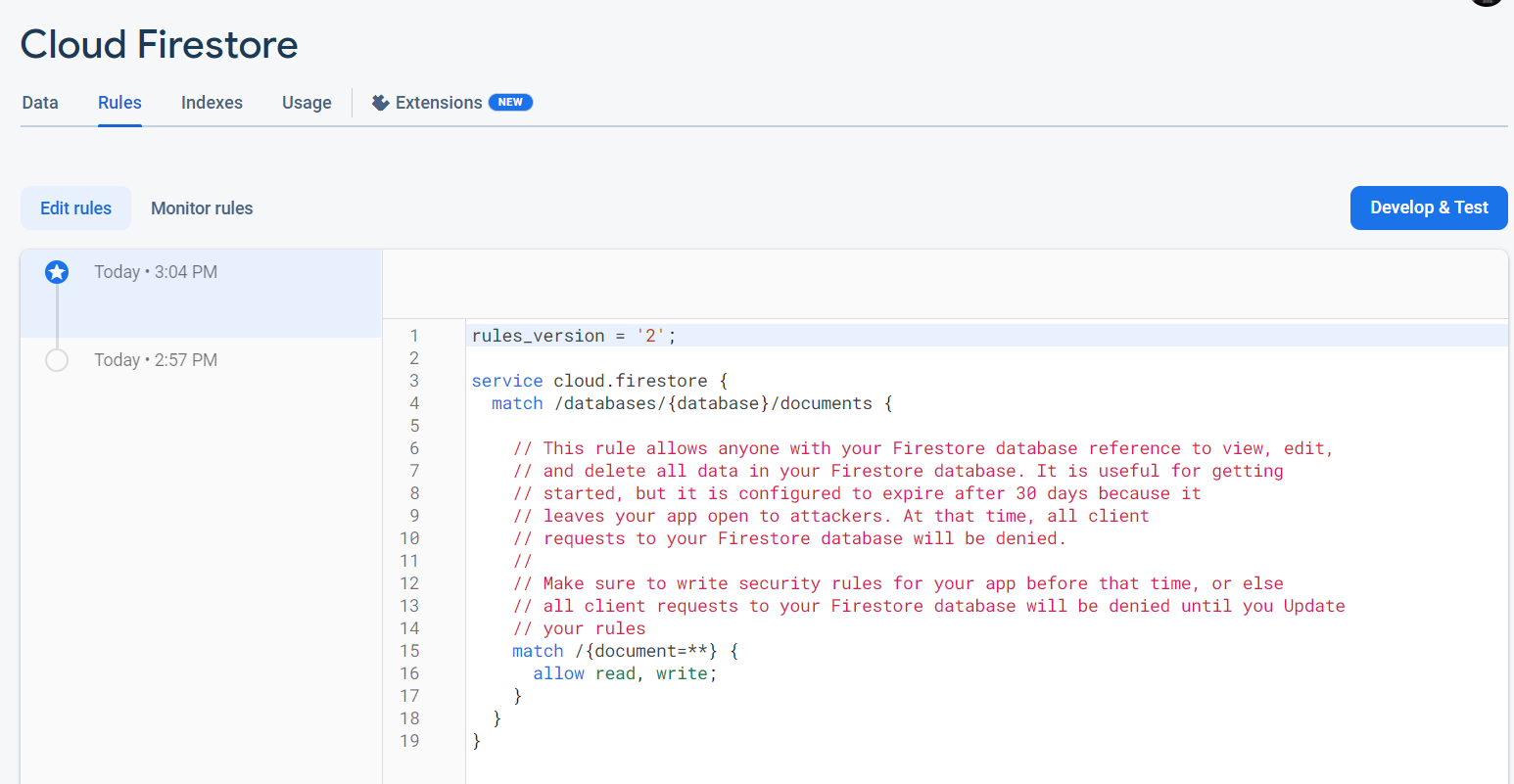
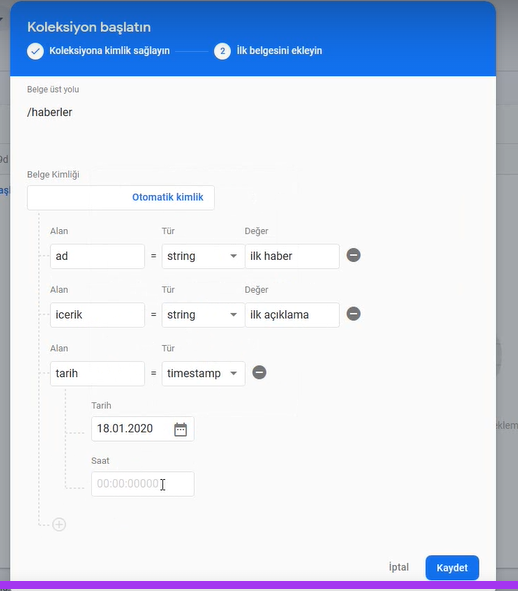
**FİREBASE**

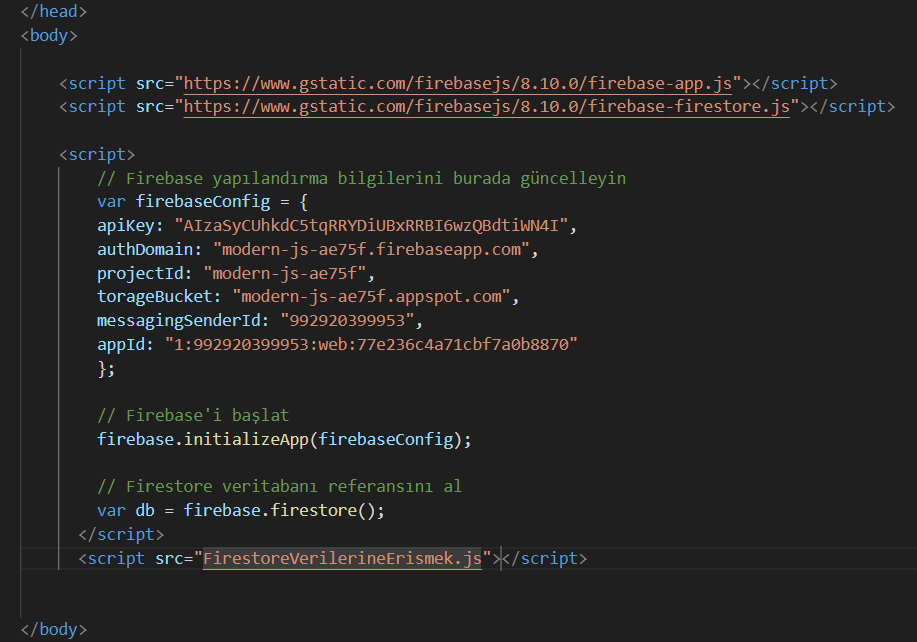
* Bir JavaScript uygulaması oluşturduğumuzda verileri bir yerde saklama ihtiyacı duyarız. Çünkü daha sonradan bu uygulamayı tekrardan açtığımızda daha önceden oluşturduğumuz verileri karşımızda görmek isteriz. Bunu da verileri saklayabilme ile gerçekleştiriyoruz. Aslında bir veritabanında o verileri kaydetme ve daha sonra o verileri çağırma şeklinde oluyor. Firebasein Cloudfirestore ile yapabiliyoruz. İstersek mongodb sql oracle şeklinde de bu verileri saklayabiliriz. Googleın desteklediği firebase platformu içerisindeki cloudfirestore da verilerimizi saklayacaz.
* Firebase sitesinde oturum açıp konsola gite tıkladığımızda bir proje üretmemiz isteniyor bu projeler sayesinde uygulamamızla cloudfirestoru birleştirebileceğiz. Burada bir proje oluştururuz. Firebase proje adının yanında unic benzersiz bir kimlik ad oluşturuyor böylece oluşturduğumuz proje dünya çapında benzersiz oluyor. Devam ettiğimizde projeye Google analytics ekleyip eklemediğmizi soruyor şimdilik devre dışı bırakıp devam edelim.
* Projemiz içerisinde authoncation, stroge, firestore, cloudfunction, hosting gibi bölümleri barındaracak. Biz şimdilik cloudfirestorea ihtiyacımız olacak. Ardından auth kısmından bir proje oluşturacağız.
* Proje oluştuktan sonra sol menüde build menüsü altında kullanabileceğimiz bölümler oluştu. Bu menüden firesrote database kısmına gelelim. Bu sayede burada bir veritabanı oluşturabiliriz. Burada 2 adet veritabanı karşımıza çıkıyor. Birincisi kullanacağımız Cloud firestore ikinciside alternatif olarak kullanılan şimdilerde realtime databasetir. Cloudfire store için veritabanı oluştur diyelim. Burada 2 seçenek karşımıza çıkar. Üretim modunda başlatma diğeri ise test modunda başla kısmı, üretim modunda başlattığımız zaman burada allow read diye güvenlik önlemi oluşturulmuş, eğer false ise diyor read ve write özelliğini allow et diyor. Bu üçüncü tarafın db yi okuma ve yazma işlemlirini reddetmesi anlamına gelir. Üretim modunda değil test modunda başlayalım. Test modunda okuma ve yazma aktif hale geliyor direkt ama sadece 30 gün boyunca bu veritabanını kullanabiliyoruz.
* Devam et dedikten sonra firestoreun konumunu belirtiriz. Biz euur3(europe-west) seçelim bize daha yakın. Enable diyelim. Böylece daha hızlı olmasını bekleriz sunucu daha yakın olunca.
* Database oluştuktan sonra kurallar menüsüne bakalım. Service Cloud.firestore diyerek firestoreumuzu aktif ediyoruz ve match /databases/{database}/documents diyerek database içerisindeki oluşturulan dökümanlara erişip burada bütün dökümanlar için bir kural yazabiliriz. Buradaki allow read,write: if request.time < timestamp.date[2020,2,17] diyerek oluşturma zamanı şu zamana kadar benim bu databasei okuma ve yazma imkanım var diyor. Eğer ben burada bu kodu write tan sonrasını silersem ve write yanına ; koyarsam ve yayınla publish dersem artık sınırsız bir şekilde süre olmadan bu veri tabanını kullanabilirim.



* Daha sonra data yani veriler sekmesine geçelim. Burada ben verilerimi oluşturabilirim. Cloudfirestore nosql yapısını kullanan bir veritabanı o yüzden tablolar yerine burada koleksiyonlar ve dökümanlar var. Koleksiyonu başlat dediğim zaman benden bir koleksiyon oluşturmamı isteyecek. Burada örneğin haberler diyerek haberleri içerisinde tutacağım bir koleksiyon oluşturabilirim. İleri dediğimiz zaman haberler koleksiyonunu oluşturduk içerisine artık belge eklemeye başlayacağız. Koleksiyonu tablo belgeyi ise tablonun kolonları olarak düşünebiliriz. Firestore bu belgeye otomatik kimlik ataması yapabiliyor. Kimlikten sonra dökümanın adını tanımlıyoruz ve belgenin türünü tanımlayıp içeriğini değerini tanımlıyoruz. İkinci doküman eklemek için alan ekle kısmına tıklarız. Tarihleri timestamp türünde oluşturmamız gerekir dikkat.



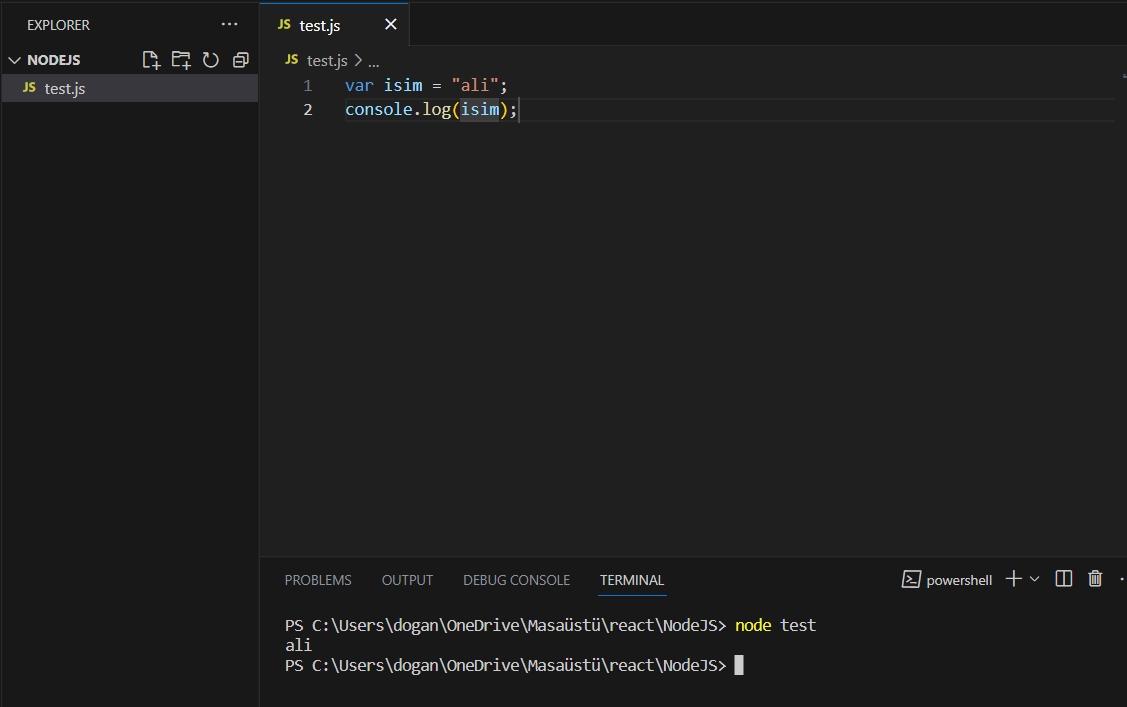
* Kaydet dediğim zaman cloudfirestore da haberler koleksiyonu içerisine ilk belgemizi ekliyoruz ve alanlarını gözlemleyebiliyoruz belgeye tıkladığımız zaman. Belgenin id si ise otomatik olarak oluşturuldu. Burada bu koleksiyene başka belgelerde ekleyebiliyoruz.
* Oluşturduğumuz bu veritabanını yani firebasei projeye bağlamak için Project overview kısmına tıklıyoruz sol menüden. Burada bulunan web ikonunu seçiyoruz. Daha sonra uygulama için takma isim veriyoruz. Bu uygulama için firebase hosting ayarlayın diyor bunu şimdilik tıklamayalım. Uygulamayı kaydet diyelim. Bundan sonra firebasein sdksı oluşturalacak. Bu sdk bizim ihityacımız olan alanı hem npm ile hemde script tagları ile alabiliyoruz şimdilik script taglari ile alalım. Bu alanı kopyalayıp index sayfamıza body etiketinin en altına yapıştırılaım. Kendisi firebase cdnini import etmiş ama benim ayrıca firestore cdnini de import etmem gerekli. Birde firestoredan bir database referansına ihtiyacım var database e ulaşmak için onuda initializeapp metotunun altında const db = firebase.firestore(); diyerek db değikeni ile firestore a ve dbye erişmiş olacağız. Örnek yapı aşağıda resimde



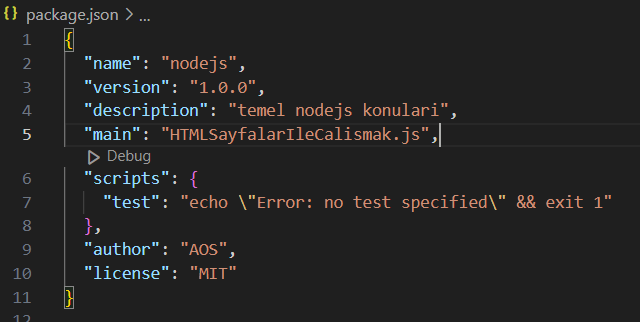
* Daha sonra Firestore verilerine erişmek için script dosyasına geçeriz. Konsola bu db değişkenini yazdırarak firestorumuzun bilgilerine ulaşabiliyoruz. Db.collection() metotuyla koleksiyonlarıma ulaşabiliyoruz. Bu metota parametre olarak koleksiyon ismini veririz. İlgili işlemler kodlar script dosyasında mevcut ordan bakabilirsin. GERİ KALAN YAPILACAK CRUD İŞLEMLER İLGİLİ BÖLÜMÜN SCRİPT DOSYASINDA ORADAN BAK.

