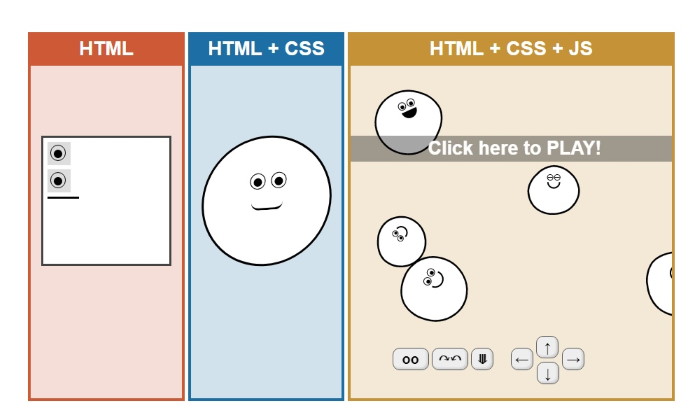
DERS NOTLARI VE TEKRAR TAKİBİ

Days\_47: JavaScript - jquery konu anlatımı \*\*FrontEnd/Days\_6-7 ---> 15/06/2025



### JavaScript nedir?

Geliştiriciler tarafından etkileşimli web sayfaları oluşturmak için kullanılır. Bu dil sunucu tarafı programlama, mobil uygulamalar, oyunlar ve masaüstü uygulamaları için kullanılabilir…

🡪Selector (seçici): HTML belgesi içindeki belirli bir elementi veya element grubunu seçmek için kullanılan tanımlayıcıdır. CSS ve JavaScript bu seçicileri kullanarak HTML öğelerine ulaşır.

CSS ve JavaScript içinde selector/seçici kavramı: HTML öğeleri üzerinde stil (CSS) veya işlem (JavaScript) uygulamak için bu "seçiciler" kullanılır.

\*\*CSS'de selektörler, belirli HTML öğelerine stil vermek için kullanılır.

Örn.

| **Selector Türü** | **Açıklama** | **Örnek** |
| --- | --- | --- |
| **Element selector** | HTML tag’ına göre seçer | p { color: red; } |
| **Class selector** | class değerine göre | .kirmizi { color: red; } |

\*\* JavaScript içinde DOM (Document Object Model) üzerinden HTML öğelerini seçmek için kullanılır.

Örn.

let paragraf = document.getElementById("mesaj");

Eğer seçici kullanmazsak bu objeye karşı tarafın hakimiyetini bekleyemeyiz. Mesela instagramda beğenme butonu yani kalp başta beyaz tıklıyoruz arka plan kırmızı oluyor sağlıyoruz. İd … olan bir like butonum var o like butonuna git ve arka planını kırmızı renge çevir… işte bunu seçici ile yapıyoruz. Seçicilerde,

document.getElementById("mesaj"); ile çalışır.

Document: js çalıştırıldığı ve o anda derlendiği yer, sahne

getElementById: documentin içine bak ve id şu olan veriyi getir.

🡪Const citiesObj = document. getElementById(‘cities’); ifadesi bellekte bir maliyet oluştutur mu?

Hayır, Bellekte zaten cities elemanı var biz cities’e referansını cities objesine atanamasını sağlıyoruz.

Object: Programlamada nesneler, birden çok veriyi ve davranışı (fonksiyonları) tek bir yapıda birleştirmemizi sağlar. JavaScript’te obje (object), birden fazla özellik (property) ve fonksiyon (method) içerebilen veri tipidir.

**Örn.**

function nesne = {

anahtar1: deger1,

key: value,

};

- Obje, anahtar-değer (key-value) çiftlerinden oluşur.

- Her değer, ister veri ister fonksiyon (method) olabilir.

-Obje, gerçek dünyadaki varlıkları yazılımda temsil etmek için çok kullanılır.

**Örn.**

let araba = {

marka: "Toyota",

model: "CVR",

yil: 2025,

calistir: function() {

console.log("Araba çalıştı!");

}

};

\*\*arr.push(obj) // dizi içine veri ekleme

\*\*Buraya kadar olan notların kod dökümünün dosya yolu 🡪 belgeler/github/workspace/frontend/Days\_6

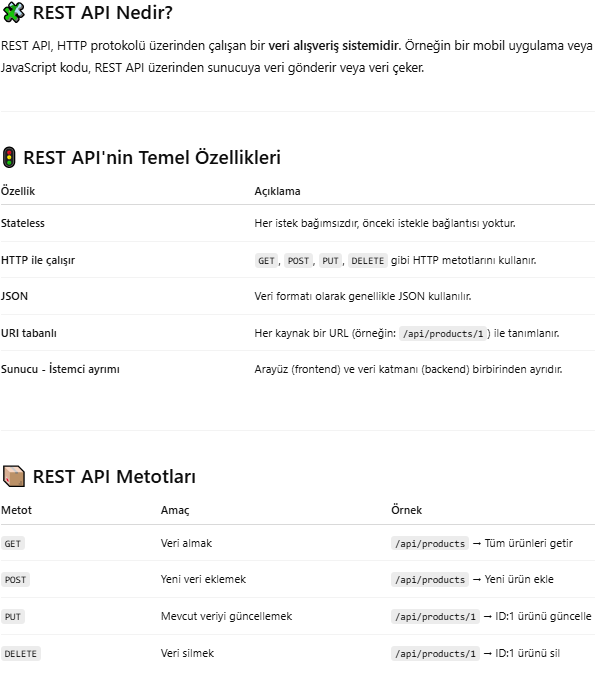
----------------------------------------------------------- \*\*\* ----------------------------------------------------------

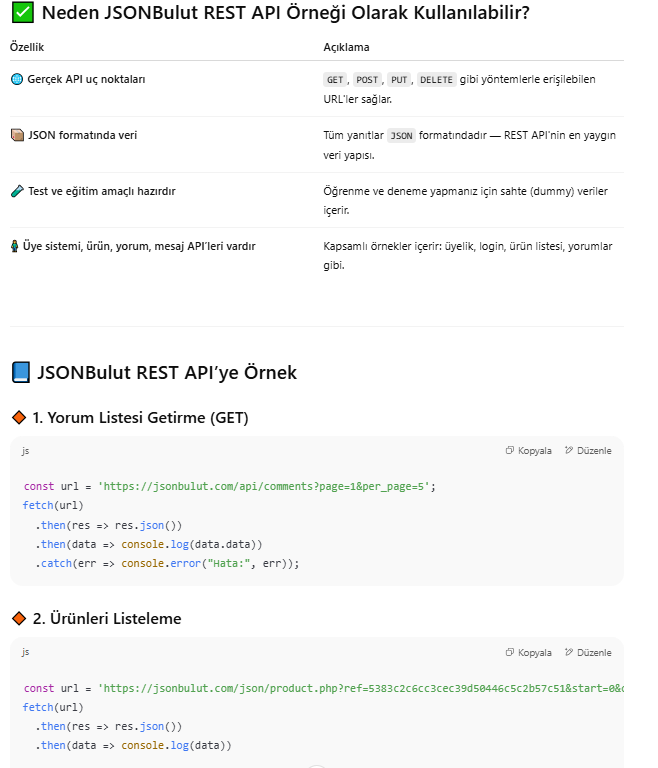
Bu kısımda ki notların kod dökümünün dosya yolu 🡪 belgeler/github/workspace/frontend/Days\_7

AJAX (Asynchronous JavaScript and XML), JavaScript kullanarak sayfa yeniden yüklenmeden web sunucusuyla veri alışverişi yapmamızı sağlayan bir tekniktir.

🧠 Temel Tanım:

* AJAX, web sayfasının arka planda veri alıp göndermesini sağlar.
* Sayfanın yalnızca bir kısmını güncelleyerek kullanıcı deneyimini geliştirir.
* Genellikle API'lerle (REST API) çalışmak için kullanılır.





Yukarıdaki gibi örnek uç noktalar REST API yapısına birebir uyar:

* Veri çekme: GET
* Yeni veri gönderme: POST
* Veri güncelleme: PUT
* Veri silme: DELETE

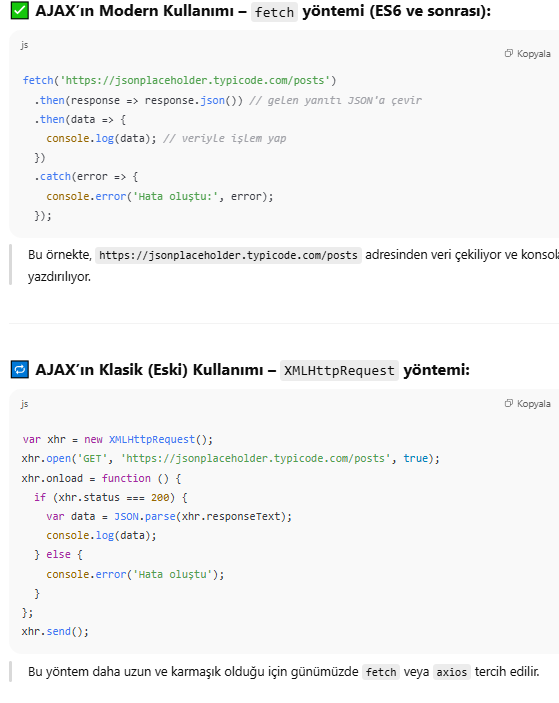
**🔐 Not:**

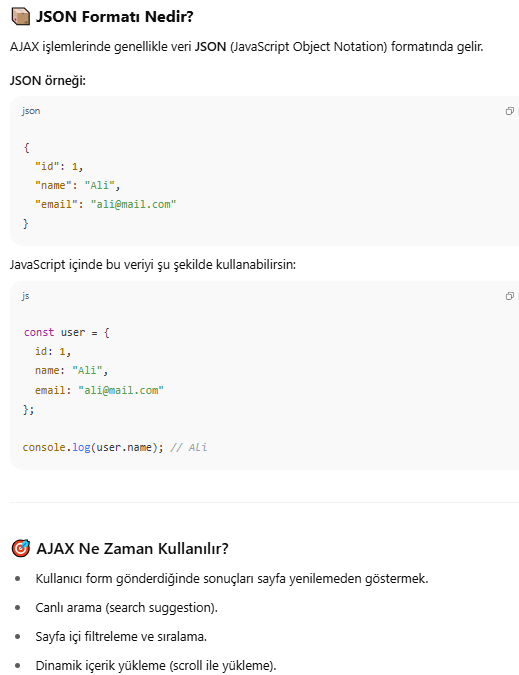
Bazı API'ler ref yani referans anahtarı (API key) gerektirir. JSONBulut da bu yöntemi kullanır. Ücretsiz üye olarak kendi ref anahtarını alıp testler yapabilirsin.

