev

> nodemon server.js

[nodemon] 2.0.16

[nodemon] to restart at any time, enter `rs`

[nodemon] watching path(s): \*.\*

[nodemon] watching extensions: js,mjs,json

[nodemon] starting `node server.js`

Server Started..

**Ders\_2 GET Request ,Postman kullanımı,** **middleware ?**

Dersin bu kısmında veri tabanı olmayacak bu sonraki derslerde olacak biz burda get request nasıl oluşturuluyor bunu görmek istiyoruz bunun için öncelikle içinde json objeleri barındıran bir users değişkeni oluşturucaz.

1. const users = [{id:1,name:”Cihan Arğan”,place :”Ankara”}, {id:2,name:”Serhat Say”,place :”Londra”}]

Not: temel mantık şu şekilde kare = front end bir uygulama flutter kodlin android app vs olabilir oval kare ise bizim serverımız kareden oval olana bir get request gelince bizim serverımız bir response döndürecek post işlemlerinde ise frontend den gelen requestte ek bir bilgi gelecek serverımız bunu yuvarlak ile gösterdiğimiz veri tabanına gönderecek kaydettikten sonra serverımız bir mesaj döndürür.

11) App oluşturduktan sonra bizim get requesti karşılıyacak bir işlem gerçekleştirmemiz lazım.Bizim serverımıza get request yapıldığı zaman ne response döneceğimizi söylemediğimiz için ilk başta bunu vermemiz lazım.Bizim bu appimize bir ttane get request olduğunda get metodunu kullanırız.get metodu ikitane argüman alır .birincisi hangi routa örneğin <http://localhost:5040/users> olduğunda anlamı katıyor. 2.sinde ise bizim burda response muzu göndermemiz gerekiyor buda bir callback fonksiyon aslında (req,res,next)=> bizim requestimiz response muz ve henüz öğrenmediğimiz next argüman olarak bu fonksiyonda yer alacak yani bizim requestimiz içindeki gönderdiğimiz bilgi burdaki req objesinde bunları express gönderiyor.response döneceksek bunlarda res içinde next daha sonradan anlatılacak. Bu routa bir get reqıest olursa bizim fonksiyonumuz çalışacak.istersen console.log ile req ve resp görebilirsiniz fakat ben burda 3 çeşit veri response olarak döndürebiliyordum birincisi text res.send(“hello express”) diyerek

İkinci veri tipimiz ise html dökümü göndere biliyoruz yine res.send ile 3. Olarak ise json dosyası döndürebiliyoruz. Biz users değişkeni içindeki jsonu göndericez res.json(users); diyerek gönderdik.

app.get('/users', (req, res, next) => {

//res.send('hello express');//text

//res.send("<h1> Hello Express </h1>");//html

res.json(users);//json

});

Get metodu içinde kalan kısma request handler deniyor.Diğer bölümlerde controller fonksiyonlarda diyeceğiz ama doğrusu request handler olarak bilinmelidir.req ve next express tarafından gönderiliyor.

7.adım postman => frontend tarafında bizim requestlerimizi atmak için kullandığımız bir uygulamadır.bu hem chrome eklentisi şeklinde kullanılıyor hemde bilgisayara indirip kullanabiliyoruz.postman chrome extension şeklinde kurabiliriz.indirdikten sonra new collection diyoruz name kısmına fundamentals dedik GET seçip <http://localhost:5040/users> dedikten sonra send diyoruz bizim json dosya formatımız geldi raw diyerek görüntüsünü görüyoruz prewiev diyerek dosyanın içini görüyoruz.

Middleware nedir bu kısımda bunu öğreneceğiz.

app.get(“/users”) dediğimizde biz tüm users ları döndürmüştük burası başka bir rout olsa yani bir userın yeni bir makale eklemesi gibi bir router olsa /articles/add şeklinde bir route a biz get request yada herhangi bir request isteği yapsak. Şimdi burda yeni bir makale eklemek için ilk başta frontend tarafındaki kullanıcının daha öncesinde serwer tarafına login olması gerekmektedir.Yani oturum açması gerekmektedir.yani biz burda request yaptığımız zaman direk o sayfayı göstermek yerine ilk başta bizim bu kullanıcı login olmuş mu olmamış mı sorgulamamız gerekmektedir.yani bir controllden geçirmemiz gerekmektedir.bunu request handler içinde yapabiliriz.request handler içinde bu oturum kontrolü yapılıp yapılmama durumunu kontrol ediyorsak ve yapılmışsa biz bu sayfamızı gösterebiliriz.yapılmadıysada geriye başka bir sonuç döndürebiliriz.nasıl bir sonuc döndürebiliriz you are not autoryce. Yani giriş yapmadınız şeklinde bir dönüş yapabilirsiniz.biz bu rout içerisinde bu kontrolu yapıyoruz ama başka routlar içinde de bu kontrolü yapmamız gerekecektir.ozaman burdaki işlemlerin aynısınıburdada yapmamız gerekmektedir.Uygulamamızda 100 tane rout varsa her birinde bu kontrol yapılıyorsa bu oldukca zorlar.Peki bizim bunları tek bir merkeze toplayan ve bu requestimizin gönderilidiği zaman arada bunu kontrol eden mekanizmamız olsa çok iyi olurdu.Bu durum express js içinde Middleware(ara kataman handlerları) olarak adlandırıldı.