**NODEJS**

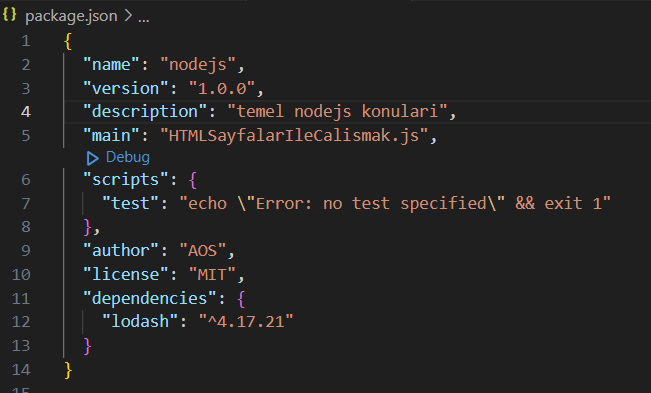
* **Nodejs Nedir:** Javascript ile server-side uygulamalar yazabileceğimiz, Joyent tarafından 2009 yılında geliştirilen Javascript Runtime platformudur. Bir exe dosyasını pc’ye kurduktan sonra artık server taraflı uygulamaları javascript ile yazabileceğiz. V8 denilen javascript motoru üzerinde çalışır.
* Web uygulaması geliştirirken yazdığımız kodlar ikiye bölünür;
* Client Side
* Server Side
* Client Side tarayıcıların anlayacağı kodlardır ve server side’dan aldığı bilgileri kullanıcıya göstermek için çalışırlar.
* Server Side ise server tarafında yazılan ve web uygulamalarının diğer platformlar ile iletişim kurmak (DB, API vb.) ve kullanıcılardan gelen isteklere yanıt vermek için çalışırlar.
* V8 Motoru Nedir: V8 Google tarafından geliştirilen, Chrome tarayıcısında üzerinde çalıştığı C, C++ ve javascript diller ile kodlanan bir motordur ve tek amacı javascript kodunu makine koduna çevirmektir.
* NodeJs javascript ile yazılan kodu makine koduna dönüştürürken V8 Engine kullanılır.
* **NodeJs İle Neler Yapılabilir?;**
* Dosyalar okunabilir ve dosyalara yazılabilir
* Veri tabanına erişim sağlanabilir
* Web uygulamaları geliştirilebilir
* Node js kurulu mu değil mi veya hangi versiyonu kurulu öğrenmek için cmd ye node -v yazabiliriz.
* Cmd ye sadece node yazarak artık nodejs komutlarını kullanabileceğim bir alan açmış oluyorum. Ctrl+d ile bu alandan çıkabiliriz.
* mkdir isim yazarak istenilen alana klasör oluşturabiliyoruz boş yeni klasör. Sonrasında cd oisim yazarak o klasöre geçiş yapabiliriz.
* Projeyi kurmak istediğimiz konumda cmdyi açarak code . yazarak bu konum için visual studioyu çalıştırmış oluruz.
* Bu konumun içine test.js diye dosya açalım ve içerisine bir değişken tanımlayıp konsola yazdıralım. İşte ayrım burada başlıyor nodejs kullanmadan direkt sadece javascript ile bu işlemi yaparsak sonucu konsolda görmek için bir tarayıcının konsoluna ihtiyacımız olurdu. Ancak nodejs ile çalıştırmak için vs üzerinde terminali açıyoruz node dosyanınadı böyle yazdığımız zaman direkt terminal ekranında tarayıcı konsoluna ihtiyaç duymadan sonucu görebiliyoruz.



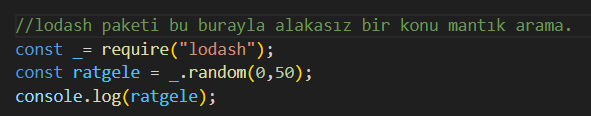
* GERİYE KALAN ANLATIMLAR KOD OLDUĞU İÇİN İLGİLİ JS KODLARINDADIR.
* NPM yani Node package manager ile dosya indirme paket indirme işlemini gerçekleştirebiliyoruz.
* Nodemon paketi ile serverda bir değişiklik yaptığımızda serveri durdurup tekrar açmamız gerkiyordu işte bu paketle bunun önüne geçebiliyoruz. npm install -g nodemon diyerek bu paketi bilgisayarıma global olarak yükleyebilirim. Global olarak yüklenmesi sayesinde isteğim zaman farklı bir projede de olsa nodemon u çalıştırabiliyorum. Artık projemi node projeismi ile ayağa kaldırmak yerine nodemon projeismi yazrak ayağa kaldırıp bu server kapat aç işlemlerinden kurtulabilriim. Powersheelde çalışması için yönetici olarak açmak gerekiyor dikkat.
* Npm init dediğim zaman package-jsonu oluşturmuş oluyorum, package json bizim node ile oluşturduğumuz projelerin ana yürütme paketidir. Npm init dedikten sonra bize package name sorar boş bırakırsak konumdaki klasörün adını alır. Sonra versiyonu soruyor boş bırakabiliriz kendi ayarlar. Description soruyor açıklama metni girebilriiz daha sonra entrypoint soruyor yani bu varsayılan başlangıç projemizi soruyor onun ismini girmeliyiz burda .js olarak tam adını girmeliyiz. Daha sonra test commad, git repository, keywords boş bırakabiliriz author yazarı yazalım, lisansınıda MIT olarak ayarlarız. Son olarak y dersek oluşmuş oluyor. Npm init -y diyerek anında hızlı kurulum yapabiliyoruz. Npm i -D nodemon diyerek nodemon paketini yeni projenin package.jsonunda devDependencies kısmına kurulum yapıp yeni paketleri böyle kullanabiliriz. Tabi bunları daha önceden bilgisayarımıza global şekilde kurmuş olmamız gerekli. Dependencies kısmına gelen paketler bu projenin bağlı olduğu paketlerdir yani yarın öbür gün node modüle silindiği zaman ve tekrar yüklendiği zaman burdaki dependencies paketlerine göre yeniden yüklenir. O anlık kullanacağımız paketleri bu dependencies kısmına almamak lazım.
* "scripts": {
* "start": "node server.js",
* "dev": "nodemon server.js"
* },
* Yukarıdaki bilgileri package.json dosyasında kendimize göre düzenleyebiliriz start durumu npm proje isminde çalışacak koddur, dev özelliği ise npm run dev dediğimizde çalışacak özelliktir.

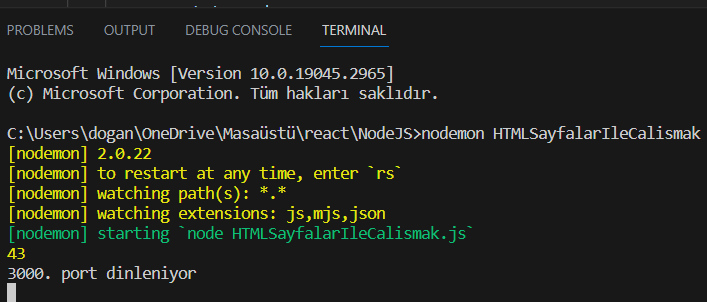


* Burda main attributeunu HTMLSayfalarIleCalismak.js olarak ayarladığımız için start anında bunu çalıştıracak.
* Bir başka paket olan lodash paketini kullanabiliriz. Bu paket içerisinde bir takım metotlar barındırır. Npm i –save lodash diyerek kurabiliriz.



* Burada yeni paketi kurarken –save dediğimiz için package-json dosyamızın içinde dependencies altında paketimizin geldiğini görebliyoruz ve sürümünüde görürüz.
* Lodash paketinin kullanımı aşağıdaki gibi.





**NODEJS VE REST API**

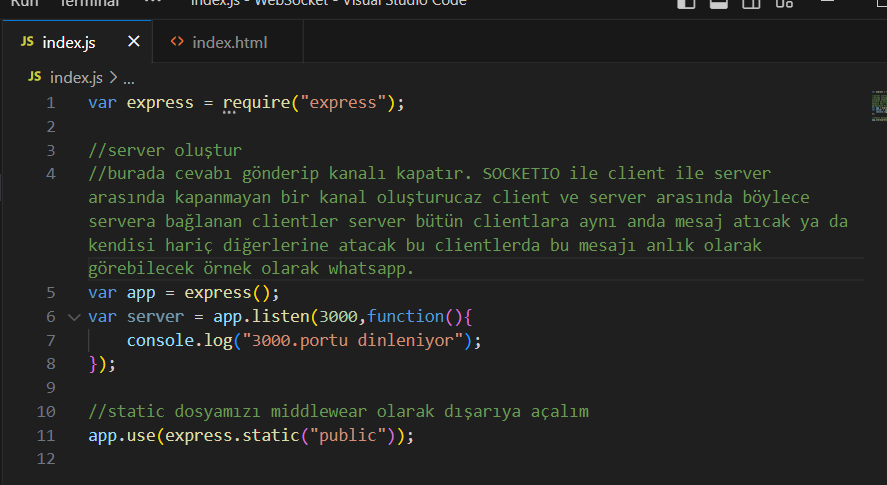
* **Rest Nedir?;**
* Rest (Representational State Transfer), 2000 yılında Roy Fielding tarafından doktora tezinde tanıtılmış ve tanımlanmıştır. REST, dağıtık sistemler tasarlamak için kullanılan bir mimari tarzıdır.
* REST, client-server arasındaki haberleşmeyi sağlayan http protokolü üzerinden çalışan bir mimaridir.
* REST, servis yönelimli mimari üzerine oluşturulan yazılımlarda kullanılan bir transfer yöntemidir.
* İstemci ve sunucu arasında XML ve JSON verilerini taşıyarak uygulamanın haberleşmesini sağlar. REST mimarisini kullanan servislere ise RESTful servis denir.
* **HTTP Metotları Nelerdir?;**
* GET
* POST
* PUT
* PATCH
* DELETE
* GET veri listeleme, veri görüntülemek için kullanılır. GET requestler güvenli ve idempotent olmalıdır, yani aynı parametrelerle kaç kez tekrar ettiğine bakılmaksızın sonuç aynıdır. GET ile veri gönderilirken, adres çubuğunda gönderilir. Gönderilen değişkenler ve veriler adres çubuğunda görüntülendiği için güvenilir değildir. Ancak veriler MD5 şifreleme ile güvenilir bir şekilde de gönderilebilir.
* POST veri eklemek için kullanılır ancak mevcut olan bir veriyi güncellemek için de kullanılır. Doğrudan sayfaya veri gönderilir ve veriler adres çubuğunda görünmez. GET e göre daha güvenilir yöntemdir.
* PUT veri güncellemek için kullanılır. PUTun POSTdan farkı idempotent olmasıdır. Yani bir request birden fazla kez tekrarlansa da sonucunun aynı olmasıdır.
* PATCH verinin sadece bir parçasını güncellemek için kullanılır.
* DELETE kaynaktan veriyi silmek için kullanılır.
* **MVC Nedir?;**
* Yazdığımız uygulamanın iş mantığı ile (business logic) kullanıcı arayüzünü birbirinden ayrıştıran, uygulamanın farklı amaçlara hizmet eden kısımlarının birbirine girmesini engelleyen yazılım mimarisidir.
* Model
* View
* Controller
* MODEL Uygulamada verileri temsil eden bir katmandır. Bu katmanda verilerin modelleri oluşturulur veri tabanı ile bu katman ile iletişim kurulur.
* VİEW uygulamanın görünen yüzüdür. Veri tabanındaki verilerin nasıl görüntüleneceğinin ayarlandığı katmandır.
* CONTROLLER Model ile view arasında bağlantı kuran katmandır. Kullanıcıdan gelen istekler bu katmanda değerlendirilip uygun yere yönlendirilir. Tüm bunların kullanımı uygulamada vardır.