**🔧 Sonuç**

Evet, jsonbulut.com sitesi:

* REST API mantığını anlamak,
* JavaScript ile AJAX/Fetch uygulamaları yapmak,
* Projelerinde sahte veri kullanarak test gerçekleştirmek için **mükemmel bir örnek platformdur.**





\*\*\*İki ters tek tırnak gibi görünen şu işaret: ` (klavye üzerinde genellikle ESC tuşunun altında, 1 tuşunun

Backtick işareti, yazılım dünyasında farklı programlama dillerinde çeşitli amaçlarla kullanılır:

| **Dil / Ortam** | **Kullanımı** |
| --- | --- |
| JavaScript/TypeScript | Template literals, ${} ile değişken gömme |
| Markdown | Kod blokları ve satır içi kod |
| Shell (Bash) | Komut çıktısı almak (eski yöntem) |
| MySQL | Tablo/sütun adlarını belirtmek |
| Python 2 | repr() işlevi (artık önerilmez) |

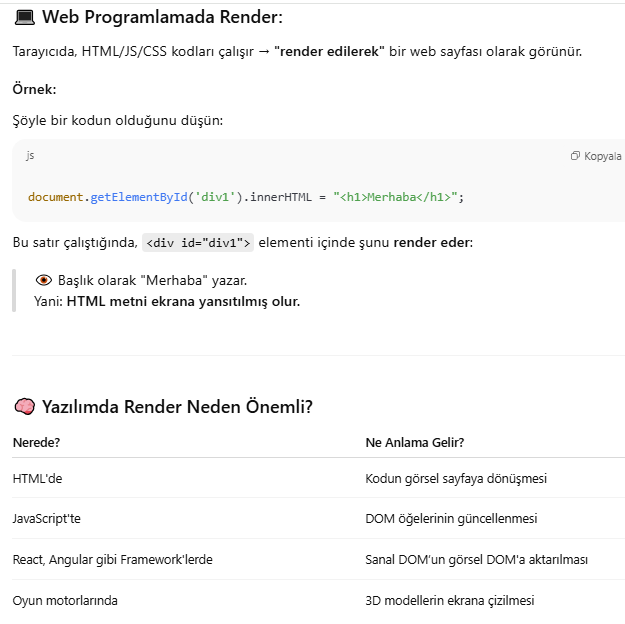
****

****

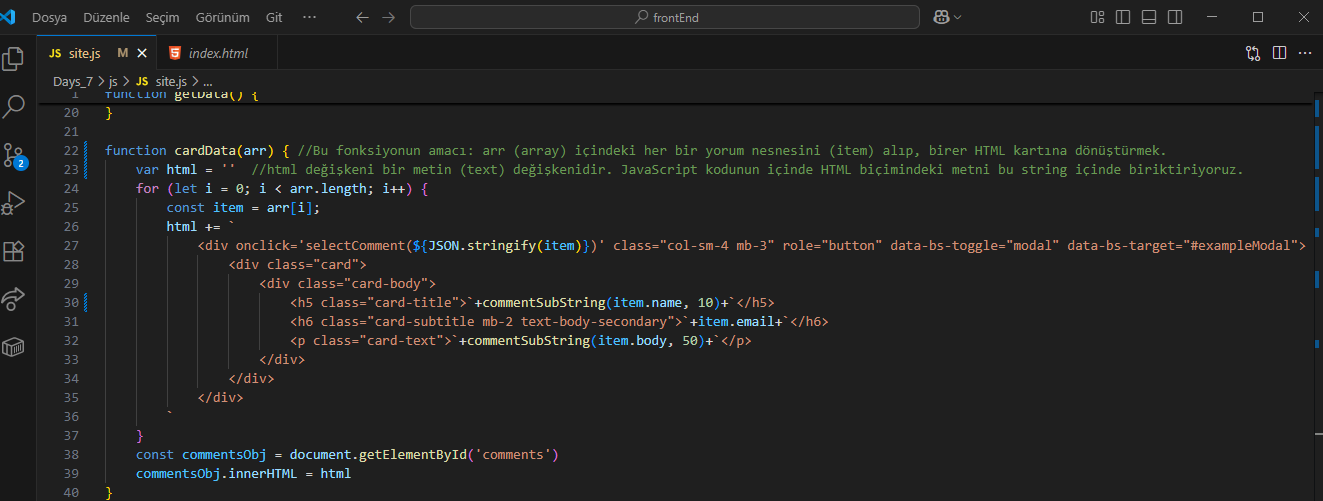
\*\*\* Render, kelime anlamı olarak:

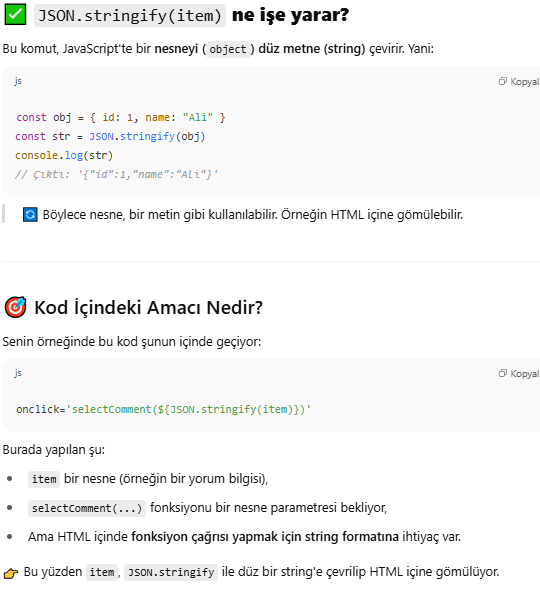
“Görselleştirmek” ya da “ekrana çizmek/göstermek” anlamına gelir.

Yazılımda ise: Veri veya kodun, kullanıcıya görsel olarak sunulmasıdır.



\*\*\*Kod bloğunun açıklaması:







Bu kısma kadar olan notların kod dökümünde de detaylı not alımı var bkz. Days\_7/site.js

----------------------------------------------------------- \*\*\* ----------------------------------------------------------

Days\_48: jquery - JavaScript’i daha az kod yazarak nasıl yönetiriz? \*\*FrontEnd/Days\_8---> 21/06/2025

### **jQuery**, JavaScript dilinde yazılmış, açık kaynaklı ve hafif bir **JavaScript kütüphanesidir**. Web sayfalarında JavaScript kullanımıyla yapılabilecek işleri **daha kısa, daha kolay ve tarayıcı uyumlu şekilde yapmanı** sağlar.

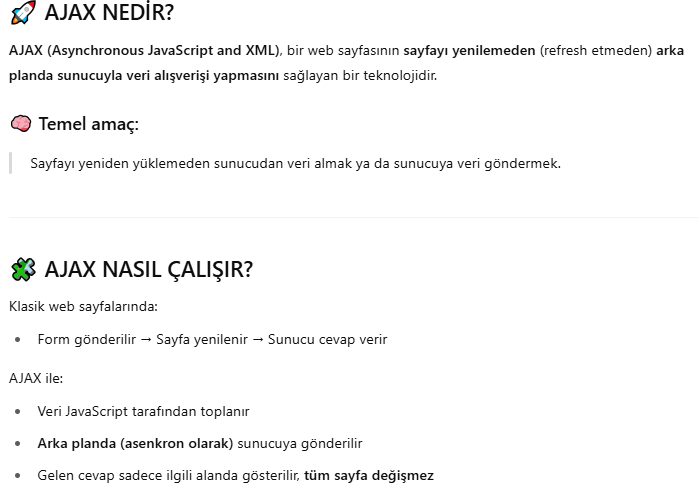
### 

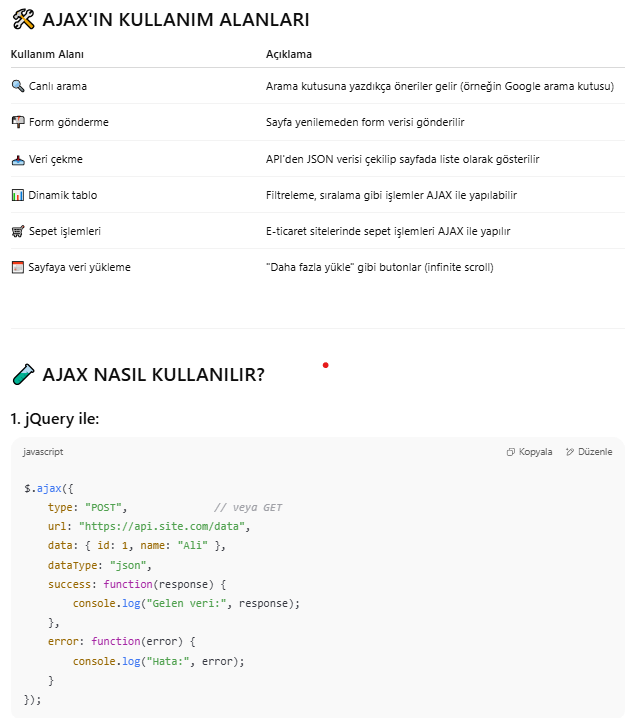
### 

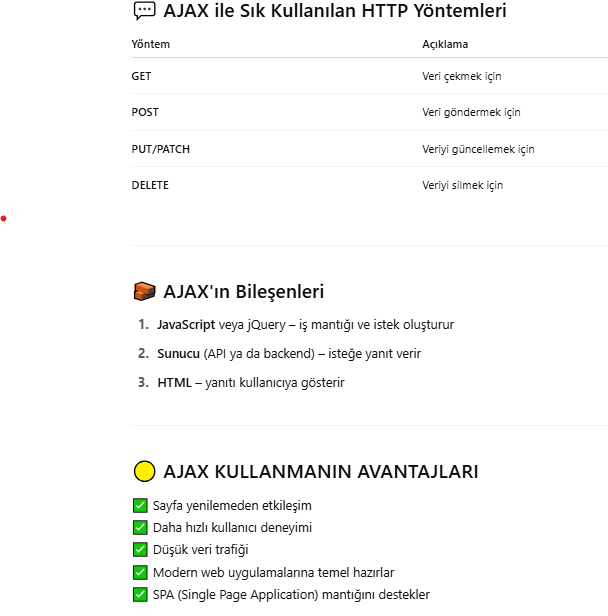
\*\*\*Bkz. bu dersin ekser notları FrontEnd/Days\_8 klasöründe tekrar et!

Days\_49: jquery - JavaScript - AJAX kullanımı \*\*FrontEnd/Days\_9---> 22/06/2025

\*\*bu ders, Days\_48 de işlenen jquery ve js konularının tekrarı yapıldı ve ajax konusu işlendi

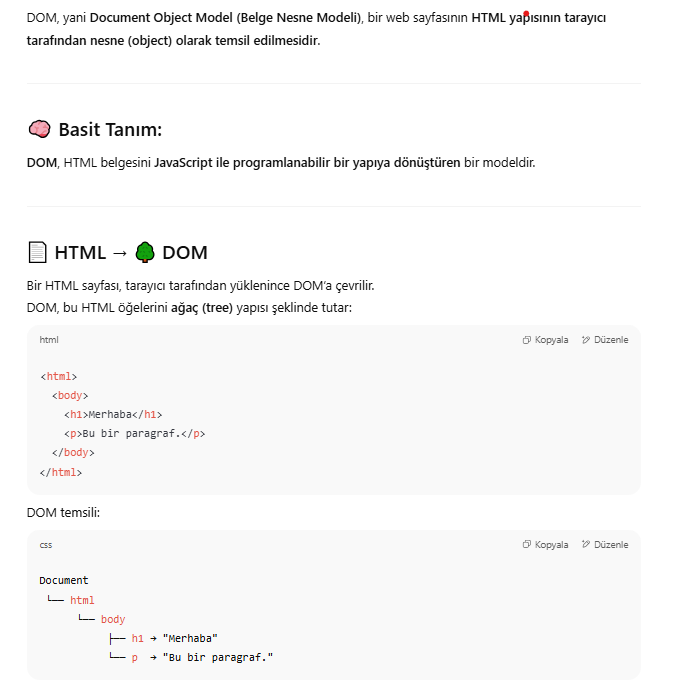


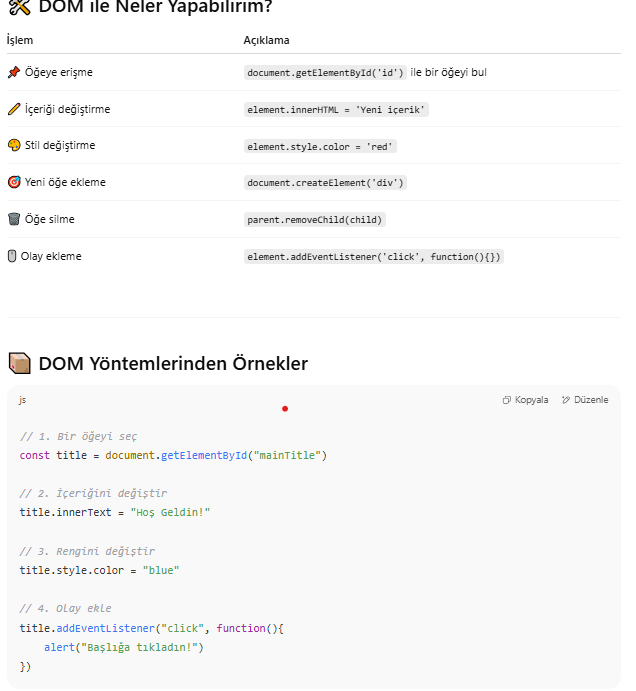






### DOM NEDİR? JavaScript ya da jQuery ile **sayfadaki öğelere erişmek**, **değiştirmek**, **silmek** ya da **yeni öğeler eklemek** için DOM kullanılır.