Biz herhangi bir routa gitmeden önce eğer o routta bizim login olup olmama kontrolu yaptığımız bir tane middleware çalışıyorsa bu rout içinde bunu kontrol etmemize gerek kalmadan sadece biz bunu middleware içinde bunu kontrol ediyoruz ve middleware eğer izin verirse bir sonraki requeste geçiyoruz..Biz middleware function larımızı belli bir request handlerdan önce çalıştırabiliriz.1 tane access controlü yapan middileware olsun ilk başta request yapıldığı zaman bu request middleware geliyor.gelende (req,res,next) alan request handlerdır.access kontrol içinde kontrol gerçekleştirilir eğer izin verirsek bir sonraki yere gitmesi için next parametresini kullanıyoruz.izin vermezsek response middleware içerisinden döndürülür server a ulaşmadan.İstersek server tarafındaki request handler döndürmeden öncede birden fazlada middileware oluşturabilirdik.

Biz middileware.js dosyası oluşturup middleware functionlarımızı başkabir dosyada oluşturacağız.

Middleware.js te

const accessControl = (req, res, next) => {

    console.log('Middleware : Acccess Control');

    next();

};

//ana yapısı budur daha sonra ben server.js de kullanacağım için ilk başta burda export etmem gerekiyor.

module.exports = {

    accessControl

};

//sonra sever.js içinde kullanıcaz.

Daha sonra server.js içine geçip en üstlerde bu dosyayı ve accessControl dahil etmemiz gerekecek.

const { accessControl } = require('./middleware'); //accesscontrolü middileware.js ten alıyoruz.

Bizim middleware fonksiyonlarımız 2 türlü kullanılabiliyor 1.si sadece o route a özgü middleware 2.side uygulama katmanında her routta kullanmak istediğimiz middlewarelerdir.onun için siz bunu uygulama katmanında kullanmak istiyorsanız bu routlarımızdan hemen önce bunu kullanmak zorundasınız bu örnekte biz app.get ten önce kullanacağız. middleware kullanmak için use metodunu kullanıyoruz.içine de dahil ettiğimiz accesControl veriliyor.başka routlarda olursa app.get gibi bir kaç tane daha rout varsa hepsinden önce bir kaç defa kullanabiliriz. her seferinde bu middleware ın çalışması gerekmektedir.

App.get kullanımından hemen önce

app.use(accessControl); diyerek uygulama kapsamında middleware ımızı kullanmış oluyoruz. Kontrolünü ise postman üzerinden istek yollarak console ekranından kontrol edeceğiz.vscode içindeki terminal ekranında çıktımızı gördük middleware: Access Control çıktısını almış olduk.

NOT: Eğer next kullanılmasaydı bizim middleware içindeki fonk çalışırdı ama bir response dönemezdi bu yüzden sürekli request sending de takılı kalırdı.

Uygulama kapsamında middleware kullanıımı bu kadar şimdi rout kapsamında kullanıımı görmek istiyoruz burda ise bir rout daha oluşturucaz (/products).bizim get requestimiz iki route içinde geldiğinde middleware ikisi içinde çalışacak. Fakat biz /users ta kullanılsın istiyorsak app.get(“/users”,accessControl,(req,res,next) diyerek access Controlden geçersen users içindeki kod bloğu çalışsın demiş oluyoruz. Bu durumda middleware bu rout için çalışmış olacak.

Middleware.js dosyamızı geliştirelim

const accessControl = (req, res, next) => {

    const access = false;

    if(!access){

        res.json({

            success:false,

            message:"You are not authorized."

        })

    }

    next();

};

Dediğimizde istek gönderdiğimizde access :false olduğu için yolladığımız res.json içindeki çıktıyı alırız. Hatamızı başarılı şekilde aldık next demedi.server.js içine ulaştırmadan middleware içinden döndürmüş olduk.access :true olsaydı next çalışacak diğer sonuç ekrana gelecekti.

 //Not2: bizim hata dahi alsak status değerimiz 200 OK geliyor ama biz bunu değiştire biliyoruz.res.status().json() diyerek json çıktısından önce status değerimizi biz verebiliriz. status(401) diyerek hataya uygun status gönderebiliriz.