EXPRESSJS MONGODB NODEJS

* EXprexxJs Nodejs tabanlı server işlemlerini kolaylaştırmak amacıyla oluşturulmuş bir frameworktur.
* ExpressJs ile Nodejsye göre daha çok daha basit, güvenli ve kapsamlı bir şekilde server oluşturma işlemi yapabiliyoruz.
* NodeJsde html sayfalarımızı görüntülerken içerisinde dinamik bir yapı oluşturmak istediğimizde bir viewengine kullanmalıyız EJS View Engine kullanırız.
* MongoDb ile veritabanı oluşturup Mongoose paketi ile veritabanı işlemleri yapabiliyoruz.
* ExpressJs nodejs ile kolay yoldan ve güvenli bir şekilde server oluşturmamızı sağlayan pakettir. Npm i Express diyerek indirebiliriz. ExpressJS ile server oluşturma kullanımı uygulamada var.
* EJS kullanmak için npm i ejs deriz. Artık html sayfalarımı aynı cshmtl yaptığımız gibi bu seferde ..ejs olarak kaydediyoruz. Yani dinamik html sayfaları yapmak için bunu kullanıyoruz. Kullanımları uygulamada var.
* Morgan middlewear ile http requestlerinin loglarını tutabiliyoruz. Npm i morgan diyerek indirebiliriz. Kullanımı uygulamada var.
* **MongoDb Nedir?:**
* Veri tabanı ihtiyacımız olan veya daha sonraları ihtiyacımız olacak bütün verileri depoladığımız sistemdir.
* Günümüz uygulamalarının (web,mobil,masaüstü) neredeyse tamamında yerel veya uzak sunucuya bağlantılı veri tabanları bulunur.
* Veri tabanlarına depoladığımız bu bilgileri daha sonraları kullanabilir, üzerlerine düzenlemeler yapılabilir veya bu verileri anlamlandırarak bilgiler elde edebiliriz.
* **2 tür veri tabanı bulunur:**
* İlişkisel veri tabanı
* NoSQL veri tabanı
* **NOSQL Nedir?;**
* Öncelikle MongoDB tıpkı Firebase gibi NOSQL veritabanı sistemini kullanır.
* Son yıllarda verinin inanılmaz boyutlara ulaşması ve katlanarak artması sonuccunda mevcut olarak kullanılan ilişkisel veritabanı sistemleri yerine ortaya atılmış bir kavramdır.
* İlişkili veritabanı sistemleri ile arasındaki en büyük fark ilişkisel veritabanı sistemlerinde veriler tablo ve sütunlar ile ilişkili bir şekilde tutulurken NoSQLde json bir yapıda tutulmasıdır.
* NOSQL daha hızlı daha ölçeklendirilebilirdir ve ilişkisel veritanaındaki gibi tablolar arasında bir ilişki yoktur.
* **NOSQL TÜRLERİ ?;**
* Key-Value
* Döküman Tabanlı (MongoDb-FireBase)
* Sütun tabanlı
* Graph (Çizelge)
* **MongoDb Nedir ?;**
* 2009 yılında geliştirilmiş açık kaynak kodlu NOSQL veritabanıdır
* **MongoDB Yapıları ?;**
* Collection
* Document
* Field
* Mongos paketini kurmak için npm i mongosee diyebiliriz.

**WEB SOCKET ve SOCKETIO**

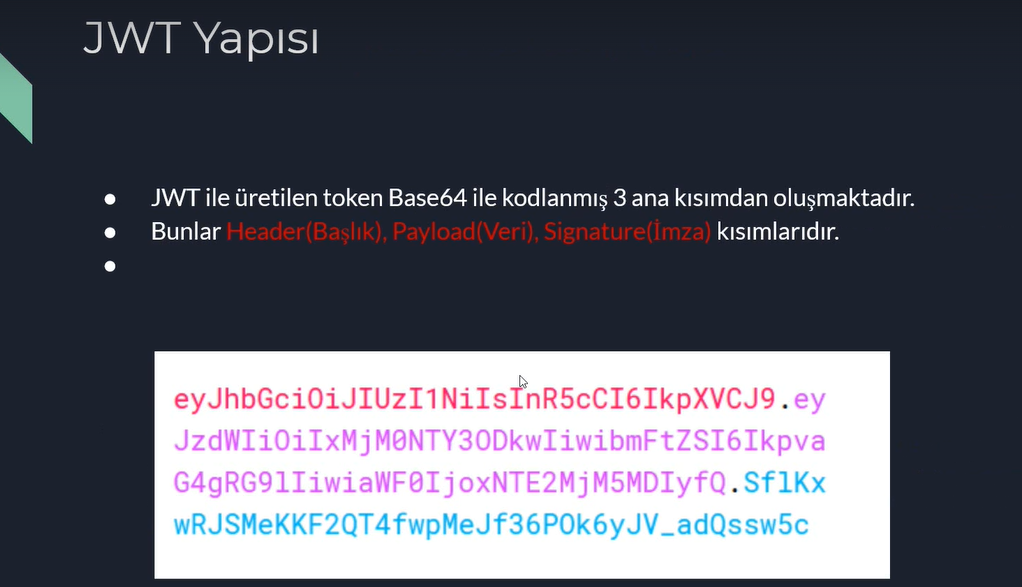
* **Web Socket Nedir ?:**
* Websocket teknolojisi sürekli açık olan bir TCP bağlantısı üzerinden çift yönlü mesaj gönderebilme işlemini yapabilen bir protokoldür.
* http protokolüne uygun olmayan real time web uygulamalarımızdaki karmaşık yapının basitleştirilmesini sağlar.
* WebSocket istemciye bir bağlantı kurup bu bağlantıyı istediği kadar sürdürebilmesini ve veriyi iki taraflı olarak hem almasını hem de göndermesi işlemini gerçekleştirir.
* Socketio her platformda (tarayıcılar ve cihazlar – ios,andorid) çalışan güvenilir ve hızlı olmaya odaklanmış, gerçek zamanlı, çift yönlü ve olay tabanlı bir iletişimi sağlayan yapıdır.
* Nodejs normal olarak request-response olayında isteğe cevap hazırlanıp gönderildikten sonra açılan kanal kapatılıyorken SocketIO ile oluşturulan kanal kapatılmaz ve sürekli olarak dinlenir.
* Eğer socket.io modülü olmasaydı nodejs her seferinde kullanıcıyla TCP connection açacak ve zaman, enerji ve gereksiz fazla internet sarfıyatına yol açacaktı.
* SOCKETIO sayesinde bir çok client kanal kapanması olmadığı için birbiriyle aynı zamanlı olarak veri akışı sağlayabilirler.



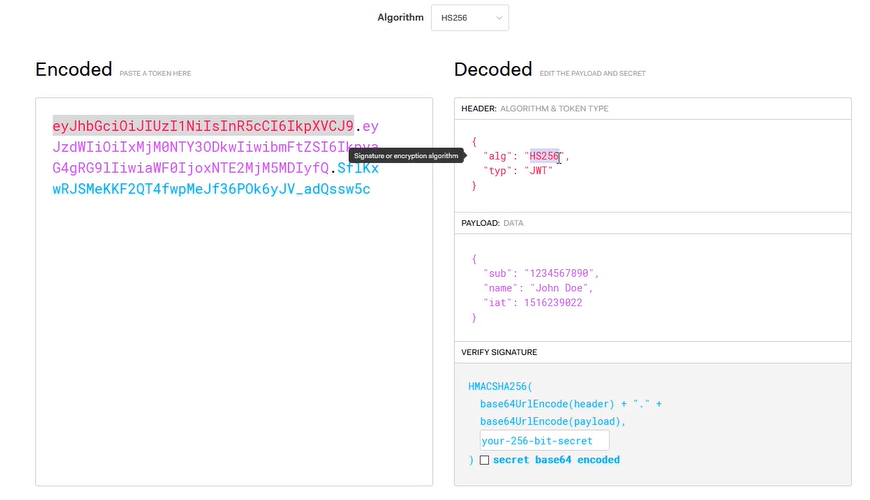
* Npm i socket.io diyerek ilgili paketi projeye kurabiliriz. İlgili işlemler uygulamada vardır.

**JSON WEB TOKEN**

* **Token Nedir?:**
* Web uygulamalarımızı geliştirirken kullanıcı kimliklendirme/yetkilendirme işlemi oldukça önemlidir.
* Uygulamalarımızı yetkisiz kişilerden korumak ve yalnızca yetkili kullanıcıların erişim için çeşitli yöntemler kullanırız.
* Bu çözümlerden biri se token kullanmaktır. JWT token kullanma yöntemlerinden biridir ve çok popülerdir.
* **Authorization ve Authentication;**
* Authentication kimlik doğrulamadır
* Authorization ise sisteme giriş için yetki kontrolüdür.
* Authentication birkez yapıldıktan sona, atılan her istekte bir atuhorization işlemi yapılır. Tabi bu durum geliştirdiğiniz uygulamaya göre değişebilir.
* **JWT Nedir?;**
* Json Web Token tarafların birbirleri arasındaki veri alışverişini ve bunun doğrulamasını sağlayan JSON tabanlı RFC 7519’de tanımlanmış açık bir standarttır.
* Örneğin bir sunucu kullanıcının yönetici ayrıcalıklarına sahip olduğunu belirten bir anahtar token oluşturabilir ve bunu kullanıcıya gönderebilir. Kullanıcı daha sonra bu anahtar ile kendisine tanımlanmış olan yönetici yetkisini bir istemcide kullanabilir ve bütün taraflar tarafından yetkisi doğrulanabilir.

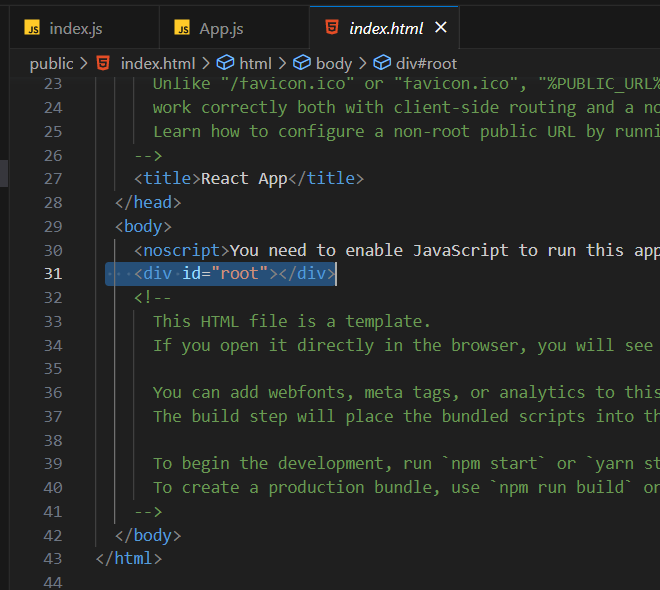


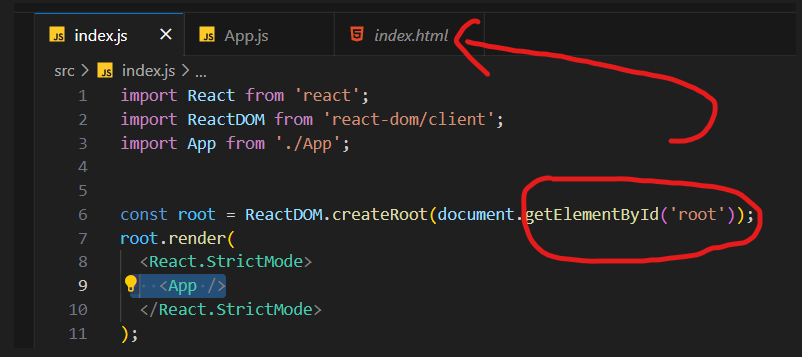
* **JWT AVANTAJLARI**
* Stateless çalışır. Yani kontrol edecek bir Seesion bulunmamaktadır. Bilgiler ve son geçerlilik tarihi ne sunucuda ne client tarafında tutulur. Token içerisinde gerekli bilgiler tutulur.
* Portable çalışır. Birden çok backend ile çalışabilir, yalnızca 2 taraf arasında kullanılmak zorunda değildir. Bu durum hem web uygulamamız hem de mobil uygulamanız aynı web servisi kullandığında oldukça önemlidir.
* JSON formatını kullanır
* Doğrulama işlemi diğer Authorization metodlarına göre daha hızlıdır. Doğrulama işlemi için veri tabanı ile bağlantı kurmaya gerek kalmaz.

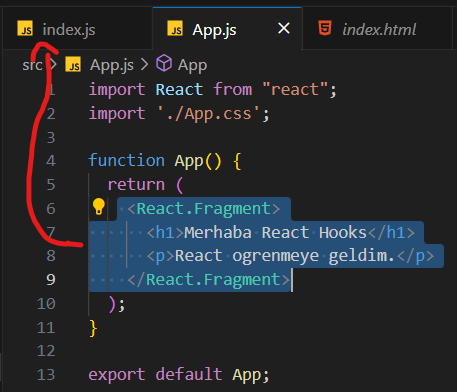


**REACT**

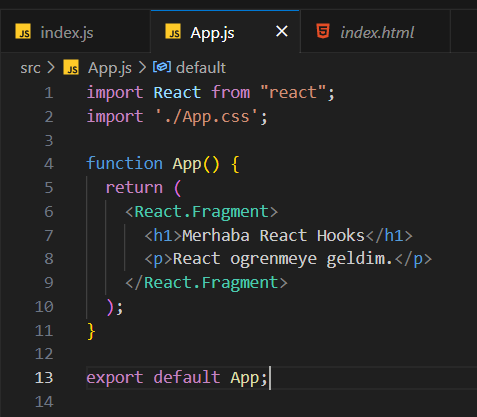
* Yeni bir react projesi başlatmak için ilgili konumdaki cmdyi açıp npx create-react-app projeismi şeklinde kod yazıp paketlerin yüklenmesini bekliyoruz.
* Uygulamayı başlatmak için ise projenin konumundaki cmdye yarn start deriz. Veya code terminalinden npm start dedikten sonra <http://localhost:3000/> adresine gidebiliriz.
* Node modules klasörü içerisinde gerekli olan paketler yer alıyor.
* Public klasöründeki index.html sayfasına bizim react kodlarımız render edilecek. Oradaki root idli div içerisine render edilecek. Bunun ayarı src klasöründeki index.js içerisindedir. document.getElementById('root') dendiği için bu durum böyle. Root.Render() metotuylada render ediliyor. Buradanın içerisinde <App /> dosyamız ana dosya olarak belirlenmiş. App.js dosyasının içine geldiğim zamanda bir fonksiyon yer alıyor ve buranın içerisine yazdığımız şey index.html içerisine index.js vasıtasıyla render ediliyor. Burda bu fonksiyonun içindeki dive App classı verildiği ve App.css içerisinde bu classa text-align: center verildiği için buraya gelen içerik ortalanacaktır.