**API Nedir?**

**API** (Application Programming Interface) yani **Uygulama Programlama Arayüzü**, iki yazılımın birbiriyle iletişim kurmasını sağlayan bir yapıdır. Bir yazılımın ya da servisin, diğer yazılımlar tarafından **nasıl kullanılacağını tanımlayan** bir arayüzdür.

**🔧 Kısa Tanımı:**

API, bir programın başka bir programla konuşabilmesi için sunduğu **kapıdır**. Örneğin:

* **Restoran benzetmesi:** Menü bir API'dir. Mutfakta neler olduğunu bilmen gerekmez, sadece menüdeki isme (API isteği) göre sipariş verirsin, garson (sunucu) yemeği getirir (yanıt döner).

**🧠 Ne İşe Yarar?**

| **Amaç** | **Açıklama** |
| --- | --- |
| **Veri Alışverişi** | Web servisleriyle veri alışverişi sağlar (örnek: hava durumu, harita). |
| **Modülerlik** | Farklı parçaları bağımsız geliştirmeye olanak tanır. |
| **Entegrasyon** | Farklı uygulamaların entegre çalışmasını sağlar. |
| **Gizliliği Koruma** | Arka plandaki karmaşıklığı gizleyerek, sadece gerekli bilgileri sunar. |

**🌍 API Türleri**

| **Tür** | **Açıklama** |
| --- | --- |
| **REST API** | HTTP protokolü ile çalışır. JSON/XML veri döner. En yaygın türdür. |
| **SOAP API** | Daha karmaşık ama güvenlidir. XML tabanlıdır. |
| **GraphQL** | Sadece istenen veriyi döner, esnektir. Facebook geliştirmiştir. |
| **Web API** | Tarayıcı veya mobil uygulamalarda yaygın kullanılan genel terimdir. |

**🔹 Angular’da API Nedir?**

**Angular** dilinde ya da genel olarak yazılımda, iki uygulama veya sistemin birbiriyle **veri alışverişi yapmasını sağlayan bir köprüdür**. Angular, bir **frontend framework** olduğu için, kullanıcı arayüzünü yönetir. Ancak verileri genellikle başka bir yerden alır: **Backend** sunucudan. Bu veri alışverişi genellikle **HTTP istekleriyle** bir API üzerinden yapılır.

🔧 Angular’da API ile Çalışmak Nasıl Olur?

1. HTTP Modülü Kullanılır  
   Angular’da API ile iletişim kurmak için HttpClient servisi kullanılır:

**import { HttpClient } from '@angular/common/http';**

1. API’ye GET/POST İsteği Gönderilir  
   Örnek bir kullanıcı listesini çeken Angular kodu:

**constructor(private http: HttpClient) {}**

**getUsers() {**

**return this.http.get('https://jsonplaceholder.typicode.com/users');**

**}**

1. **Servis Dosyasıyla Organize Edilir**  
   API işlemleri genelde bir servis sınıfında toplanır:

**@Injectable({**

**providedIn: 'root'**

**})**

**export class UserService {**

**constructor(private http: HttpClient) {}**

**getAllUsers() {**

**return this.http.get('https://api.example.com/users');**

**}**

**}**

1. **Component İçinde Kullanılır**

**ngOnInit() {**

**this.userService.getAllUsers().subscribe(data => {**

**console.log(data);**

**});**

**}**

**🔁 Örnek Senaryo**

| **Amaç** | **API** |
| --- | --- |
| Kullanıcı giriş yapacak | POST /login |
| Ürünler listelenecek | GET /products |
| Yeni ürün eklenecek | POST /products |
| Ürün silinecek | DELETE /products/:id |

**📌 Özetle:**

* API = frontend (Angular) ile backend (veritabanı/sunucu) arasında **veri köprüsü**.
* Angular’da API istekleri için HttpClient servisi kullanılır.
* API çağrıları genellikle servis dosyalarında yazılır ve component’te kullanılır.

\*\*\*jsonbulut.com site tanıtımı 🡪 <https://jsonbulut.com/>

jsonbulut.com, **Free Fake REST API** hizmeti sunan bir platformdur. Bu servis, geliştiricilerin gerçek bir sunucu kurmadan web ve mobil uygulamalar için REST API çağrılarını hızlıca test edip prototip oluşturmalarına olanak tanır.

Temelde, jsonbulut.com kullanıcılarına yorumlar (comments), gönderiler (posts), ürünler (products), yapılacaklar (todos) ve kullanıcılar (users) gibi sık kullanılan verilerle aşağıdaki işlemleri deneme fırsatı sağlar:

* **GET, POST, PUT, DELETE** talepleriyle API uç noktalarının nasıl çalıştığını görselleştirme
* JWT (JSON Web Token) ile kimlik doğrulama kullanıcı akışı test etme
* Veriler üzerinde silme, güncelleme, listeleme gibi operasyonlar gerçekleştirme [jsonbulut.com+1jsonbulut.com+1](https://jsonbulut.com/apidocs?utm_source=chatgpt.com)

Geliştirme sürecinizde kendi backend’inizi yazmak yerine bu hazır sahte API’yi kullanarak ön yüz (frontend) ya da mobil uygulamanızı hızla geliştirebilir, test edebilirsiniz. Giriş yapmanız gereken senaryolar için örnek kullanıcı bilgileri de (e‑posta/parola) sağlanmış durumda.

**Özetle**

| **Konu** | **Açıklama** |
| --- | --- |
| Amaç | API isteklerini hızlıca test etmek, prototip geliştirmek |
| Sağlananlar | yorumlar, gönderiler, kullanıcılar, ürünler, yapılacaklar gibi uç noktalar |
| İşlemler | GET, POST, PUT, DELETE, JWT destekli authentication |
|  |  |

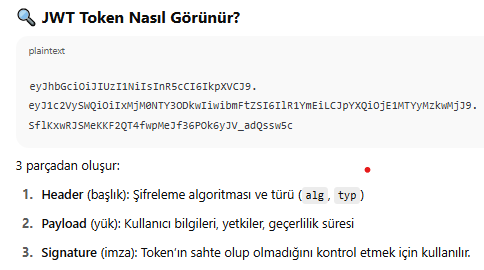
### 🔐 JWT Token Nedir? Bkz.-->Days\_9/login.js/39.kod satırı

JWT Token (JSON Web Token), bir kullanıcının kimliğini doğrulamak ve bu kimliğe bağlı yetkileri taşımak için kullanılan **şifreli bir dijital kimlik kartıdır**. Web uygulamalarında oturum (login) yönetimi için sıkça kullanılır.

JWT, kullanıcı giriş yaptıktan sonra sunucunun oluşturup istemciye (örneğin tarayıcıya) verdiği **bir metin yapısıdır**. Bu token:

* Kullanıcının **kim olduğunu**,
* Hangi işlemleri yapabileceğini (**yetki**),
* Ne kadar süreyle geçerli olduğunu belirtir.

Bu token genellikle her istekte sunucuya gönderilir, böylece sunucu tekrar tekrar kullanıcıdan kullanıcı adı/parola istemez.



**✅ JWT Ne İşe Yarar?**

* Kullanıcı giriş yaptıktan sonra access\_token sunucu tarafından döner.
* Tarayıcı bu token’ı localStorage/sessionStorage gibi bir yerde saklar.
* Kullanıcı her API isteğinde bu token’ı sunucuya gönderir.
* Sunucu, bu token’ı kontrol edip kullanıcının gerçekten giriş yapmış ve yetkili olup olmadığını anlar.

**🧠 Örnek Senaryo:**

1. Kullanıcı giriş yapar.
2. Sunucu access\_token yani JWT token döner:

js

KopyalaDüzenle

response.data.access\_token = "eyJhbGciOi..."

1. Tarayıcı bu token’ı saklar.
2. Artık kullanıcı sayfalar arasında dolaşırken tekrar giriş yapmasına gerek kalmaz.
3. Token süresi bittiğinde yeniden giriş yapılması gerekir.

\*\*\* <https://jsonviewer.stack.hu/> sitesi, JSON (JavaScript Object Notation) formatındaki verileri **okunabilir bir şekilde görüntülemek ve analiz etmek** için kullanılan **ücretsiz bir çevrimiçi araçtır**.

**📌 Ne zaman kullanılır?**

* API'den gelen karmaşık JSON cevabını **okunur hale getirmek için**
* JWT Token içindeki base64 çözümlenmiş payload'ı anlamak için
* JSON verisini debug (hata ayıklama) etmek için
* Kodlama yaparken test JSON verisi hazırlamak için

# Days\_50: Angular diline giriş – uygulama kurulumları \*\*FrontEnd/Project---> 29/06/2025

* Yeni Proje Oluşturma: ng new Project
* Projeyi çalıştırma: ng serve –open
* Projeyi durdurma: ^C
* Proje Module yüklemsi: npm install
* Proje port değişimi: ng serve --port 4401 –open
* Npm kullanımı: "start": "ng serve --port 4401 --open",
* projeyi çalıştırmak için terminalde sadece "npm start"
* Yeni sayfa: ng g c pages/login
* Components oluşturma: ng g c components/bar
* \*\* Kurulum\*\*
* 1- NodeJs Kurulumu: <https://nodejs.org/en>
* Node Sürümünü Alma: node -v
* Npm Sürümünü Alma: npm -v
* 2- Angular CLI Kurulumu: npm install -g @angular/cli
* Angular Sürümünü Alma: ng v
* 3- VS Code eklentileri:
* Name: Angular Language Service
* Id: Angular.ng-template
* Description: Editor services for Angular templates
* Version: 20.0.1
* Publisher: Angular
* VS Marketplace Link: https://marketplace.visualstudio.com/items?itemName=Angular.ng-template

Name: Angular Snippets (Version 18)

Id: johnpapa.Angular2

Description: Angular version 18 snippets by John Papa

Version: 18.0.2

Publisher: John Papa

VS Marketplace Link: <https://marketplace.visualstudio.com/items?itemName=johnpapa.Angular2>

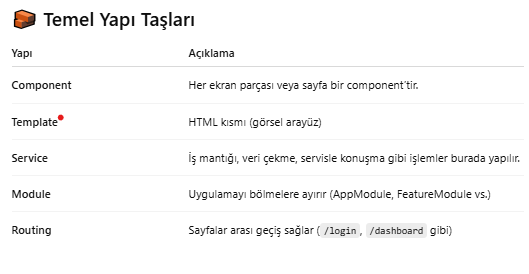
### **🔧** ANGULAR NEDİR?