Not-3 Birden fazla middleware kullanmak istiyorsak

const defaultMiddleware = (req, res, next) => {

    console.log('Default Middleware');

    next();

};

module.exports = {

    accessControl,

    defaultMiddleware

};

Diyerek ikinci middleware oluşturduk

Server.js içinde

const { accessControl,defaultMiddleware } = require('./middleware');

diyerek dahil ettik

app.get('/users', [accessControl,defaultMiddleware], (req, res, next) => {

ya bu şekilde yada

app.get('/users', accessControl, defaultMiddleware, (req, res, next) => {

bu şekilde iki middleware birlikte kullanılabilir sırası önemli ilk önce hangisi istiyorsak onu yazmamız lazım.

**Ders\_3 Post Request Gönderimi ve Veri Ekleme ?**

Post request isteği nasıl yapılıyor bunu öğrenelim

app.post(“/users”,(req,res,next)=>{

res.json({

success:true,

data:”Post Request”

})

})

Not:Kontrolümüzü Postman üzerinden POST metodunu seçerek yapıyoruz.

Normalde post işlemi için frontend tarafından bir veri girilmesi gerekli fakat biz burda postman üzerinde post işlemini seçtikten sonra body kısmından raw kısmından verimizi json olarak göndereceğiz.Oyüzden text kısmının olduğu yerden json seçeneğini seçmemiz gerekecek.burda json formatında verimizi yazıyoruz. Save As diyerek oluşturduğumuz fundamental collection içinde add new user isminde kaydediyoruz. Send diyerek

success:true,

data:”Post Request”

çıktıyı alıyoruz eklenen veriyi yada verileri görmek için

post işlemi içinde console.log(req.body); diyerek görebiliriz fakat bundan önce bizim express in bize veridiği middleware uygulama kapsamında kullanmamız gekli yoksa undifened çıktısı alırız.app.use(express.json()) diyerek yukarıda bunu dahil etmeliyiz.

//console.log(req.body);//Burda alıdğımız req.bodyi biz yukardaki users verilerimizin arasına kaydetmeliyiz.

const newdata = req.body;

users.push(newdata); diyerek users verilerimizin arasına dışarıdan veri eklemiş olduk.

{ id: 3, name: 'Sezai Arğan', place: 'Aksaray' } verisini eklemiş olduk.

Ayrıca res.json içindeki data kısmında data:users diyerek verilerimizi de gösterebiliriz.

"success": true,

"data": [

{

"id": 1,

"name": "Cihan Arğan",

"place": "Ankara"

},

{

"id": 2,

"name": "Fazıl Say",

"place": "Londra"

},

{

"id": 3,

"name": "Sezai Arğan",

"place": "Aksaray"

}

]

Şeklinde görebiliriz.Uygulamayı kapatıp açınca mecburen silinmiş olacak çünkü biz veritabanı kullanmadık.

**Ders\_4 Put Request Gönderimi ve Veri Güncelleme ?**

Bu dersimizde veri güncelleme işlemi gerçekleştireceğiz.Verilerimize id eklediğimiz için bunlara göre güncelleme yapacağız

users/1 şeklinde bir request gelirse buna göre güncelleme yapıcaz

app.put('/users/:id', (req, res, next) => {

    res.json({

        success: true,

        data: 'Put Request'

    });

});

Bu id yi nasıl alıcaz req.params.id ile bu idyi alabiliriz. Postman içinde put request yapıcaz users/1 ekleyeceğiz.

Console.log(req.params.id);ile bakabiliriz sonuc string geldiği için işlem yaparken int yapmalıyız.

app.put('/users/:id', (req, res, next) => {

    const id = parseInt(req.params.id);

    for (let i = 0; i < users.length; i++) {

        if (users[i].id === id) {

            users[i] = {

                ...users[i], //users içinde gönderilen id ye ait eleman bilgileri gelecek

                ...req.body //req body ile güncelenmesi istenen veriler güncellenecek.

            };

        }

    }

    res.json({

        success: true,

        data: users

    });

});

Postman tarafında put seçeneğinin ordan body kısmına orda da raw tarafına geçip orda json formatını seçip güncelleme yapmak istediğini değiştirebilirsin ben place bilgisini değiştirecğim.

**Ders\_5 Delete Request Gönderimi ve Veri silme ?**

Yukarıdaki işlemin hemen hemen aynısını yapıyoruz kod bloğu aşşağıdadır.sadece body işlemleri yok sadece silme işlemini gerçekleştiriyor.

//11-Delete request

app.delete('/users/:id', (req, res, next) => {

    const id = parseInt(req.params.id);

    for (let i = 0; i < users.length; i++) {

        if (users[i].id === id) {

            users.splice(i, 1);

        }

    }

    res.json({

        success: true,

        data: users

    });

});