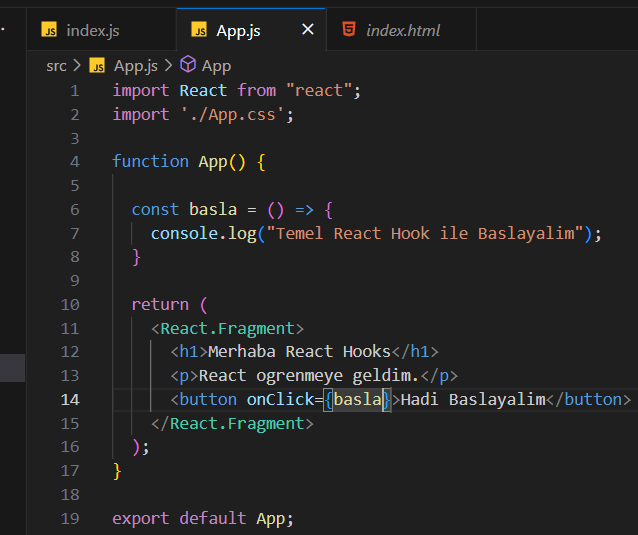




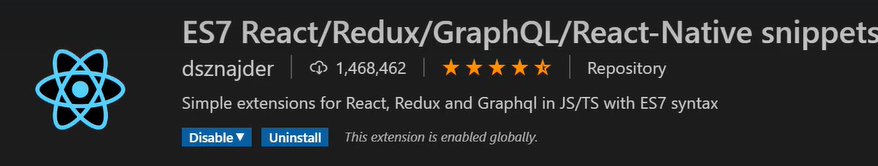
* FONKSİYONEL COMPONENTİMİZ APP.JS İÇEİRSİNDEKİ APP FONKSİYONUDUR. RESİMLERDE OLDUĞU GİBİ APP.JS. İÇİNDEKİ FONKSİYON İÇİNDE YAZILANLAR İNDEX.JS DOSYASINDAKİ APP KISMINA GÖNDERİLİR. BUNU SAĞLAYAN İŞE AŞAĞIDA APP İN EXPORT YAPILMASIDIR. DAHA SONRA İNDEX.JS TE İNDEX HTML SAYFASINDA BULUNAN ROOT İD Lİ DİVE ERİŞİLEREK APP ETİKETİ KISMINA GELEN BU ELEMENTLER O DİV İÇERİSİNE RENDER EDİLİR YERLEŞTİRİLİR YANİ.
* Src klasörüne burada bizim kullandığımız react klasörlerimiz dosyalarımız yer alacak. Biz bir klasör ya da dosya oluşturmak istediğimizde mutlaka src klasörü içerisinde oluşturmalıyız aksi halde hata alırız.
* Gitignore git işlemleri için, gitin değişiklik olarak algılamasını istemediğimiz dosyaları buraya gireriz.
* Package.json bizim temel bilgilerimizi barındıran json tipinde dosyamızdır. Yarn paket indirme managerini kullanaarak yeni bir paket indirdiğimizde bu dosya içindeki dependencies kısmına bu paket gelecek.
* App.js fonksiyonu içerisinde React bir jsx kullandığı için yani javascript içerisine html yazabildiğimiz için burada bir element oluşturmamız lazım. Bütün elementler ana bir element içinde olmalı. Buanda React.Fragment diyebiliriz. Eğer burada React.Fragment kullanamzsak bu fonksiyon içinde sadece 1 element kullanabilirdik bir tane h1 kullanmak gibi ama ana kapsayıcı olan fragmenti kullandığımızda onun içerisinde birden çok element kullanabiliyoruz. Burada boş bir element aç kapa işareti de kullanabiliriz fragment yerine. Yani <> </> içerisine elementlerimizi de yazabilirdik.



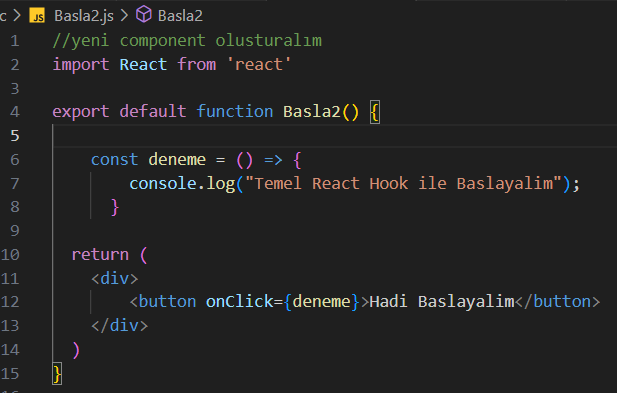
* Eğer kendim bir fonskyion yazmak istersem app.js içindeki fonksiyonel komponentimiz olan App fonksiyonu içerisine return üstüne fonksiyon yazabiliriz. Daha sonra butonun onclickinde bu fonksiyonu kullanabiliriz.



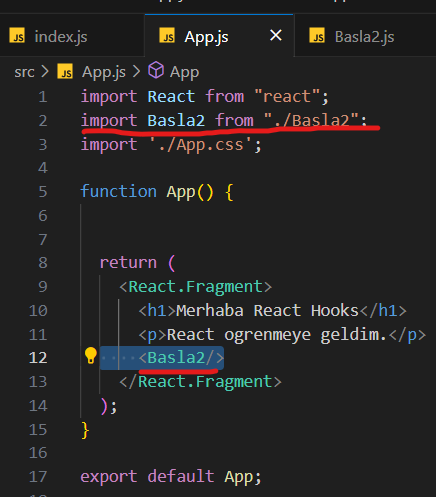
* Şimdi bu yaptığımız işlemi ayrı bir component içerisinde tanımlayalım ve burada o componenti çağırarak kullanalaım. Yeni dosyayı src içerisinde oluşturmayı unutmayalım. Uzun uzun yazmaktansa extension kullanıcaz aşağıdaki extensionu.



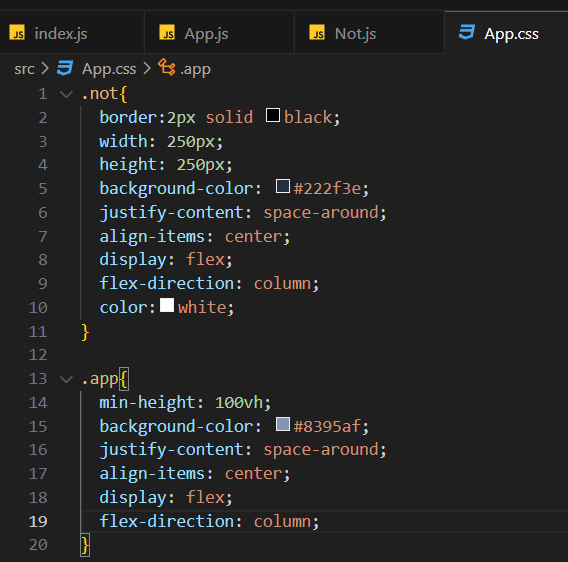
* Ardından rfc kısayolunu kullanarak direkt bir react component oluşturabiliyorum.
* Yeni component oluşturup bu componenti kullanma işlemini gerçekleştirelim haydi.

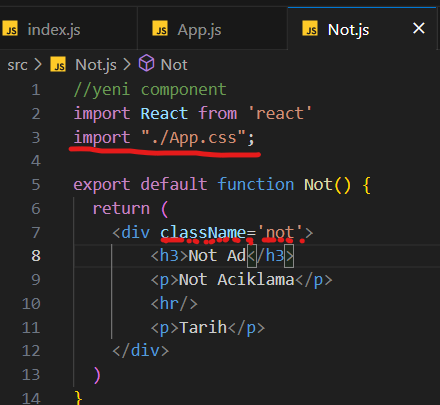


COMPONENTİ ÇAĞIRMAK

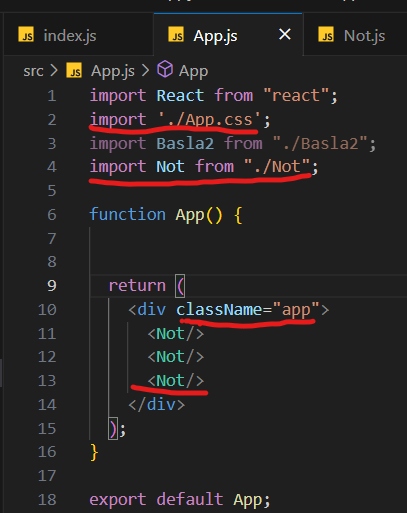


* Style yani css kullanımı için App.css içerisine css kodlarımızı yazdık daha sonra yeni bir component oluşturduk component divine class vermek için className= attribute unu kullanıyoruz. Ayrıca yeni componentimizde css e ulaşmak için css dosyamızı import etmemiz gerekiyor.
* Haydi reactta style kullanımı yapalım.

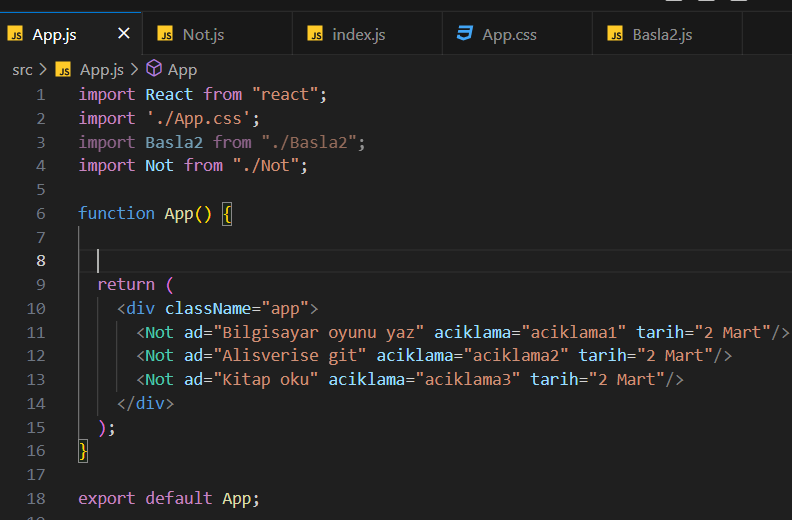


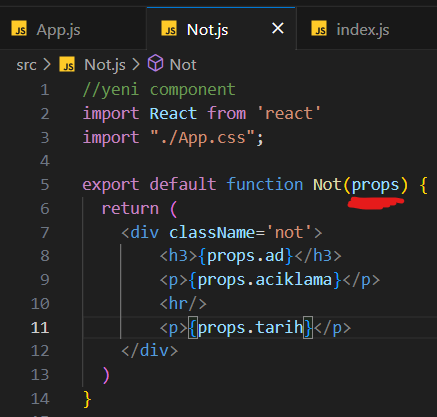


CSS DOSYALARIMIZ VE YENİ COMPONENTİMİZ HAZIR ŞİMDİ BU COMPONENTİ ÇAĞIRIP KULLANABİLİRİZ. AYRICA BU COMPONENTİ BİRDEN FAZLA KEZ ÇAĞIRARAK BİRDEN FAZLA KULLANABİLİRİZ.

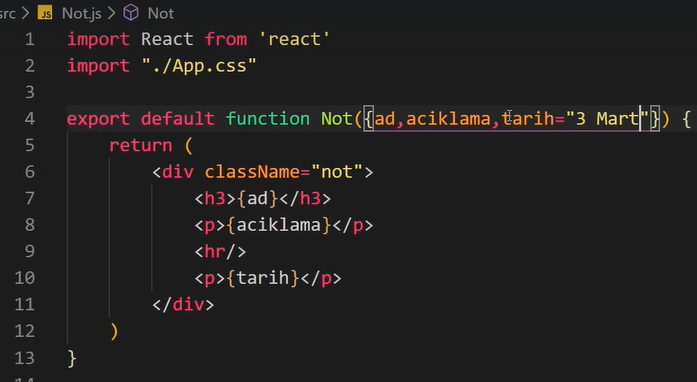


* Props kullanımını görücez ve propslar sayesinde kullandığımız Not componenti içerisindeki alanları değiştirebilicez. Propları kullanarak komponentlerime veri gönderebiliyorum. App.js de App komponenti içerisinde Not komponentimi çağırdığımız yerde propsları tanımlayabiliyoruz. Propsları kullanarak componentlere veri gönderiyoruz. Propları App componentiden göndertikten sonra Not componentinde bu propslara erişmek için function un parametresine props yazıyorum veya bu parametreye direkt propsun adını da süslü parantez içinde yazabilirim {ad} diyerek ve ve artık süslü parantez içerisinde {props.ad} diyerek gönderdiğim ad propsuna erişebiliyorum. İkinci kullanımı yaparsak parametreye {propadı} şeklinde olanı bu sefere {ad} diyerek direkt propu kullanabiliriz. Bir componente birden fazla props gönderebiliriz yani bir component içerisinde birden fazla props kullanabiliyoruz.