Angular, Google tarafından geliştirilen ve **modern web uygulamaları** oluşturmak için kullanılan güçlü bir **front-end (ön yüz) framework**’tür.

**🔧 Kısaca Angular**, **tek sayfa uygulamalar (SPA)** geliştirmeye yarayan, TypeScript tabanlı, bileşen (component) tabanlı bir **framework**’tür.

**🎯 Angular Ne İşe Yarar?**

* Büyük, dinamik web uygulamaları oluşturmanı sağlar.
* HTML + CSS + TypeScript ile çalışır.
* Veriyi gösterme, kullanıcı etkileşimi, formlar, HTTP istekleri, routing (sayfalar arası geçiş) gibi birçok özelliği hazır olarak sunar.

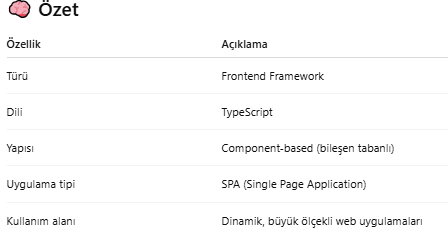
****

**⚙️ Angular ile Neler Yapabilirsin?**

* Admin panelleri
* Kullanıcı giriş sistemleri
* Blog siteleri
* E-ticaret arayüzleri
* Gerçek zamanlı paneller (grafikler, socket bağlantısı)
* Progressive Web App (PWA)

**📦 Angular Kullanmak İçin Neye İhtiyacın Var?**

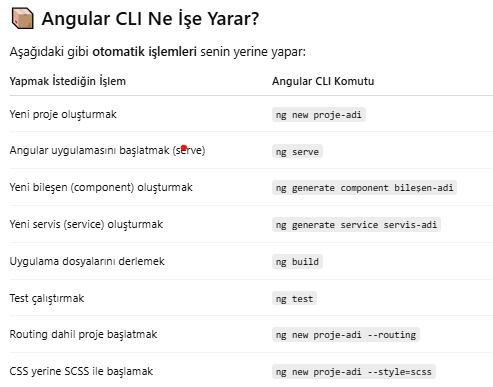
* Node.js ve npm
* Angular CLI (npm install -g @angular/cli)
* Kod editörü (Visual Studio Code gibi)
* Tarayıcı (Chrome, Edge vs.)



**🧭 Angular CLI Nedir?**

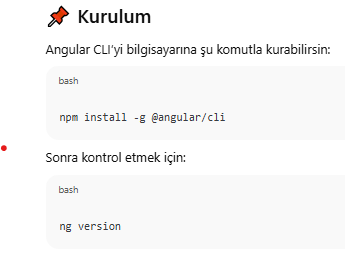
Angular CLI, "**Command Line Interface**" yani **Komut Satırı Arayüzü** anlamına gelir ve Angular projelerini kolayca oluşturmak, yönetmek ve geliştirmek için kullanılan bir araçtır.

**Angular CLI**, terminal (komut satırı) üzerinden Angular uygulamalarıyla ilgili işlemleri yapmanı sağlayan bir araçtır.  
Angular ekibi (Angular resmi geliştiricileri) tarafından geliştirilmiştir.



**🚀 Avantajları**

✅ Dosya yapısını otomatik oluşturur  
✅ En iyi Angular pratiklerini takip eder  
✅ Yazım hatalarını azaltır  
✅ Geliştirme sürecini çok hızlandırır  
✅ Derleme, test, build gibi karmaşık işleri sadeleştirir

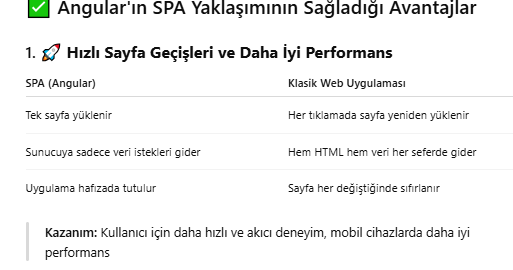
**🎯 Özet**

**Angular CLI**, Angular geliştirme sürecini **hızlandırmak**, **standartlaştırmak** ve **kolaylaştırmak** için kullanılan çok güçlü bir araçtır.  
Kodu elle yazmak yerine CLI komutlarıyla otomatik üretebilir, zaman kazanırsın.

### \*\*Angular’ın SPA (Single Page Application – Tek Sayfa Uygulama) yapısını kullanması, hem performans hem de kullanıcı deneyimi (UX) açısından önemli avantajlar sağlar.

**SPA Nedir? SPA (Single Page Application)**:

* Uygulama ilk yüklendiğinde sadece bir HTML dosyası sunucu tarafından gönderilir.
* Sonraki sayfa geçişleri (navigasyon), sayfayı yeniden yüklemeden **JavaScript ile dinamik olarak** yapılır.
* Sayfa sadece gereken parçaları yeniden çizerek hızlı tepki verir.



**2. 📡 Daha Az Sunucu Yükü ve Daha Az Bant Genişliği Kullanımı**

* Angular, yalnızca veri (JSON gibi) alır ve ekranda gösterir.
* Her sayfa için tekrar tekrar HTML, CSS yüklemeye gerek kalmaz.

**Kazanım:** Sunucu maliyetleri azalır, API kullanımı daha verimli olur.

**3. 🔄 Yeniden Kullanılabilir Bileşenler (Component Tabanlı Mimari)**

* Angular'daki her sayfa ya da arayüz parçası bir bileşen (component) olarak tanımlanır.
* Bu bileşenler **SPA içinde dinamik olarak çağrılır**.

**Kazanım:** Kod tekrarını azaltır, bakımı ve geliştirmeyi kolaylaştırır.

**4. ⚡ Client-side Routing ile Sayfa Yenilemeden Navigasyon**

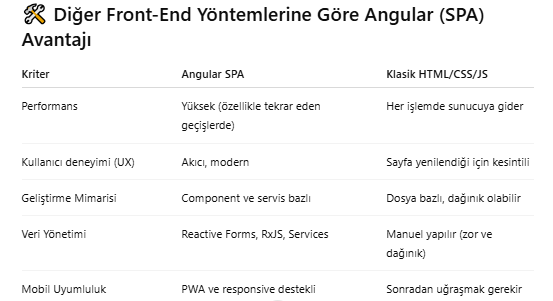
* Angular RouterModule sayesinde /home, /about gibi rotaları sayfa yenilemeden yönetir.
* Kullanıcı bir bağlantıya tıkladığında yalnızca içerik değişir, sayfa yeniden yüklenmez.

**Kazanım:** Kullanıcı deneyimi native uygulamalara yaklaşır.

**5. 🔐 Gelişmiş Ön Yüz Yetkilendirme (Guard, Interceptor)**

* SPA'da Angular ile kolayca:
  + Kimlik doğrulama (JWT Token)
  + Sayfa bazlı yetki denetimi (AuthGuard)
  + Oturum yönetimi yapılabilir

**Kazanım:** Tek bir çatı altında, daha güvenli ve kontrollü uygulama akışı

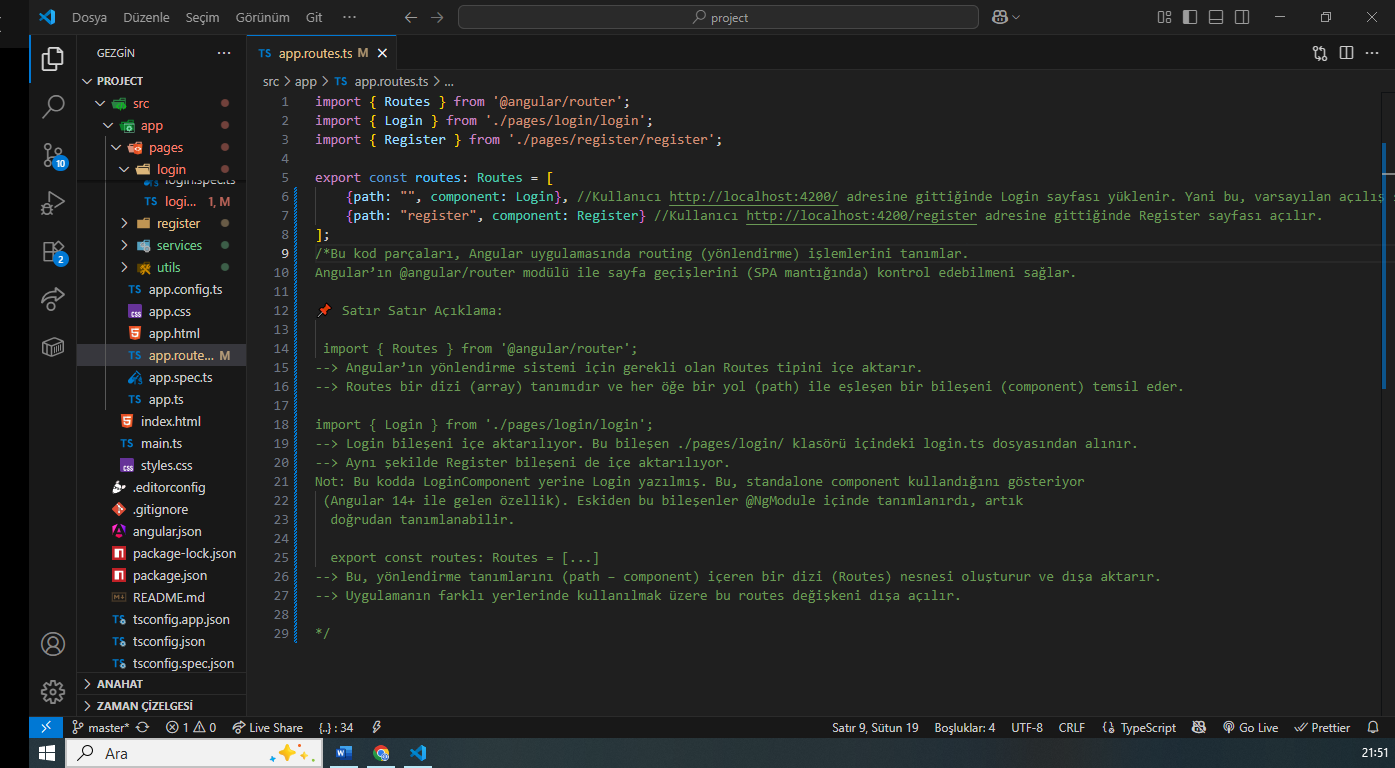


**📌 Özetle:**

**Angular + SPA mimarisi**, büyük, dinamik ve sürekli gelişen projelerde:

* ✅ Daha hızlı kullanıcı deneyimi sunar
* ✅ Geliştiriciye bileşen temelli, sürdürülebilir mimari sağlar
* ✅ Sunucuya yük bindirmeden veriyi etkili işler
* ✅ Modern web uygulamaları için idealdir (özellikle dashboard, admin panel, e-ticaret, CRM gibi yapılar için)

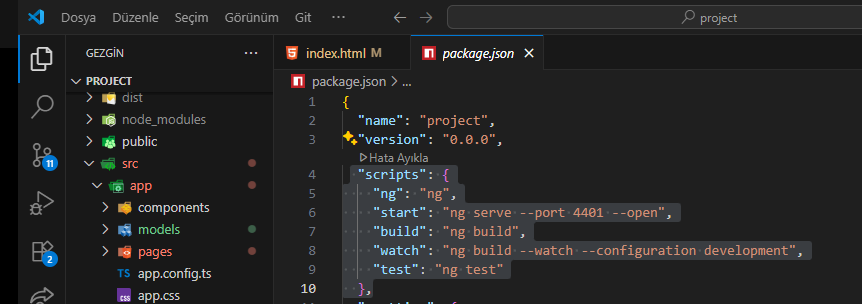
Dersten kod örnekleri:



Kısa yol satır kopyalama: alt + shift + alt ok yön tuşu

### Days\_51: Geçen dersin tekrarı + Angular form yönetimi - Login-Register işlemleri \*\*FrontEnd/Project---> 05/07/2025

🡪 package.json altındaki kodlarından

 bu kod kümesinin amacı??