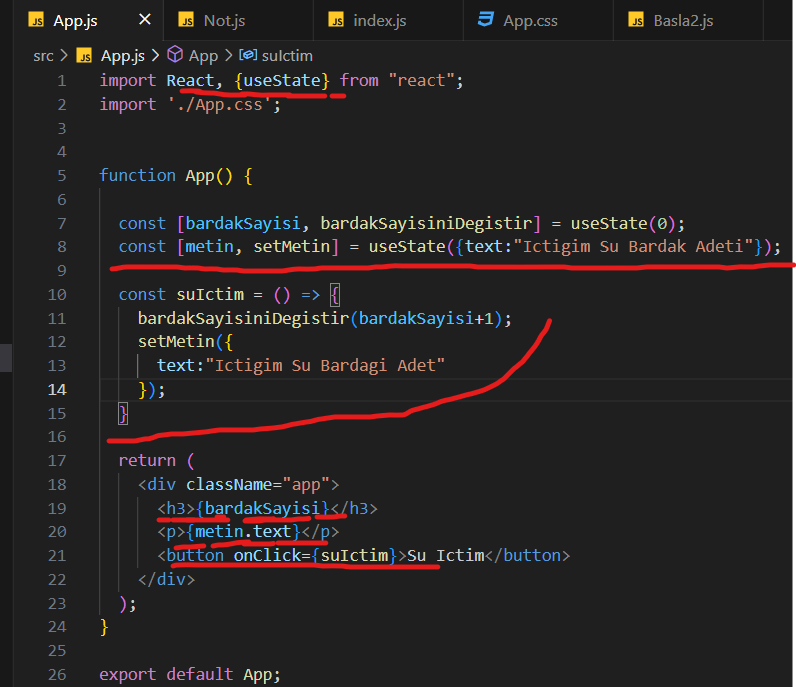




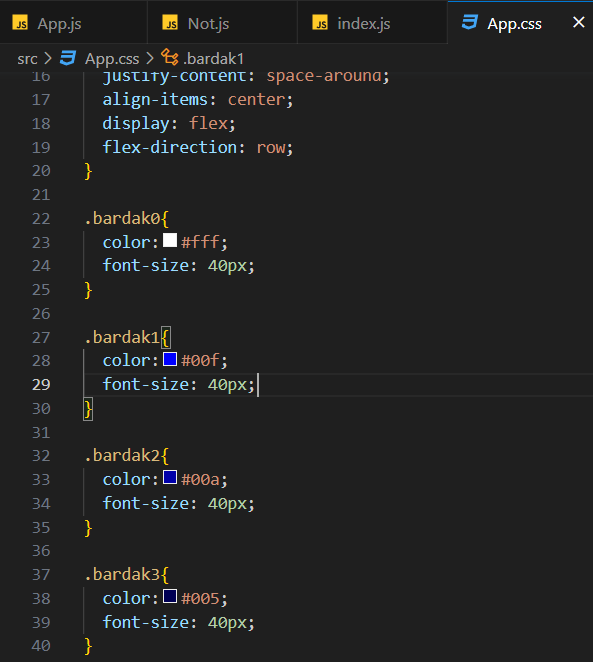
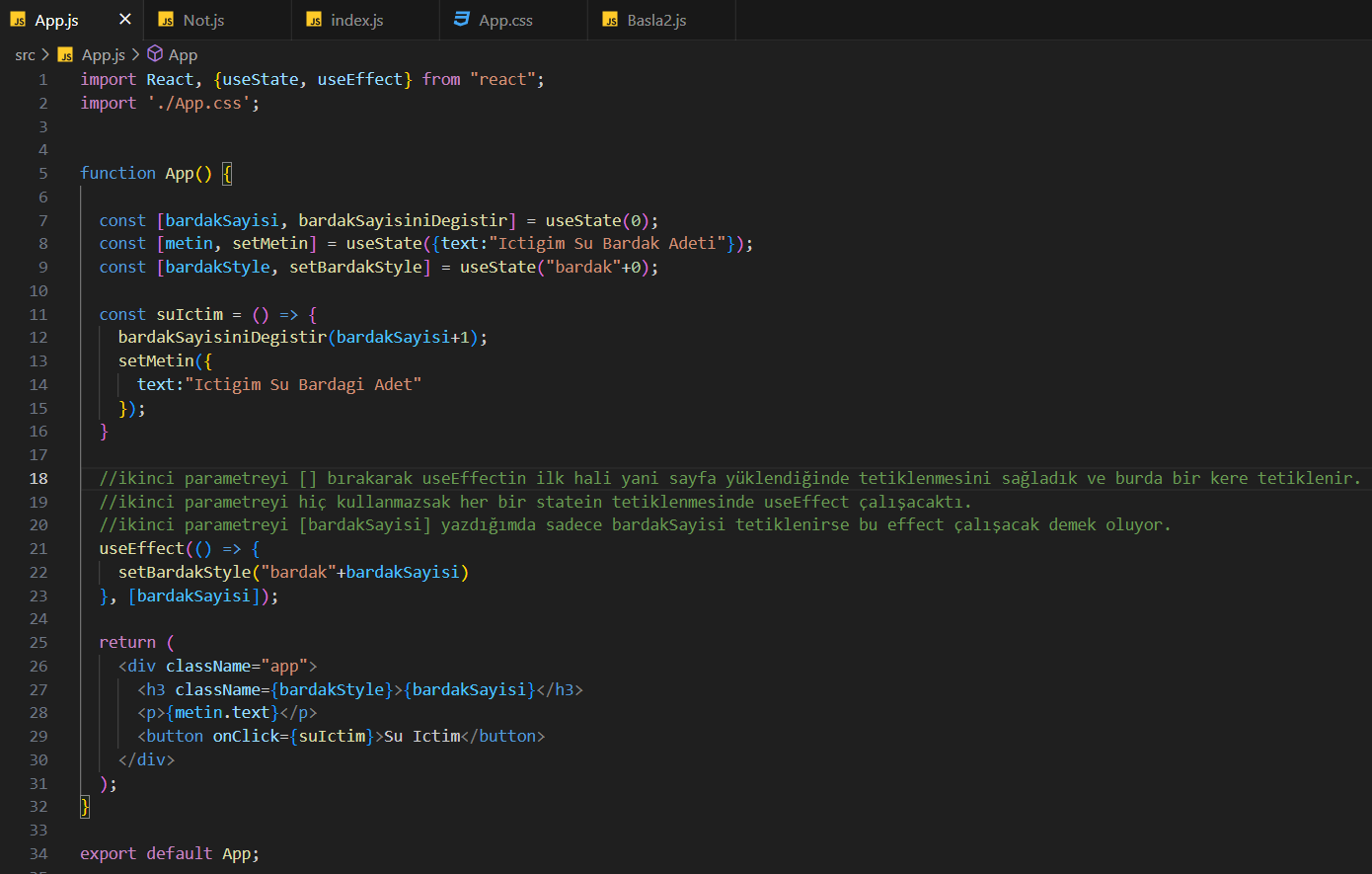
VEYA İKİNCİ YOLU KULLANARAK DİYELİM Kİ TARİH PROPSU GÖNDERİLMEDİ BU PROPSA VARSAYILAN DEĞER DE ATAYABİLİYORUZ.



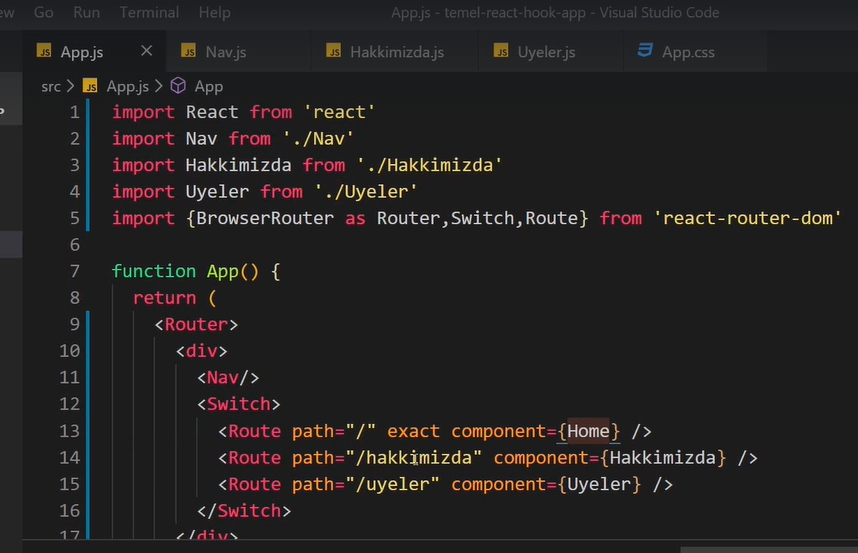
* useState ile değişkenlerimi bir state yapısı içerisinde kullanabilirim. Bu propslarımı bir state içeirisnde kullanarak bu bilgilere daha kolay bir şekilde erişebilirim. useState kullabilmem için app.js sayfamda useState modülünü önce import etmem gerekiyor. Ardından useState tanımlamak için const [] = useState() kullanabiliriz. useStatei kullanabilmem için değişkenimin ismi olmalı ve bu useState i güncelleyebileceğim bir metot olmalı. useStatein parametesi içinde ise bunun ilk değerini verebiliyoruz.
* USESTATE KULLANIMI GERÇEKLEŞTİRELİM VE BİR BUTONUMUZ OLSUN BUTONA TIKLADIĞIMIZDA BU USESTATE VERİLERİNİ GÜNCELLEME İŞLEMİ YAPMIŞ OLALIM.

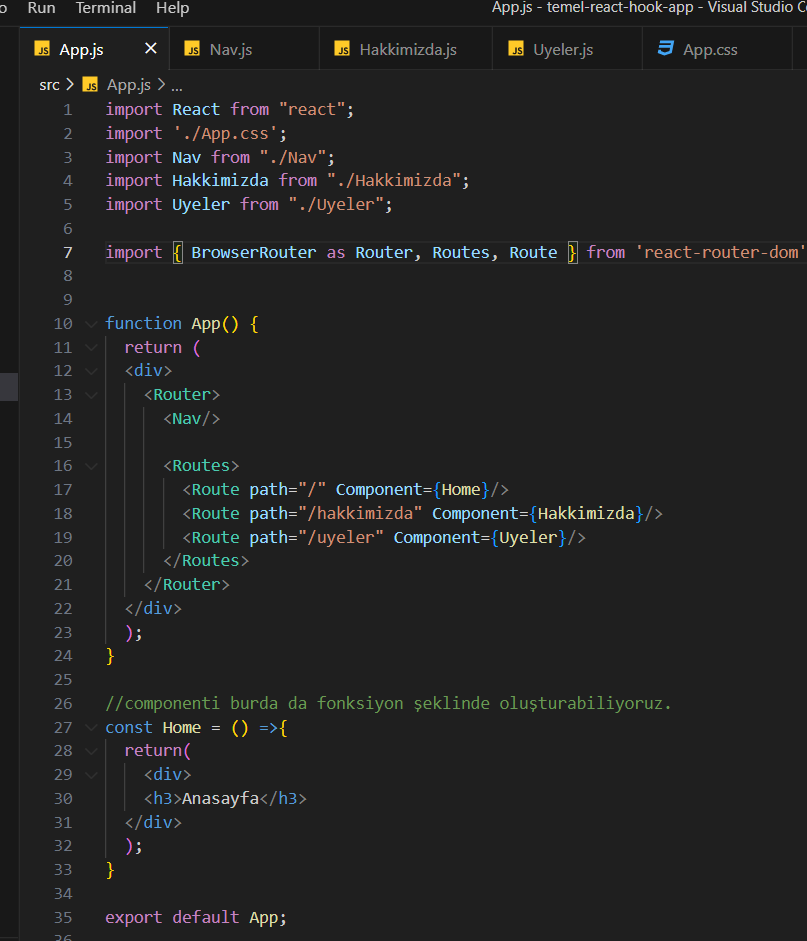


* useEffect kullanmak için aynı useStatete olduğu gibi import işlemi yapmamız gerekiyor. 3 Farklı yolla useEffecti kullanabiliyoruz. 1 sayfa ilk yüklendiği zaman tetiklenen bir metottur. 2. Olarak stateler değiştiğinde tetiklenen bir metottur. 3. Olarakta belirli bir state kontrol edilebilir o tetiklendiği zaman çalışan tetiklenen bir metottur. Her butona tıkladnığında tanımladığımız cssleri alan bir örnek yapalım

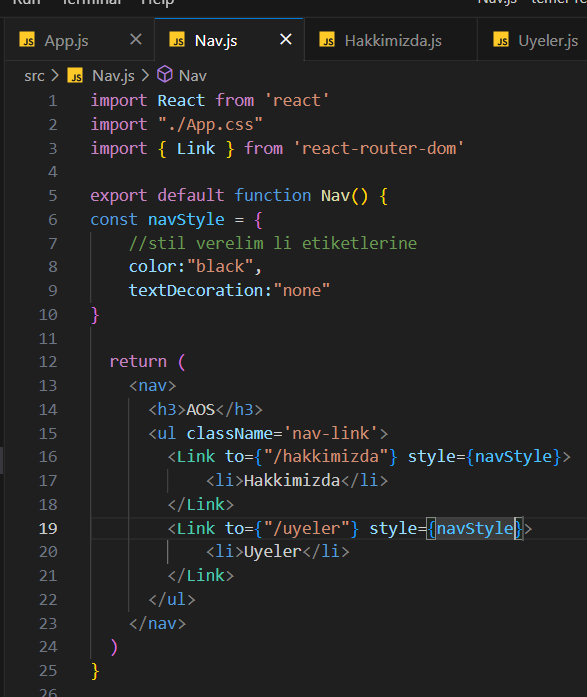
,

* Artık temel react router işlemlerine geçebiliriz. Navbarda bulunan hakkimizda a etiketine bastiğimizde hakkimizda componenti uyeler a etkiketine bastığımızda uyeler compoennti karşıma gelsin istiyorum işte bu işlemi router yardımıyla gerçekleştireceğiz. Öncelikle yarn add react-router-dom paketine ihtiyaç var. Artık yönlenme için bütün gerekli olan componentlere sahip oluruz. Bunlar BrowserRouter-Switch-Route componentleridir. Bizim yönlenme için ana componentimiz app.js componentidir. Yönlendirme componentlerini app.js dosyasına import etmek gerekiyor. BrowserRouteri kullanarak tüm componentlerimi onun içerisine atıp yönlenme işlemlerimi burda yaparım. Yönleneck olan componenteleri burada Route componenti ile çağırabiliriz veya yönlemeyecek direkt gelecek componentleri ise ellemiyoruz mesela nav componenti buna örnektir tüm sayfalarda olacağı için öyle kalacak. Açağıdaki örnekte Nav componenti direkt olarak her sayfada çalışır Routes componenti içine yazılan Route componentleri ise yönlendirme yapar urlye <http://localhost:3000/hakkimizda> girildiğinde mesela Hakkimizda componenti çalıştıralacaktır mantık bu şekilde işler. Burda BrowserRouter componenti uzun bir isim olduğu için as diyerek istediğimiz ad ile aynı paketi çağırabiliyoruz. Switch componenti ile ise doğru bir yönlendirme sağlayabiliyoruz mesela / tüm routelarda olacağı için componentler iç içe girebilirler bu noktoda switch yardımımıza koşuyor. Exact diyerek tamamıyla doğru olduğunda bu pathi çalıştır diyebiliyoruz. Ancak bu swtich componenti React Router v6 versiyonnuda kaldırıldı onun yerine <Routes> bileşeni kullanılmaktadır.
* ESKİ KULLANIM BU ŞEKİLDEYDİ

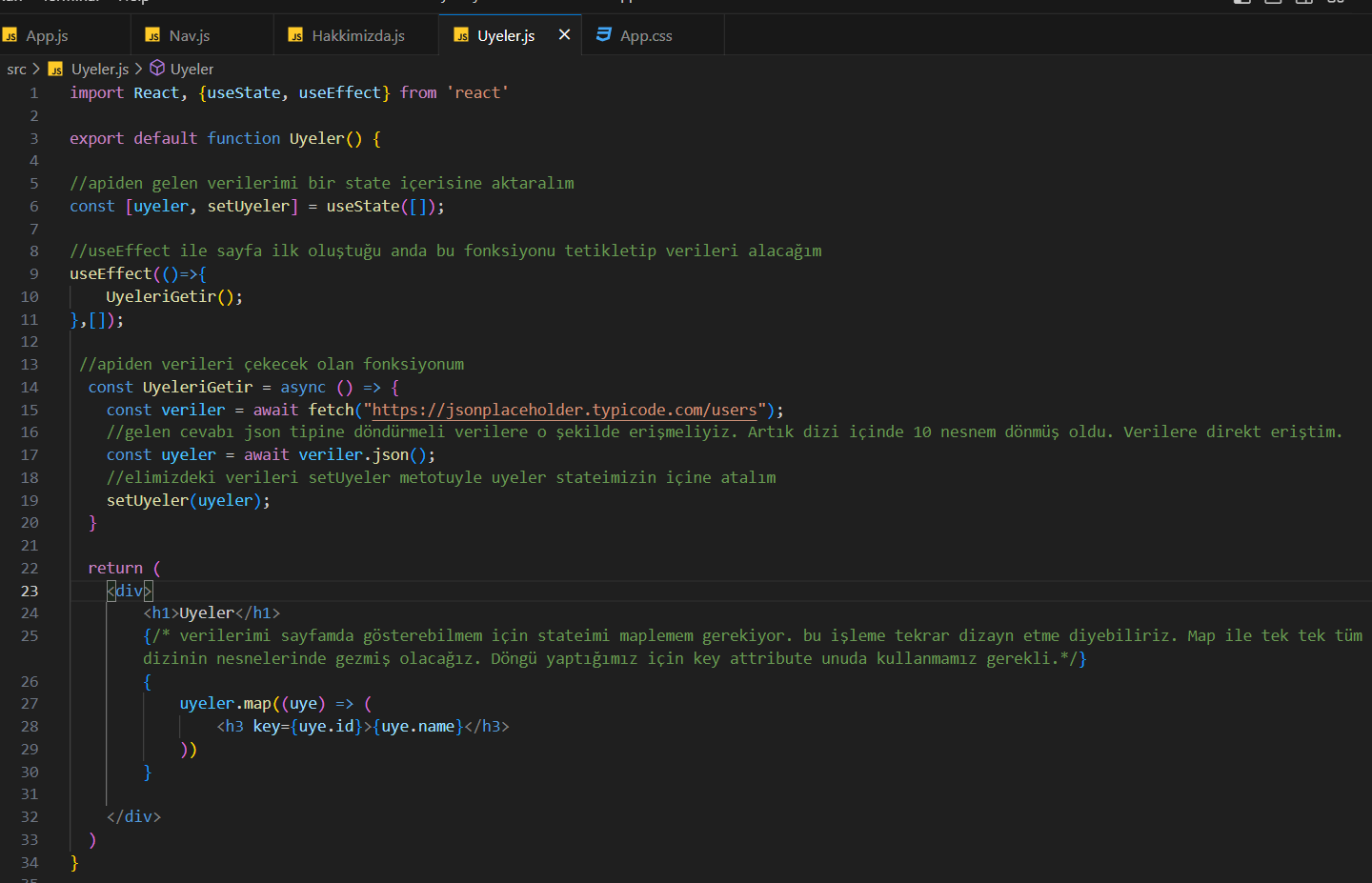




* Link componenti ile yönlenme işlemi gerçekleştirebiliyoruz. Navbardaki menülere tıkladığımızda o componentlere geçiş yapabilmemiz gerekiyor bunun için link componentini kullanıyoruz. Öncelikle ilgili componente mesela nav componentine Link componentini import etmem gerekiyor. Daha sonra bu lilerimi link componenti kapsamının içine alıyorum. Link içerisinde to attributeunu kullanıyoruz to içine route içerisinde ayarladığımı path attributeu bilgisini giriyoruz. Bunu yaptığımızda biz a etiketi kullanmış olmasak bile li etiketlerimiz a etiketine dönüşüyor. Href bilgileri ise to ya verdiğimiz path bilgisi olmuş oluyor.



* Harici bir apiden verileri çekicez. Bu veriler üyelerimizi oluşturacak ve herhangi bir üyeye tıkladığımızda o üye ile ilgili ayrıntılı bilgi ediniceğimiz ayrı bir sayfaya yönlenicez. Tabi bu sayfa bir parametre yöntemiyle oluşacak parametreleri yönlendirme yaparken nasıl kullanacağımızı öğrenicez. Öncelikle apiden veri çekmek için kullancağımız useEffect ve useStateleri import ederiz. Verileri çekmek için asekron bir fonksiyon oluşturmalıyız. Fetch metotundan yararlanarak bu apiye istek atıp verileri cevap olarak alacağız. Bu fonkisyonu useEffect ile sayfa ilk oluştuğu anda 1 defa tetikleteceğiz.
* COMPONENTİMİZDE APİDEN FETCH İLE VERİ ÇEKEN BİR FONKSİYON YAZALIM USEFFECT KULLANARAK BU FONKSİYONUN COMPONENT ÇALIŞTIĞINDA 1 KERE ÇALIŞMASINI SAĞLATALIM VE FONKSİYON İÇİNDE USETATE KULLNARAK VERİLERİMİZİ STATE İÇİNE ATALIM VE MAP İLE VERİLERDE DOLAŞIP HER VERİYİ BİR ETİKET İÇİNE YAZDIRABİLİRİZ.



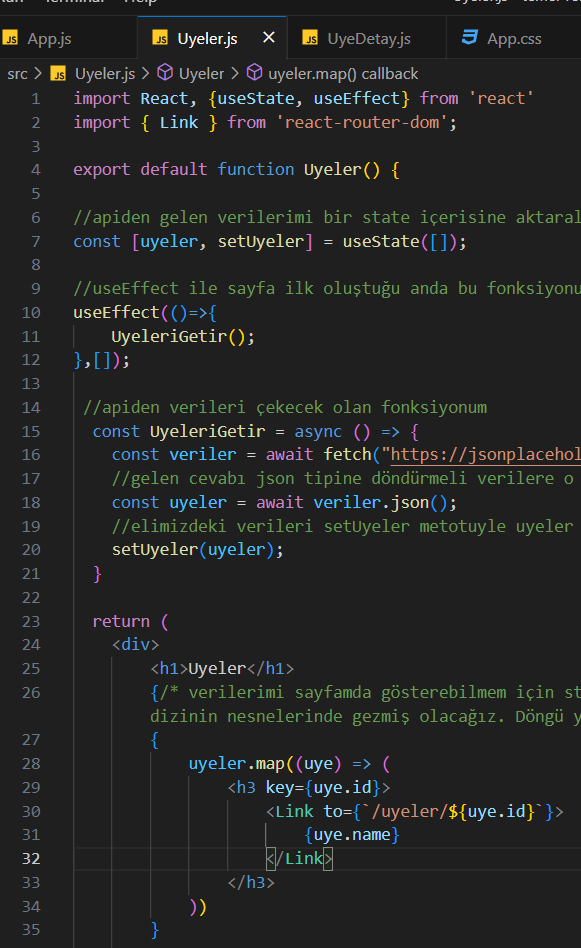
1. **Neden map() kullanıyoruz?**: React'ta listeleri veya dizileri render etmek için genellikle **map()** fonksiyonunu kullanırız. **map()** fonksiyonu, bir dizi üzerinde döngü yaparak her öğe için işlem yapmamızı sağlar. Bu durumda, **uyeler** adlı bir diziye sahip olduğunuzu varsayalım. Her bir **uye** öğesi için **<h3>** etiketini oluşturmak istediğinizde, **map()** fonksiyonunu kullanarak diziyi döngüleyebilir ve her öğe için ilgili JSX kodunu üretebilirsiniz.
2. **Neden key attribute'unu kullanıyoruz?**: React, listelerdeki her bir öğenin benzersiz bir kimliğe sahip olmasını ister. Bu, React'in sanal DOM'da verimli bir şekilde değişiklikleri takip etmesini sağlar. **map()** fonksiyonunu kullandığınızda, her oluşturduğunuz öğeye benzersiz bir **key** attribute eklemeniz önerilir. Bu, React'in her bir öğeyi benzersiz bir şekilde tanımlayabilmesini sağlar.

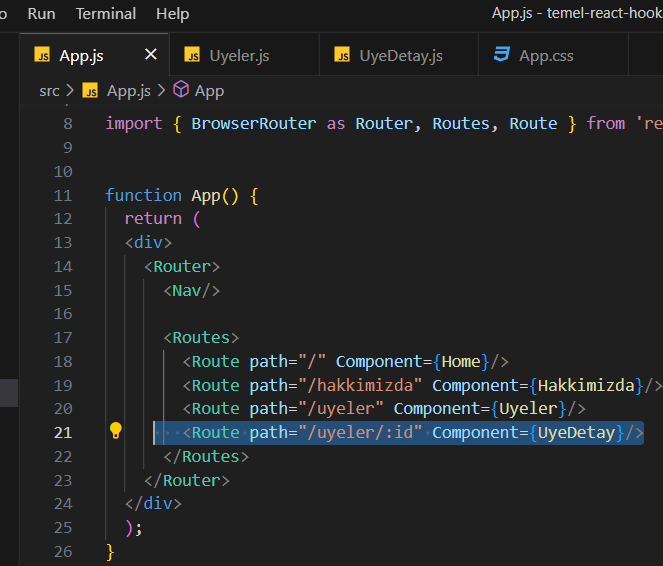
Genellikle veri öğeleri bir ID'ye sahip olduğunda, bu ID'yi **key** attribute olarak kullanmak iyi bir uygulamadır. Bu, React'in her öğeyi etkili bir şekilde takip etmesini ve gerektiğinde optimize etmesini sağlar. Ayrıca, **key** attribute'i eşsiz olmalı ve bir liste içinde benzersiz olmalıdır.

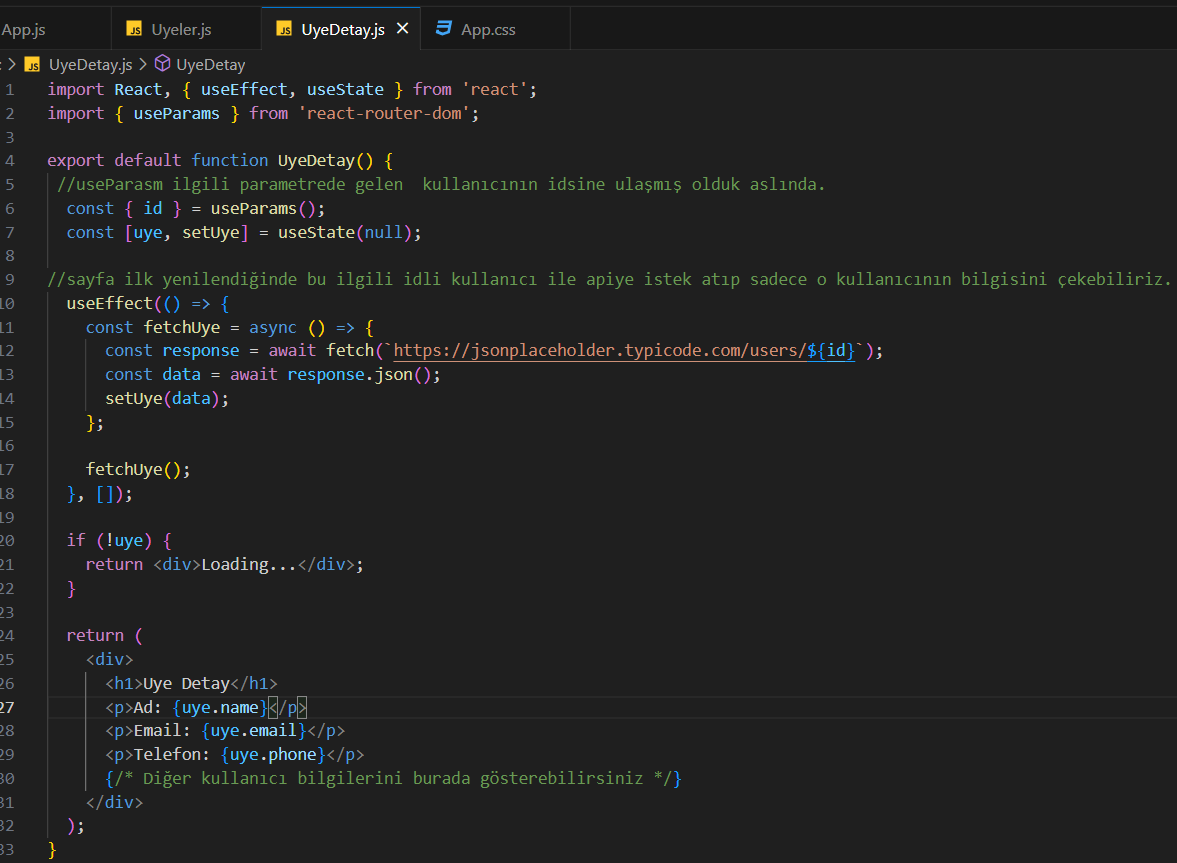
Örneğin, yukarıdaki kodda her bir **uye** öğesi için **id** alanını **key** attribute olarak kullandık: **<h3 key={uye.id}>{uye.name}</h3>**. Bu, her bir öğenin benzersiz bir şekilde tanımlandığından emin olur.

Bu şekilde, **map()** fonksiyonunu kullanarak bir dizi üzerinde döngü yapabilir ve her bir öğe için JSX kodunu üretebilirsiniz. Ayrıca, her öğeye benzersiz bir **key** attribute ekleyerek React'in performansını optimize etmesini sağlayabilirsiniz.

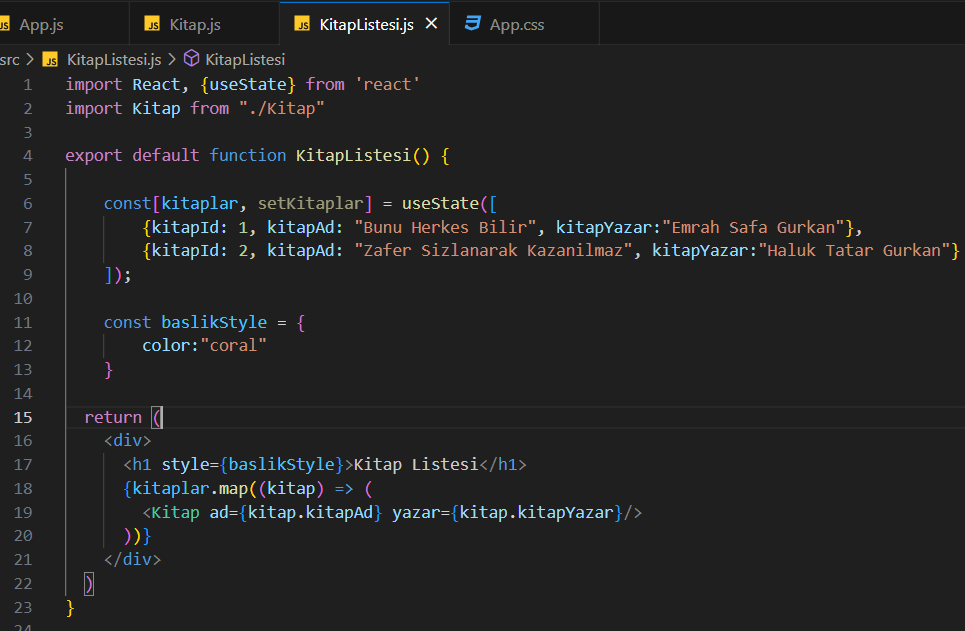
* Artık her bir üyenin ismine tıklayarak o üye için detay sayfasına gidebiliriz. Yani parametreli yönlendirme işlemi yapıcaz. Yani match Props oluşturucaz. Link komponentinden faydalnıcaz yine bunun için. Uyeler componentimizde Link componentini import edelim. Uye isimlerinin oluştuğu noktada Link componentinin içine alarak kullanalım. Linkin to attributunda url bilgisinden sonra parametreyi verebiliriz yani `/uyeler/${uye.id}` yapabiliriz. Daha sonra üye detay componentinin parametresinde {match} propsundan yararlanabiliriz. Match kendiliğinden oluşan bir propstur. Matchi atatıktan sonra useEffect ile sayfa ilk yüklendiğinde matchin içerisindeki bilgileri yakalayabiliriz. Daha sonrasında bu uyedetay sayfasını route olarak belirtmek için app.js componentinde yeni bir route oluşturuyoruz bunun pathine parametre gönderceğimizi belirtmek için /uyeler/:id oalrak belirtebiliyoruz. Ancak bu match propsu v6 sürümüyle birliktte reacttan kalktı. Onun yerine useParams kullanılmaktadır.