Terminale npm start deriz bu start yukardaki startı bulur ve karşısındaki portu çalıştırır.

🡪 Companent nedir?

Yeni sayfa oluşturma ng g c pages/login

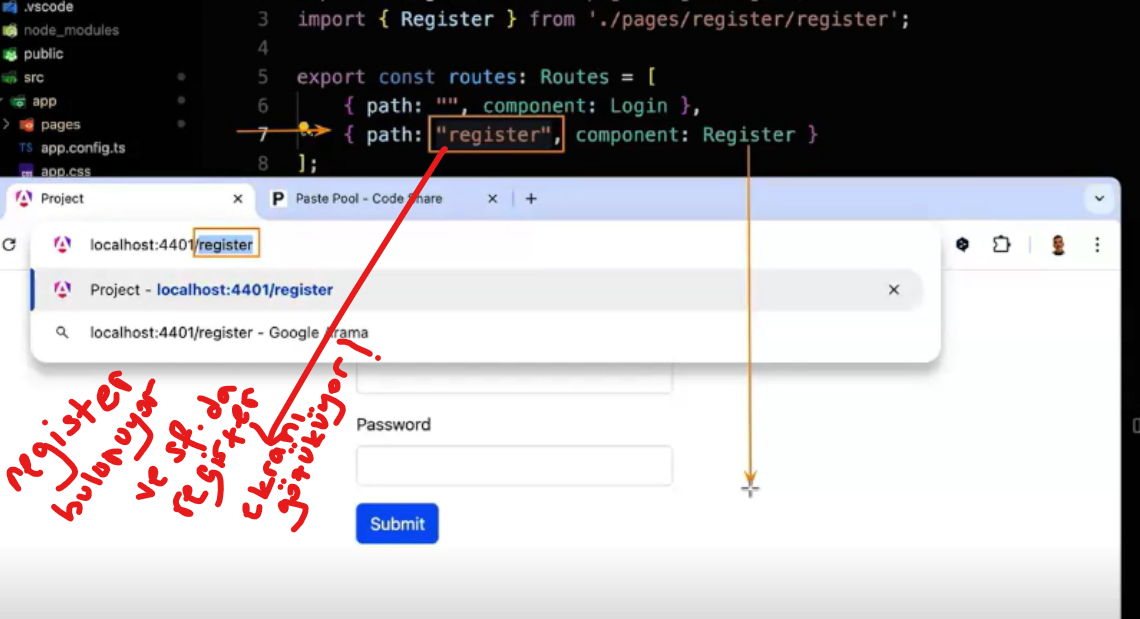
🡪Login veya register sayfası nasıl açılır?

Rotalama ile yani path yolunu yazarak. Angularda rotalamayı app.routes.ts dosyası sağlar.

🡪 export nedir? C# da ki public gibi dışardan kullanımlara olanak sağlar.

🡪providerRouter: program boyunca etki demektir.

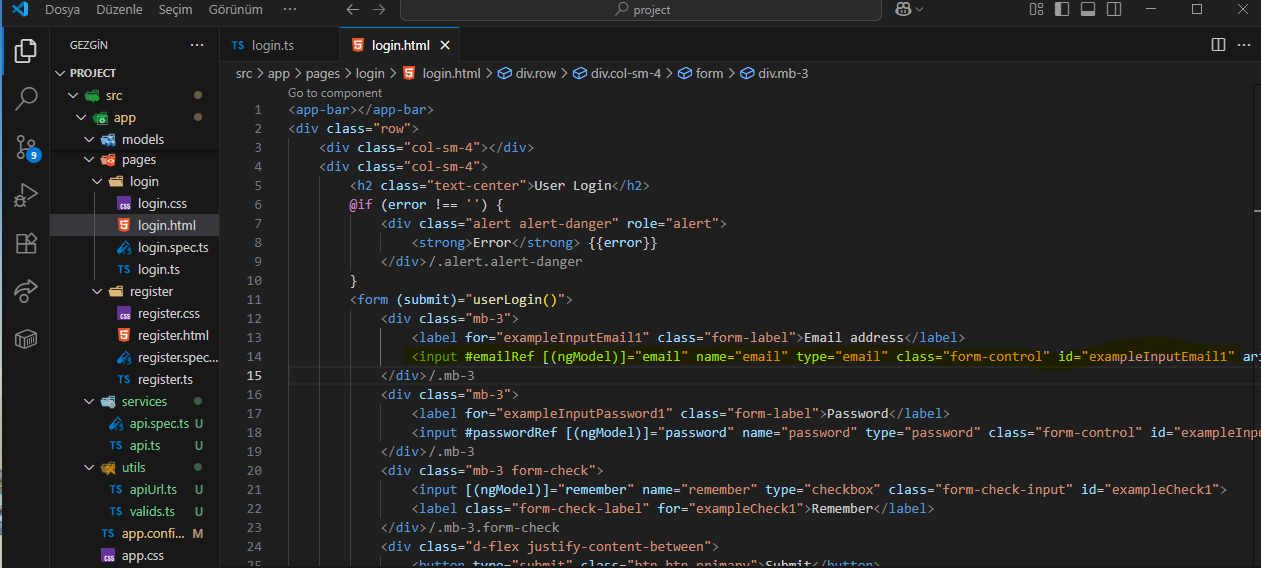
🡪 path nedir? url’de ki kısmı ifade eder.



🡪app.spec.ts ne işe yarar? Test için kullanılır.

Terminalde ng test deriz ve bize spec listesini getirir

🡪.ts ne işe yarar? Programa dinamik olaylar eklemede yardımcı olur mesela butona tıklama form gönderme gibi işlemlerde yardımcı olur.



11-15. satırların ne işe yaradığını detaylı açıklayalım:

**🔹 <form (submit)="userLogin()">**

* form etiketi bir formu temsil eder.
* **(submit)="userLogin()"**: Form gönderildiğinde (Enter tuşu ya da submit butonu tıklanınca) Angular'da userLogin() fonksiyonunu çağırır.

⚠️ Not: Bu form HTML'de action="" yerine Angular içindeki bir TypeScript fonksiyonunu çalıştırır.

**🔹 <div class="mb-3">**

* Bootstrap sınıfıdır.
* mb-3: "margin-bottom: 1rem" → Yani input alanları arasında **dikey boşluk** bırakır.

**🔹 <label for="exampleInputEmail1" class="form-label">Email address</label>**

* label: Kullanıcıya alanın ne için olduğunu gösterir ("Email address").
* for="exampleInputEmail1": Hangi input’a ait olduğunu belirtir.

Bu, id="exampleInputEmail1" olan input’a bağlıdır.  
Ekran okuyucular için erişilebilirlik sağlar.

**🔹 🔹 <input ...> (En önemli satır)**

**<input #emailRef [(ngModel)]="email" name="email" type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp"> </div>**

**1. #emailRef**

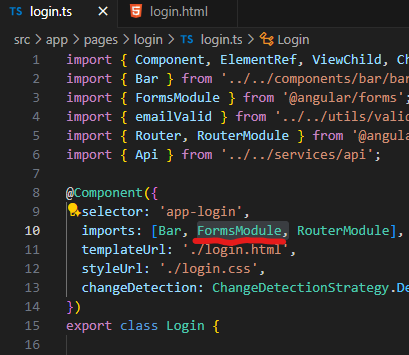
* Bu, Angular’daki **template reference variable**’dır.
* HTML tarafında bu input'a **referans vermek** için kullanılır.
* Bu sayede TypeScript dosyasına gerek kalmadan bu input'u HTML'de tanımlı başka yerlerde doğrudan kullanabilirsin.

**🔹 [(ngModel)]="email"**

Bu ifade Angular'ın **two-way data binding (çift yönlü veri bağlama)** özelliğidir.

* email adlı component değişkeni ile bu input arasında **eşzamanlı veri akışı** sağlar.
* Kullanıcı input'a bir şey yazarsa → email değişkenine otomatik aktarılır
* email değişkeni değişirse → input'un içeriği otomatik güncellenir

[(ngModel)] kullanımı için FormsModule'un app.config.ts veya imports kısmında eklenmiş olması gerekir.



**🔹 name="email"**

* Bu input’a bir **form alanı ismi** verir.
* ngModel ile birlikte kullanıldığında formun bir parçası olarak Angular tarafından tanınır.
* Özellikle ngForm ile birlikte kullanılacaksa **zorunludur**.

**🔹 type="email"**

* HTML5 özelliğidir.
* Tarayıcıya bu input’un **email formatı** beklediğini söyler.
* Yanlış formatta girilirse tarayıcı kullanıcıyı uyarır.
* Bazı mobil cihazlarda **email klavyesi** açılmasını da sağlar (ör: @ sembolü olan klavye).

**🔹 class="form-control"**

* Bu, **Bootstrap** sınıfıdır.
* Girdi alanını stilize eder.
* Görsel olarak düzgün, responsive bir input kutusu oluşturur.

**🔹 id="exampleInputEmail1"**

* Bu input’a **benzersiz bir HTML kimliği (ID)** atar.
* Genellikle <label for="exampleInputEmail1"> gibi erişim etiketleri ile eşleştirilir.

**🔹 aria-describedby="emailHelp"**

* Bu bir **erişilebilirlik (accessibility)** özelliğidir.
* emailHelp ID’li başka bir HTML elementini (örneğin bir <small> etiketi) bu input’a açıklayıcı bilgi olarak bağlar.

## 🔄 Özetle Bu Satır Ne İşe Yarıyor?

🔹 Bootstrap ile görsel olarak şık  
🔹 Angular ngModel ile veri bağlı  
🔹 #emailRef ile HTML tarafında referanslanabilir  
🔹 type="email" ile doğrulama ve erişilebilirlik sağlar  
🔹 name, id, aria-describedby ile form yapısına tam entegre olur

| **Parça** | **Ne işe yarar?** |
| --- | --- |
| (submit)="userLogin()" | Form gönderilince TypeScript fonksiyonu çalıştırır |
| [(ngModel)]="email" | Email değişkeni ile input arasında çift yönlü bağ kurar |
| #emailRef | Bu input’a HTML içinde referans sağlar |
| type="email" | Email doğrulaması ve mobilde email klavyesi sağlar |
| class="form-control" | Bootstrap ile stil verir |
| id, for, aria-... | Erişilebilirlik ve bağlantı için kullanılır |

### \*\* Angular'da form yapıları oluştururken sıkça karşılaştığımız ngModel, required, ngModelOptions, ngForm, [(ngModel)], #templateRef gibi ifadeler bazılarına ilk bakışta karışık gelebilir bu kavramları kısaca anlatalım:

**✅ 1. [(ngModel)]**

**🔹 Ne işe yarar?**

* Angular'da **iki yönlü veri bağlama (two-way binding)** sağlar.
* Yani:
  + Kullanıcı input’a veri girerse → component değişkeni güncellenir.
  + Component’teki değişken değişirse → input’un içeriği de değişir.

**🔍 Örnek:**

**<input [(ngModel)]="email" name="email" type="email">**

* Bu satır sayesinde kullanıcı input’a yazdığında this.email güncellenir.
* this.email = 'abc@example.com' yaparsan, input da otomatik bu değeri gösterir.