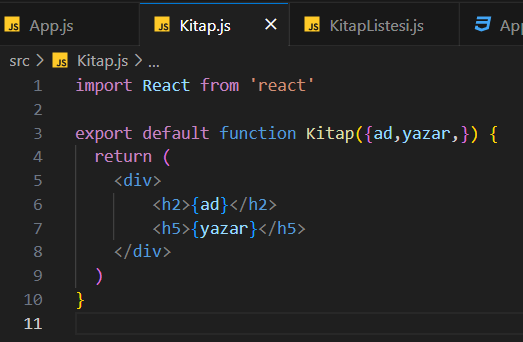


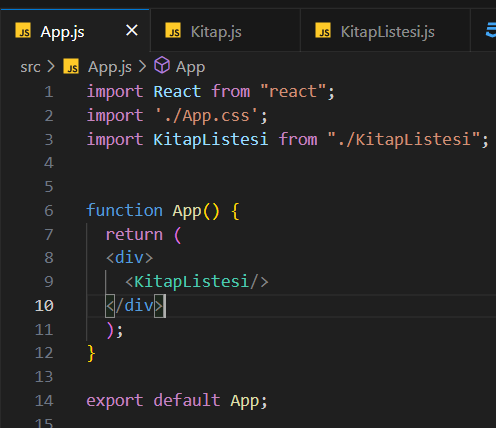




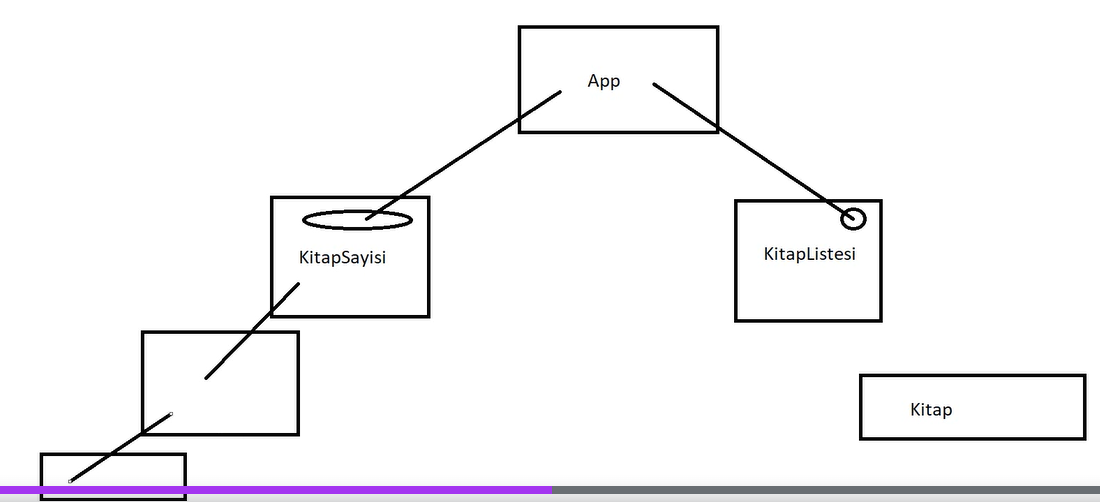
* Artık ContextApi işlemlerini gerçekleştirebiliriz. Öncelikle props ile veri aktarmayı göstericez sonrasında bu props ile veri aktarınca ortaya çıkan karışıklığa değinip contextapi ile verileri nasıl aktaracağımızı belirteceğiz. Kitap Listesinden veri dizimi state olarak oluşturduk ve bu verilerin hepsini map işlemiyle gezerek yani stateimizin içindeki verileri gezdik ve bu stateimizin verilerini props olarak Kitap componentine gönderdik. Kitap componentinde ise bu verilere props olarak aldık eriştik. Sonrasında ise app.js componentimde bu Kitaplistesi componentimi çektim. Burda propsla sadece stateleri değil metotlarımızda aktarabiliriz unutmayalım. İşte metot aktarımı yaparken ortalık karışacak.



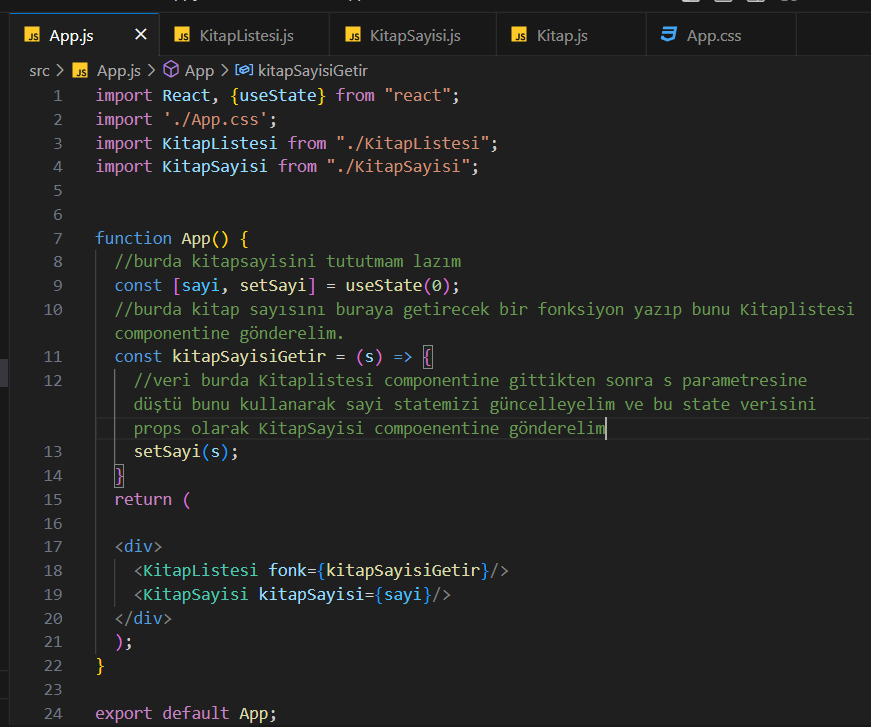


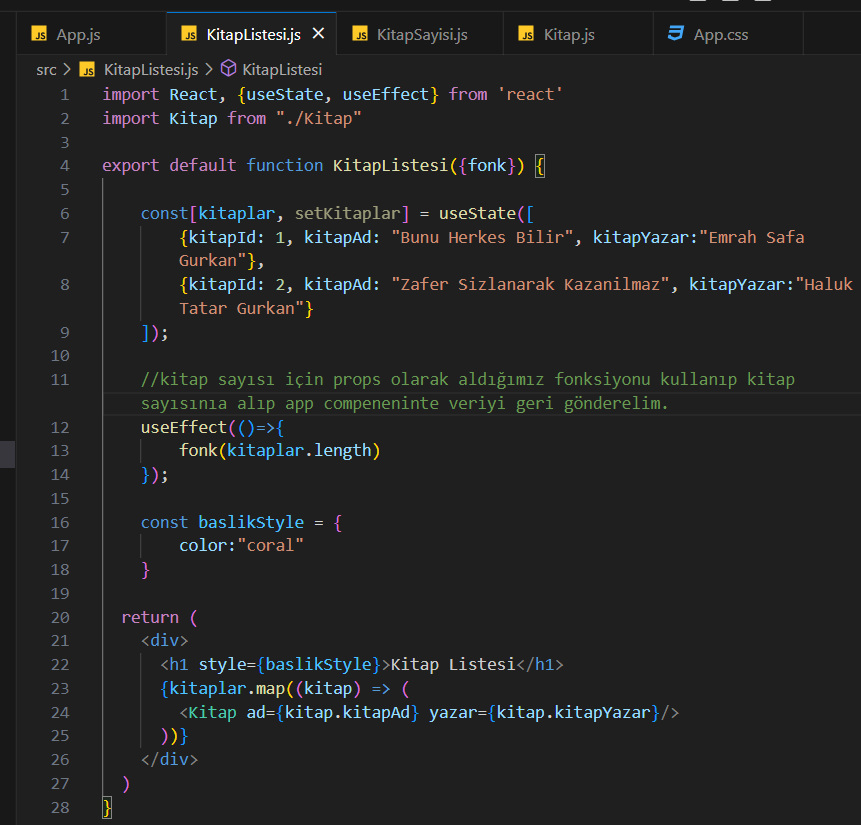


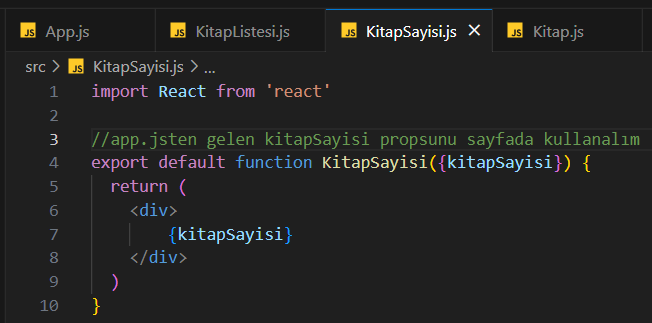
* Şimdi KitapSayisi diye bir componentim daha olsun. Ve bu componenti App.js te KitapListesi componenti altında gösterelim hemen. Bu componente props olarak kitaplistesi componentindeki kitaplar stateti içindeki elemanların sayısını göndericem. Burada kitapsayisi ile kitaplistesi componentlerinin bir bağlantısı olması lazım bunuda app componenti aracılığıyla yaparız. Bu bağlantı için önce kitapsayisinin KitapListesinde olması ordan App componentine aktarılması lazım ordan da KitapSayisi compoenntine aktarılması lazım. Bunun altında da başka bir component onun altında da başka bir component olsaydı bu kitapsayisi bilgisi aşağıya doğru gidecekti 50 tane iç içe componentimiz olsaydı her birinde dolanması lazımdı.

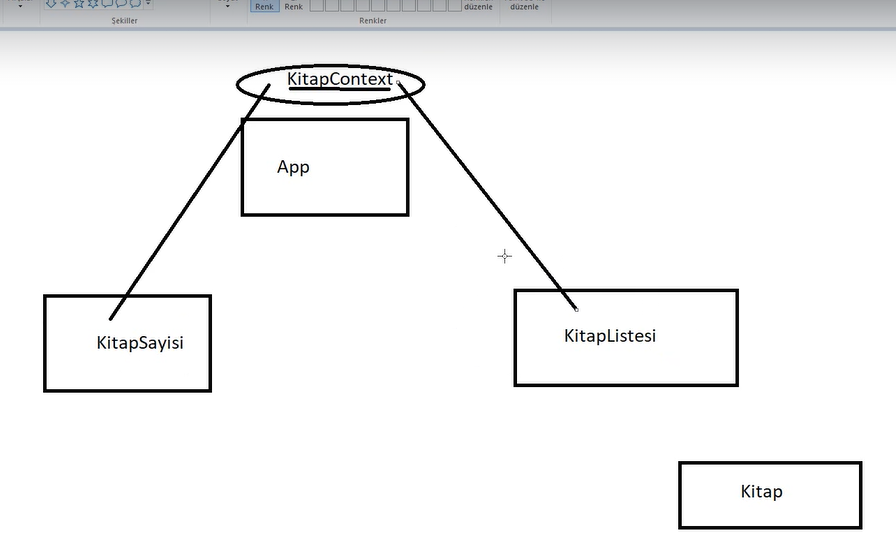


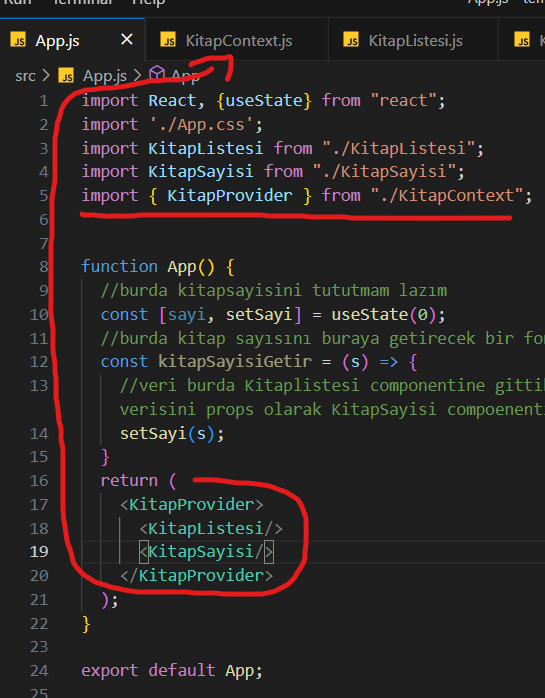
* App componentte bir fonksiyon oluşturarak bu fonksiyonu kitaplistesi componentine bir props olarak gönderdik bunun sayesinde kitapsayısı verisini aldık bu sayısı appcompoenntte kendi statei içerisinde güncelleyip ordan da props ile kitapsayisi compoentine bu veriyi gönderdik. Bu bir karmaşıklığa yol açıyor. Componentler çoğaldıkça ve iç içe component yapısı dallandıkça karmaşıklık çokça artacaktır.

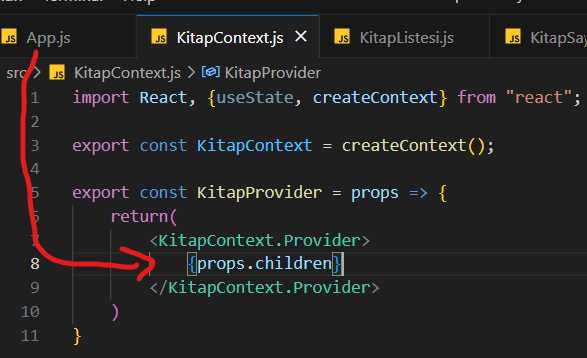




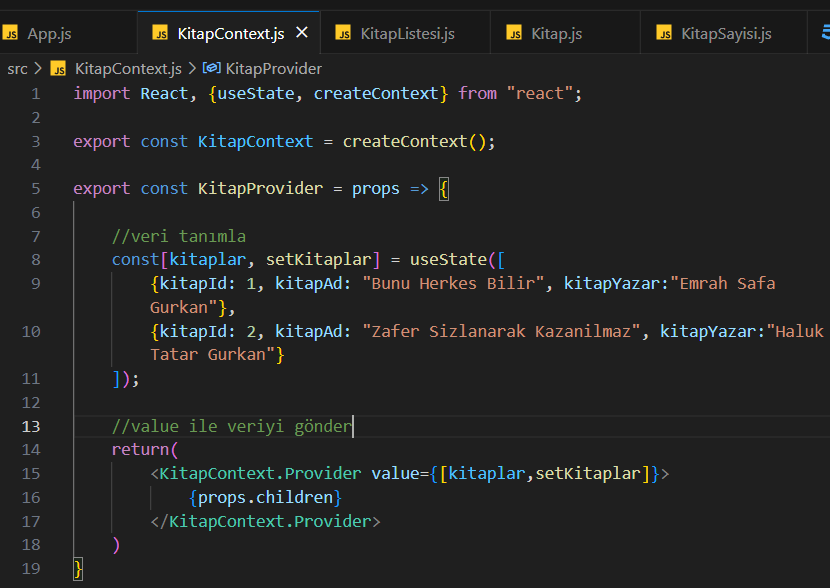


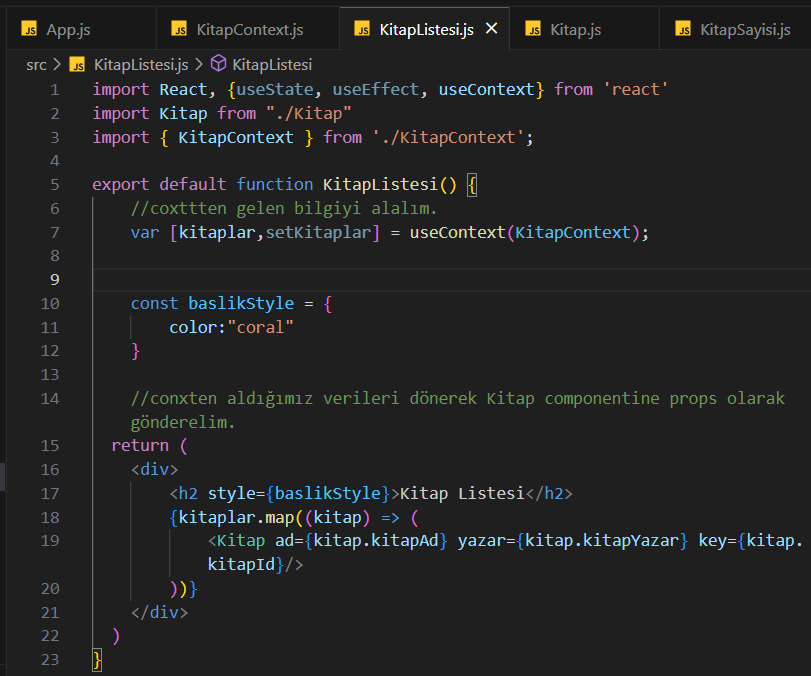
* ConextAPi ile bu karmaşıklığı ortadan kaldıracağız state yönetimini sağlayacağız. Bir tane context oluşacak. Bu içerisindeki provider sayesinde state bilgisini aktaracak. Artık stateimizi kitaplistesi compenentinde değil contextin içerisinde tutacağız. Context ise isteyene bu kitap bilgilerini gönderecek. Örneğin kitap sayısı bilgsiini KitapSayısı componenti istedi o zaman direkt bu bilgi contexten KitapSayisi compoenntine gönderilecek. Kitap listesi KitapListesi componenti tarafından istendiğinde direkt olarak context bu bilgiyi KitapListesi compoennetine gönderecek. Yani App Componenti içerisinde bir provider ayarlıcaz bu providerin içersinde de bu ccontexti ve içerisindeki stateleri ya da metotları kullanan bir provider sayesinde fonksiyonlarımız olacak. 
* Src içerisine contextimizi oluştururuz. createContext i import edip bunu kullanarak context oluşturuyoruz. Providerimizin da olması lazım burda çünkü provider dağıtan bir nesne görevinde verileri. Provider etiketleri içerisinde contexti kullanacak olan componentleri yerleştiriyor olmamız lazım. Buraya {props.childen} yazarsak ve bu KitapProvider i app component içinde kullanırsak orda tanımladığımız componentler buraya otomatik olarak düşecektir.

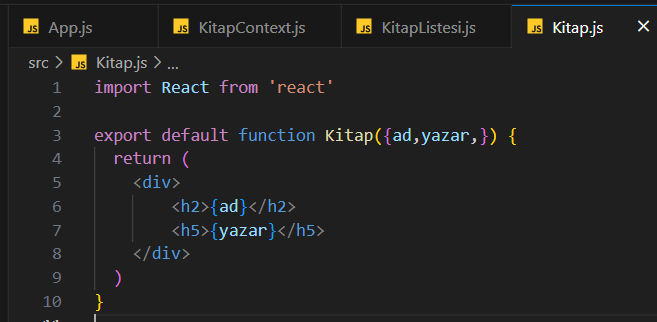


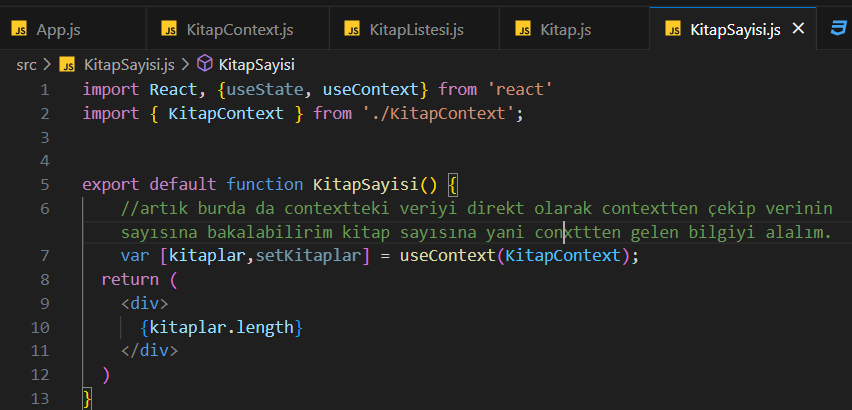


* Context içerisindek bir veriyi diğer componentlerle paylaşabiliriz. Context içerisindeki providerin value attribute u bulunuyor bu value attribute u ile paylaşılacak olan nesneleri verileri yayabiliyoruz. Bu şekilde yayılmış olan verileri burdaki children componentler kullanabilirler. Veriyi kullanacak ilgili component içerisinde useContext i import etmeliyiz. Çünkü kullanacağımız contexti belirtmemiz gerekir Birden fazla contextimiz olabilir ve onun içerisindeki verileri kullanabiliriz. Contexti belirttikten sonra direkt olarak gelen veri formatında bu contextteki açılan veriye componentler içinde kolayca ulaşmış oluruz.

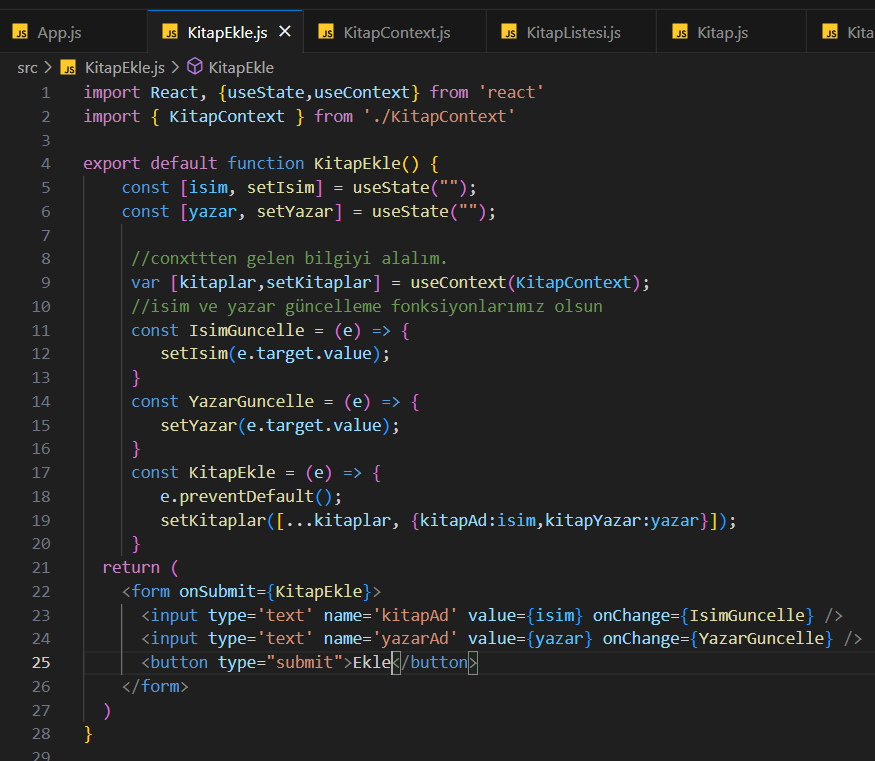






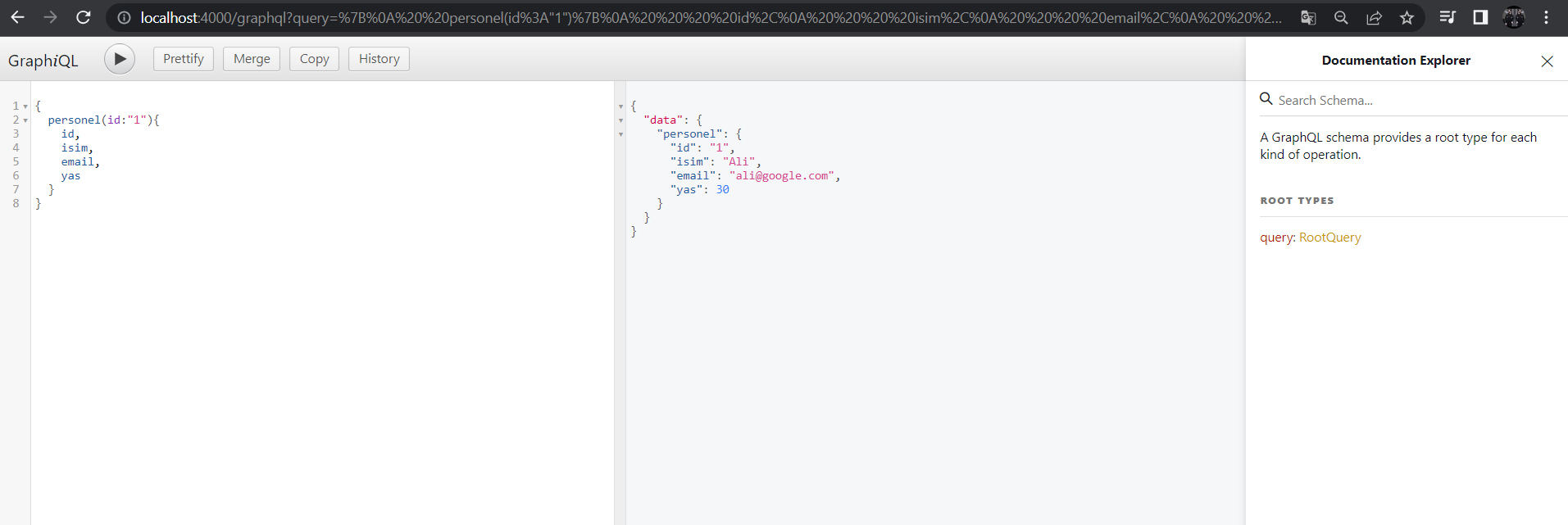


* Burada farkettiysek bir stateyi contextten dışarıya açtık ve sadece statei değil statein set fonksiyonunda dışarıya açtık setKitaplar diyerek. Bunun sayesinde bu fonksiyonu kullanıp kitap ekleme işlemi yapabiliriz veri dizimize.



**GRAPHQL**

* Graphql Client-Server iletişiminde daha fazla esneklik ve verimlilik ihtiyaçları ile baş edebilmek için ortaya atılmış bir yaklaşımdır.
* Özellikle client tarafında hızla değişen isteklerin REST ile karşılanması proje geliştirme maliyetinin artmasını sağladı
* Bu sıkıntıların önüne geçmek için Graphql geliştirilmiştir ve daha karmaşık işlemleri daha kolay yapabilmek hedeflenmiştir.
* GraphQl şema tabanlı çalışır ve tek bir endpoint üzerinden işlemler gerçekleştirilir.
* Tüm data bu endpoint üzerinden iletilir ve ihtiyacımız olan fieldler ile server ile iletişim kurulur.
* **GraphQL Elemenları?;**
* **Types**: Veritabanında bulunan tabloların ve alanların tanımlamalarını oluşturur.
* Queries: Query içerisine yazılabilecek sorgulama işlemlerini tanımlar. Buraya **yazdığımız** tanımlamalar dışında herhangi bir şey çalıştıramayız.
* **Mutations**: Veri oluşturma, silme ve güncelleme işlemleri için kullanılır.
* **Subscriptions**: Subscriptionlar server tarafında belirli bir event gerçekleştiğinde WebSoket kullanarak belirtilen detayı gerçek zamanlı olarak client’A gönderdir.
* **Resolvers**: Graphql çalıştığı zaman yapılacak işlemlerin tanımlandığı alandır.
* Graphql server oluşturabilmek için yarn add express express-graphql paketi gerekli.
* Normal bir Express server oluşturken sonra bir Express middlewear olarak graphql http kullanabiliriz.
* //queryler ile get işlemlerini mutationlar ile post pust delete işlemlerini yapıcaz
* Graphql nesnelerine erişebilmek için yarn add graphql paketine ihtiyaç var.



* Yukardıdaki örnekse ıdsi verilen kaydı istenilen yazdığımız alanlara göre kayıt getiren bir örnek. Resolve fonksiyonu veriye erişim sağlar.
* Json server oluşturabilmek için yarn add json-server paketi lazım.
* Yarn add axios paketlerine ihtiyacım var.
* //güncelleme yaparken patch kullanırsak sadece değiştirdiğimiz alanlar güncellenir geri kalan alanlar old data kalır
* //eğer put kullanırsak tüm alanları girmemiz lazım yoksa bir alanı girip diğerlerini girmezsek diğer old datalarda null olarak güncellenir.
* Apollo sever kullanmak için npm i apollo-server paketine ihtiyaç var.