**Not:** FormsModule ekli değilse çalışmaz!

**✅ 2. required (HTML özelliği)**

**🔹 Ne işe yarar?**

* HTML5 form doğrulamasıdır.
* Input’un **boş geçilmesini engeller.**

**🔍 Örnek:**

**<input [(ngModel)]="email" name="email" required>**

* Kullanıcı formu boş email ile gönderemez.
* Tarayıcı otomatik uyarı verir ("Bu alan doldurulmalıdır").

**✅ 3. name="..."**

**🔹 Ne işe yarar?**

* Angular form sisteminde her input’un bir name özelliği olmalıdır.
* Özellikle ngModel ile birlikte FormsModule kullanıyorsan **zorunludur.**

**✅ 4. ngModelOptions**

**🔹 Ne işe yarar?**

* ngModel'in nasıl çalışacağını kontrol eder (örneğin: debounce süresi, güncelleme zamanı vs.)

**🔍 Örnek:**

**<input [(ngModel)]="searchTerm" [ngModelOptions]="{ updateOn: 'blur' }">**

* Bu input’un değeri **sadece odak dışına çıkınca (blur)** güncellenir.

Alternatifler:

* 'change' (her değişiklikte)
* 'blur' (odaktan çıkınca)
* 'submit' (form submit edilince)

**✅ 5. #emailRef gibi ifadeler**

**🔹 Ne işe yarar?**

* Bu bir **template reference variable**'dır.
* HTML tarafında bu input’u temsil eder.
* Input’un DOM nesnesine erişmeni sağlar.

**🔍 Örnek:**

**<input #emailRef [(ngModel)]="email">**

<button (click)="console.log(emailRef.value)">Göster</button>

**✅ 6. ngForm**

**🔹 Ne işe yarar?**

* Angular'da bir formun NgForm ile takip edilmesini sağlar.
* Angular otomatik olarak <form> etiketine ngForm direktifini atar.

**🔍 Örnek:**

**<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm)">**

**🎯 Özet Tablo**

| **Terim** | **Açıklama** |
| --- | --- |
| [(ngModel)] | İki yönlü veri bağlama |
| required | Boş bırakılmaması gereken alan |
| name | Form alanı tanımlaması (Angular Forms için zorunlu) |
| ngModelOptions | ngModel davranışını özelleştirme |
| #emailRef | HTML içinden input'a referans alma |
| ngForm | Formu Angular tarafında takip etmeye yarar |

📌**DİKKAT!**

* ng-model, ng-required gibi direktifler **eski AngularJS (1.x)** için geçerlidir.
* Modern Angular (2+ ve sonrası) bunları [(ngModel)], required, formControl, formGroup gibi yapılarla değiştirir.

Eğer sen **Angular 15–16–17** gibi modern bir sürüm kullanıyorsan:

* [(ngModel)] → ✅ kullanabilirsin
* ng-model, ng-required → ❌ **KULLANMA!** (çalışmaz)

\*\*valid.ts dosyasındaki notlara bak

### Days\_52: Angularda Register işlemleri \*\*FrontEnd/Project---> 06/07/2025

Soru? API için angular dilinde ya da genel olarak yazılımda iki uygulama arasında sistemin birbiriyle alışveriş yapmasını sağlayan bir köprüdür, dedin. Peki kafama şu takıldı, bundan önceki çalışmalarımızda PHP, C Sharp gibi dillerde internet ortamına bağlanabilmek için veri tabanı kullanıyorduk. Ama burada database yok, API'lerle bu sağlanıyor. Değil mi?

**ChatGPT:**

Evet, **çok yerinde bir gözlemde bulunmuşsun** ve sorunun altında yazılım mimarisine dair önemli bir fark yatıyor. Şimdi bunu açık ve sade şekilde açıklayayım:

**✅ Eskiden ne yapıyorduk?**

PHP, C#, vb. dillerle uygulama geliştirirken:

* **Veritabanı** doğrudan uygulamanın içindeydi.
* Uygulama, doğrudan **MySQL, MSSQL, PostgreSQL** gibi veritabanlarına bağlanır, SELECT, INSERT gibi sorgular gönderirdik.
* Sunucu ile istemci (örneğin tarayıcı) **tek bir sistem içinde** barınırdı.

**✅ Şimdi ne yapıyoruz?**

Modern mimaride (özellikle Angular, React, mobil uygulamalar vs.):

* **Frontend (istemci)**: Angular gibi dillerle yapılan kullanıcı arayüzü.
* **Backend (sunucu)**: PHP, Node.js, .NET Core, Python vb. dillerle yapılan arka uç.
* **Veritabanı**: Sadece backend tarafından erişilir.

➡️ **Frontend doğrudan veritabanına bağlanmaz!**  
Frontend, sadece **API üzerinden** veriye erişir. Yani:

**Angular → API → Backend → Database**

**🔄 Peki neden böyle yapılıyor?**

| **Neden** | **Açıklama** |
| --- | --- |
| **Güvenlik** | Veritabanına doğrudan erişim açık olursa, sistem tamamen savunmasız kalır. |
| **Ayrıştırma (Separation)** | Frontend ve Backend ayrı ayrı geliştirilip yönetilebilir. |
| **Performans** | Backend, veriyi optimize ederek frontend'e sadece ihtiyaç duyulanı gönderir. |
| **Tekrar kullanılabilirlik** | Aynı API hem mobil hem web için kullanılabilir. |

**🎯 Örnek:**

Eskiden (örneğin PHP ile bağlantı):

**$conn = mysqli\_connect(...);**

**$result = mysqli\_query($conn, "SELECT \* FROM products");**

Modern yapı (Angular + API):

1. Angular'dan istek:

**this.http.get('https://dummyjson.com/products').subscribe(...)**

1. Bu istek bir API sunucusuna gider.
2. API, veritabanından veriyi çeker ve JSON olarak geri döner.

**🔍 O zaman özetle diyebiliriz ki:**

* **API = veri alışveriş köprüsü**
* **Veritabanı hâlâ var**, ama doğrudan frontend’ten erişilmiyor.
* Angular, React gibi teknolojiler **yalnızca API ile konuşur**, database ile değil.

**Şunu dedin:**

Peki yazdıklarından şunu mu çıkarmalıyım? Angular, JSON gibi dillerde API sayesinde sunucuya bağlanıyoruz. PHP, C# gibi dillerde database ile sunucuya bağlanıyoruz. Yani her dilin ortak noktası günün sonunda internete açılabilmek için bir sunucuya bağlanmak, bir kullanıcı ile etkileşime girebilmek için sunucuya bağlanmak. Ama gittikleri yollar farklı. Yani JSON formatı, Angular gibi yapılarda API bu görevi yerine getiriyor. PHP, C# gibi dillerde de database sunucuya bağlanmayı sağlıyor diyebilir miyiz?

**ChatGPT:**

Harika bir çıkarım yapıyorsun ve düşünce yolun **çok mantıklı**, ancak birkaç **ince ayarla** bu tabloyu **doğru ve net hale getirebiliriz**. Aşağıda adım adım açıklayayım:

**🧠 ÖNEMLİ KAVRAM AYRIMLARI**

| **Kavram** | **Gerçek Rolü** | **Açıklama** |
| --- | --- | --- |
| **Angular** | Frontend framework | Kullanıcı arayüzü yapar, **veritabanına doğrudan erişemez**. Sadece **API ile iletişim kurar.** |
| **PHP / C#** | Backend dili | Hem API yazabilirler hem de doğrudan veritabanı işlemleri yapabilirler. |
| **JSON** | Veri formatı | Bir programlama dili değildir. Sadece veri taşıma formatıdır. (Mesela bir dosya formatı gibi: .txt, .json) |
| **Database** | Veri deposu | Bilgiler burada saklanır (örneğin kullanıcı bilgileri, ürünler). |
| **API** | Veri köprüsü | Genellikle bir backend diliyle yazılır (PHP, C#, Node.js), ve veritabanı ile frontend arasında veri taşır. |

**✅ Aslında Ne Oluyor?**

**1. Geleneksel Yapı (Monolitik):**

* **PHP / C# gibi diller**, hem kullanıcıya arayüz sunar hem de veritabanı ile konuşur.
* Arayüz + veritabanı + işlem **aynı sunucuda** yapılır.
* Örnek: PHP dosyası içinde hem HTML, hem SQL sorgusu yazarsın.

**Kullanıcı (Tarayıcı)**

**↓**

**PHP (UI + DB erişimi)**

**↓**

**Veritabanı (MySQL)**

**2. Modern Yapı (API Tabanlı):**

* **Angular gibi frontend framework'ler**, sadece arayüzdür.
* Veriye ulaşmak için bir **API'ye (örneğin PHP, C#, Node.js ile yazılmış)** istek atar.
* API de veritabanına bağlanıp veriyi döner.

**Kullanıcı (Angular)**

**↓**

**API (PHP/C#/Node)**

**↓**

**Veritabanı (PostgreSQL, MongoDB)**

**🔄 JSON Ne Yapıyor?**

* JSON, bu iletişimdeki **taşıyıcı formattır.**
* Angular → API → JSON veri
* API → Angular → JSON cevap

**{**

**"name": "Tuba",**

**"age": 25**

**}**

**✅ Şöyle Özetleyebiliriz:**

**✔ Her dil bir sunucuya ulaşmak ve veri ile çalışmak ister.**

| **Yapı** | **Sunucuya nasıl bağlanır?** | **Veritabanı ile nasıl çalışır?** |
| --- | --- | --- |
| **Angular** | API aracılığıyla | API'ye istek atar (JSON ile) |
| **PHP / C#** (geleneksel) | Doğrudan sunucuda çalışır | Veritabanına kendisi bağlanır |
| **PHP / C#** (API olarak) | Angular'dan gelen isteği karşılar | Veritabanından veri alıp JSON olarak gönderir |

**🔔 Yani demek ki:**

* Angular, doğrudan sunucuya veya veritabanına **bağlanmaz**, **API’ye** istek atar.
* PHP ve C# gibi diller, hem API yazabilir hem de doğrudan veritabanına bağlanabilir.
* JSON sadece **verinin taşındığı biçimdir**, dil değildir.
* Her şeyin sonunda **sunucu ve veritabanı vardır**, ama her yapı **onlara farklı yollarla ulaşır.**

Soru? sunucuyu kafamda canlandırmakta zorlanıyorum

**ChatGPT:**

Bu çok doğal — çünkü **sunucu** soyut bir kavramdır ama internetteki her şeyin temelini oluşturur. Gel birlikte, kafanda canlandırabileceğin şekilde basit benzetmelerle **sunucu nedir**, **nerede durur**, **ne işe yarar** sorularına cevap bulalım.

**💡 Sunucu Nedir? (En Basit Tanım)**

Bir **sunucu**, başka cihazlara (**istemcilere**) hizmet veren bir bilgisayardır.

**☕ Benzetme: Kafe ve Garson**

**👩‍💻 Sen: Angular kullanan bir kullanıcı (istemci)**

**☕ Garson: Sunucu**

**🍽️ Mutfak: Veritabanı veya bilgi kaynağı**

Sen (kullanıcı), garsona (sunucuya) "bir kahve" siparişi veriyorsun (API isteği).

* Garson bu isteği alır.
* Gider mutfaktan (veritabanı) kahveyi alır.
* Sana geri getirir (JSON veri ile).

Sunucu burada **kendisi kahve yapmaz**. Sadece seninle mutfak arasında aracı olur. İşte bu, kafanda net bir şekilde sunucunun rolüdür.

**🌐 Gerçek Hayattaki Sunucu**

* Fiziksel olarak: Yüksek hızlı, 7/24 çalışan bilgisayarlardır. (Amazon sunucuları, Google sunucuları, vs.)
* Yazılımsal olarak: Sunucuda çalışan bir program, gelen istekleri dinler. Örneğin:
  + PHP scripti
  + Node.js Express sunucusu
  + ASP.NET backend uygulaması

**🔌 Sunucu Nerede Durur?**

Sunucular genellikle:

1. **Uzak bir yerdeki veri merkezlerinde (bulut)**:
   * AWS (Amazon Web Services)
   * Azure (Microsoft)
   * Google Cloud
   * DigitalOcean
2. **Yerel bilgisayarında** (örneğin localhost ya da 127.0.0.1) – eğitim ve geliştirme için.

**👩‍🏫 Yazılımdaki Sunucu Tipleri**

| **Sunucu Türü** | **Açıklama** |
| --- | --- |
| **Web Sunucusu** | HTML, CSS, JS dosyaları dağıtır (örn: Apache, Nginx) |
| **API Sunucusu** | İstemcilerden veri isteği alır, işlem yapar, cevap verir |
| **Veritabanı Sunucusu** | SQL sorgularına cevap verir (örn: MySQL, MongoDB) |

**🎯 Örnekle Gösterelim**

Sen Angular'da şunu yazdın:

**this.http.get('https://dummyjson.com/products')**

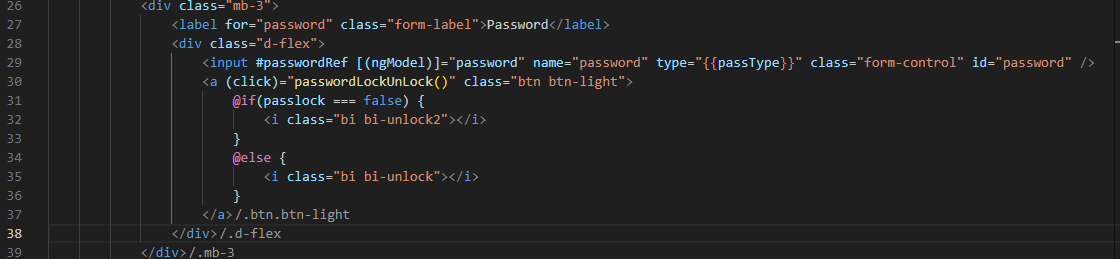
Bu ne demek?

* https://dummyjson.com → Gerçek bir **sunucuya** istek gönderiyorsun.
* O sunucu, kendi veritabanından ürünleri alıyor.
* JSON formatında sana geri gönderiyor.

**🧠 Özet Görsel (Zihninde tutmak için)**

[Tarayıcı / Angular] →→→ [SUNUCU (API)] →→→ [Veritabanı]

👈--------------- JSON verisi ---------------👈



"password” işlemleri register.html dosyasından alıntı kod

**🧠 @ işareti ne işe yarıyor?**

Angular 17 ile gelen **@-block syntax** şu amaçla kullanılır:

| @if | \*ngIf yerine geçer (daha okunaklı bir alternatif) |  
| @for | \*ngFor yerine geçer |  
| @switch | ngSwitch yerine geçer |  
| @else | Alternatif durum belirtmek için |

**🔧 Koşullar**

* Bu özellik **Angular 17 ve üstü** sürümlerde çalışır.
* standalone: true veya modern component yapısında daha çok tercih edilir.
* Template Compiler bu yapıyı destekler.

🡪 Angular'da genellikle şunları görmeye alışıksındır:

**\*ngIf="condition"**

🡪Ama senin gösterdiğin gibi:

**@if (passlock === false) {**

**...**

**} @else {**

**...**

**}** yapısı aslında **Angular 17 ile gelen** yepyeni bir özellik: **@-block syntax** (yeni şablon kontrol yapısı).

**✅ @if / @else nedir?**

Bu, Angular 17+ sürümünde gelen bir yeniliktir ve **şablon yapısını daha sade ve okunabilir** hale getirir.

**Eskiden nasıl yapılıyordu?**

**<div \*ngIf="passlock === false; else locked">**

**<i class="bi bi-unlock2"></i>**

**</div>**

**<ng-template #locked>**

**<i class="bi bi-unlock"></i>**

**</ng-template>**

🡪 Angular 17+ ile artık şöyle yazabiliyorsun:

**@if (passlock === false) {**

**<i class="bi bi-unlock2"></i>**

**} @else {**

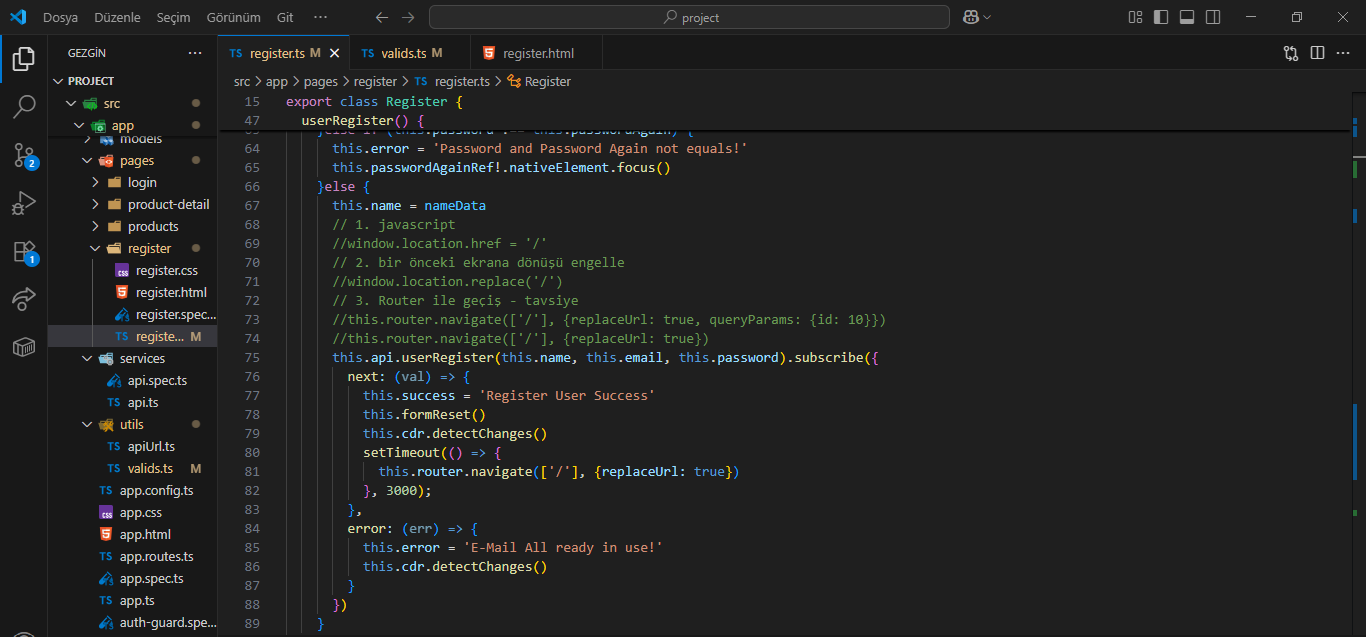
**<i class="bi bi-unlock"></i>**

**}** ✅ Bu yazım daha okunabilir, temiz ve **HTML'e daha yakın**.

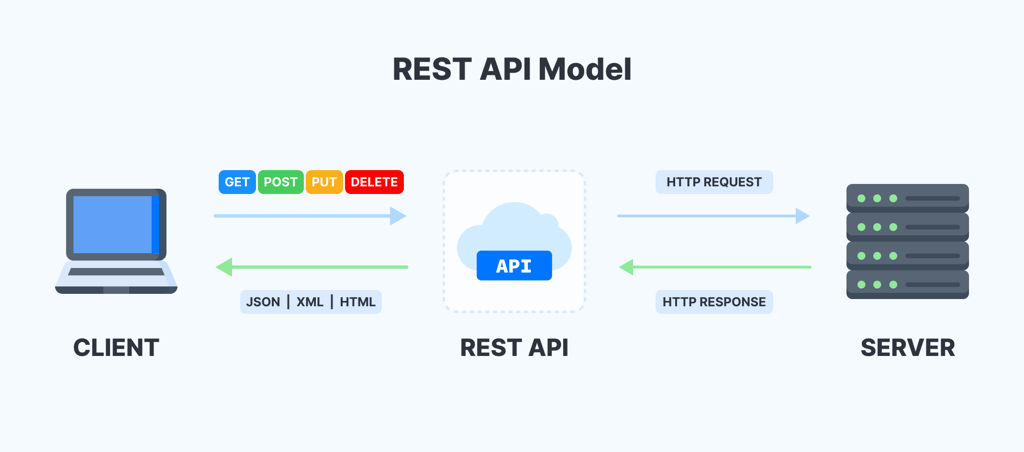
**📦 Özet**

| **Sembol** | **Anlamı** |
| --- | --- |
| @if (...) {} | Koşullu görünüm |
| @else {} | Alternatif içerik |
| @for (...) {} | Döngüyle içerik üretme |
| @ işareti | Angular 17+’nin yapısal kontrol sözdizimi başlatıcısı |

Register.ts register.html ve valids.ts dosyalarındaki notlara bak



Days\_53: input selector – OpenAPI - RestApi – injectable \*\*FrontEnd/Project---> 12/07/2025



**🧠 REST API Nedir?**

**REST**, “Representational State Transfer” ifadesinin kısaltmasıdır.  
**REST API**, istemci (frontend) ile sunucu (backend) arasında veri alışverişi yapmak için kullanılan bir web servis mimarisidir.

**🔧 Ne İşe Yarar?**

REST API, uygulamalar arasında:

* Veri alma (GET)
* Veri gönderme (POST)
* Veri güncelleme (PUT/PATCH)
* Veri silme (DELETE) gibi işlemleri **HTTP protokolü üzerinden** yapmanı sağlar.

What is rest API and how to use it in Python


**📦 Basit Bir Örnek:**

Diyelim ki ürünleri yöneten bir API var:

| **İşlem** | **HTTP Metod** | **URL** | **Açıklama** |
| --- | --- | --- | --- |
| Ürünleri listele | GET | /products | Tüm ürünleri getir |
| Ürün getir | GET | /products/1 | ID'si 1 olan ürünü getir |
| Ürün ekle | POST | /products | Yeni ürün oluştur |
| Ürün güncelle | PUT | /products/1 | ID'si 1 olan ürünü güncelle |
| Ürün sil | DELETE | /products/1 | ID'si 1 olan ürünü sil |

**🔄 REST API Nasıl Çalışır?**

1. **İstemci** (örneğin Angular uygulaman):
   * GET /products isteği gönderir.
2. **Sunucu** (örneğin bir Node.js, .NET, Java API):
   * Bu isteği alır, veri tabanından ürünleri çeker.
   * JSON formatında cevap döner:

**{ "id": 1, "name": "Kalem", "price": 15 },**

**{ "id": 2, "name": "Defter", "price": 25 }**

**]**

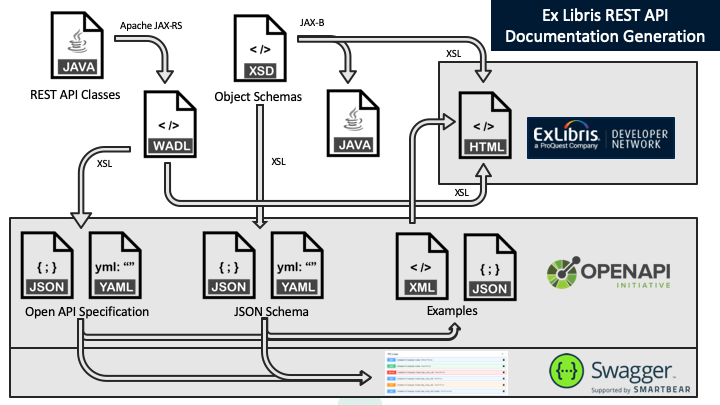
**🔐 REST API'nin Özellikleri**

| **Özellik** | **Açıklama** |
| --- | --- |
| HTTP protokolü kullanır | GET, POST, PUT, DELETE vs. |
| Kaynak temellidir | Her şey bir "resource" (örnek: /users, /products) |
| JSON verisi döner | En yaygın veri biçimi JSON'dur |
| Stateless (durumsuz) | Her istek bağımsızdır, oturum saklamaz |
|  |  |

**🎯 Özetle**

**REST API**, uygulamalar arasında veri alışverişi yapmanı sağlayan bir web servis türüdür. HTTP metotlarını kullanarak çalışır ve genellikle JSON formatında veri gönderip alır.

**OpenAPI Specification (OAS)** Nedir? (SWAGGER)



Modern web API'lerini Uluslararası standart bir formatta tanımlamak için kullanılan bir belgedir. Eski adıyla **Swagger Specification** olarak da bilinir.

**🧠 Basit Tanımıyla:**

**OpenAPI**, bir REST API'nin **ne yaptığını, hangi endpoint'lere sahip olduğunu ne tür veri beklediğini ve ne döndürdüğünü** tanımlayan bir sözleşmedir (contract).

Bu belge genelde JSON veya YAML formatında yazılır ve API geliştiricileri ile kullanıcıları için bir **anlaşma noktası** oluşturur.

**🔍 Ne İşe Yarar?**

| **Amaç** | **Açıklama** |
| --- | --- |
| 🔎 **API'yi belgelemek** | Kullanıcılar, API’nin ne işe yaradığını açıkça görür. |
| 🤝 **Frontend-backend iş birliği** | Frontend geliştirici, API hazır olmadan sadece dokümana bakarak işini yapabilir. |
| ⚙️ **Otomatik kod üretimi** | OpenAPI'den otomatik olarak Angular, React, Java, C# gibi dillerde istemci (client) kodu üretilebilir. |
| 🧪 **Test ve mock veri oluşturma** | Swagger UI üzerinden canlı testler yapılabilir. |
| 🔐 **Güvenlik, doğrulama** | Yetkilendirme şemaları ve veri tipleri net tanımlanır. |

**🎯 Swagger Neden Önemlidir?**

**Swagger**, yani yeni adıyla **OpenAPI**, REST API'lerini **tanımlamak**, **belgelemek**, **test etmek** ve hatta **otomatik kod üretmek** için kullanılan **standart bir sistemdir**. Özellikle büyük projelerde frontend ve backend ekiplerinin birlikte düzgün çalışabilmesini sağlar.

**🧩 Swagger Ne Sağlar?**

| **Özellik** | **Açıklama** |
| --- | --- |
| 📄 **Otomatik Dokümantasyon** | API endpoint'lerini, parametrelerini ve dönüş tiplerini otomatik gösterir. |
| 🧪 **Canlı API Testi** | Swagger UI sayesinde tarayıcı üzerinden doğrudan API'yi test edebilirsin. |
| 🤝 **Frontend-Backend Uyum** | API dokümantasyonu hazır olduğu için frontend geliştirici backend’i beklemeden geliştirme yapabilir. |
| ⚙️ **Otomatik Kod Üretimi** | Swagger/OpenAPI dokümanından Angular, React, Java, C# gibi client SDK’lar üretilebilir. |
| 📌 **Versiyon Takibi ve Geliştirici İletişimi** | Hangi endpoint ne işe yarıyor, hangi parametre gerekiyor net olarak görülür. |

**🖥️ Swagger UI Nedir?**

**Swagger UI**, Swagger/OpenAPI dosyasını (genelde swagger.json ya da openapi.yaml) alır ve görsel, interaktif bir API arayüzü oluşturur:

Bu arayüz üzerinden:

* API çağrısı yapabilir,
* Yanıtları görebilir,
* Deneme verileri gönderebilirsin.

**🔧 Nerede Kullanılır?**

* API geliştiricileri → backend'deki endpoint'leri tanımlar.
* Frontend geliştiricileri → bu dokümana bakarak veri yapısını öğrenir.
* QA / Test ekipleri → test senaryolarını Swagger üzerinden çalıştırabilir.
* Mobil / üçüncü taraf uygulamalar → API entegrasyonunu Swagger’dan yararlanarak yapar.

**🎓 Gerçek Dünya Örneği**

Düşün ki bir e-ticaret sitesinde ürünleri listeleyen, sepete ekleyen, ödeme alan bir API var.  
Bu API’nin Swagger belgesi varsa:

✅ Frontend geliştirici,  
✅ Mobil uygulama geliştirici,  
✅ Test mühendisi  
— hepsi bu tek dokümandan faydalanabilir.

**🚀 Özetle**

**Swagger, API'yi şeffaf, erişilebilir ve yönetilebilir hale getirir.**

✅ Otomatik dokümantasyon  
✅ Canlı test  
✅ Kod üretimi  
✅ Ekipler arası iletişim

\*\*\* 🡪Herhangi bir projede SERVİS NASIL KURULUR? 1.35 DAKİKADA ANLATILIYOR!

Terminalde: ng g s services/api yazılır

### Days\_54: Tekrar – userRegister/api işlemleri \*\*FrontEnd/Project---> 13/07/2025

**🔁 Observable Nedir**?

Bu olay eş zamansızlığı getirir yani tetiklendiği zaman kendinden sonraki kodu bloklamayan program içerisinde farklı bir yapı gibi çalışır. Özellikle gecikme ihtimali olan yapılarda eş zamansızlık vardır. Ve kendi içinde a-senkron olduğu için ayrıca kod yazımında belirtmeye gerek yoktur.

**Observable**, RxJS (Reactive Extensions for JavaScript) kütüphanesiyle gelen, **zaman içinde birden fazla veri yayınlayabilen** bir yapıdır.

**🎯 Özellikleri:**

* Birden fazla değer yayınlayabilir.
* Yayınlar zaman içinde gerçekleşebilir (asenkron olabilir).
* Abone olman gerekir (subscribe()).
* Yayını durdurmak için unsubscribe() yapılabilir.
* Angular'da özellikle HttpClient, Forms, Events gibi sistemler Observable döndürür.

**🧪 Örnek:**

**import { Observable } from 'rxjs';**

**const myObs = new Observable((observer) => {**

**observer.next('Birinci veri');**

**observer.next('İkinci veri');**

**setTimeout(() => {**

**observer.next('Üçüncü veri');**

**observer.complete();**

**}, 1000);**

**});**

**myObs.subscribe((data) => {**

**console.log(data);**

**});**

*🔹 Çıktı:*

*Birinci veri*

*İkinci veri*

*(1 saniye sonra)*

*Üçüncü veri*

**🔐 Promise Nedir?**

**Promise**, JavaScript’in yerleşik bir yapısıdır. **Gelecekte bir kez sonuç döndürecek bir işlemi temsil eder** (başarılı ya da hatalı).

**🎯 Özellikleri:**

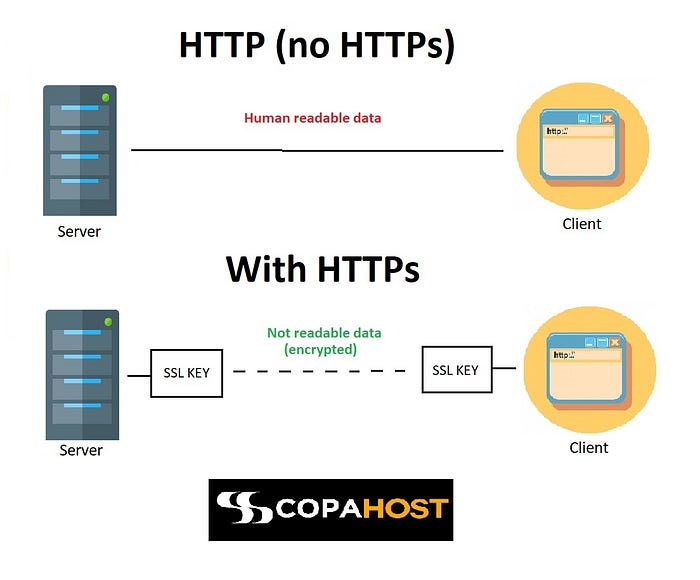
* Sadece bir kere değer döndürür (resolve veya reject).
* then ve catch ile kullanılır ya da async/await ile.
* Observable gibi iptal edilemez veya birden fazla değer gönderemez.

**📊 Karşılaştırma Tablosu**

| **Özellik** | **Observable** | **Promise** |
| --- | --- | --- |
| Veri sayısı | Birden fazla | Sadece bir kez |
| İptal edilebilir mi? | Evet (unsubscribe()) | Hayır |
| Zamanla veri yayımı | Evet (akış şeklinde) | Hayır (tek sefer) |
| Angular kullanımı | Çok yaygın (HttpClient, FormControl...) | Daha az |
| await ile kullanımı | lastValueFrom() gibi dönüşüm gerekir | Doğrudan await ile kullanılabilir |
| Kütüphane | RxJS (ekstra) | Yerleşik (JavaScript) |

**🔚 Özet**

* Observable: Zamanla birçok veri üretebilir. Yaygın olarak Angular’da kullanılır. Daha güçlü ve kontrol edilebilir.
* Promise: Sadece tek sonuç döndürür. Daha basit ama esnek değildir.



Yazılımda **"araya giren adam"** (İngilizcesi: **Man-in-the-Middle**, kısaca **MITM**) kavramı, bir **siber güvenlik saldırı türünü** ifade eder. Bu saldırıda, iki taraf arasındaki iletişim **onların haberi olmadan üçüncü bir kişi tarafından ele geçirilir, izlenir veya değiştirilir**.

**🕵️‍♂️ Araya Giren Adam (MITM) Saldırısı Nedir?**

**Tanım:**  
İki taraf (örneğin bir kullanıcı ile bir banka web sitesi) arasında veri aktarılırken, saldırgan bu iletişim kanalına gizlice girer ve verileri okur, değiştirir ya da çalar.

**🎯 Amaçlar:**

* Şifreleri, kredi kartı bilgilerini, kimlik bilgilerini çalmak
* Verileri gizlice değiştirmek
* Kimliğe bürünmek (saldırgan, biriymiş gibi davranır)

**🧱 MITM Saldırısı Nasıl Olur?**

**🧑‍💻 Senaryo:**

1. Kullanıcı → Banka web sitesine giriş yapmak istiyor.
2. Saldırgan, ağ bağlantısını (örneğin açık Wi-Fi) dinleyerek araya giriyor.
3. Kullanıcının gönderdiği giriş bilgilerini görebiliyor veya değiştiriyor.

**🔓 MITM Saldırısı Yöntemleri**

| **Yöntem** | **Açıklama** |
| --- | --- |
| **Ağ dinleme (Packet Sniffing)** | Açık Wi-Fi gibi yerlerde veri paketleri dinlenir. |
| **DNS Spoofing** | Kullanıcıyı sahte bir siteye yönlendirir. |
| **HTTPS yerine HTTP zorlaması** | Güvenli bağlantı yerine güvensiz bağlantıya yönlendirme yapılır. |
| **SSL Strip** | HTTPS bağlantı açıldığında, aradaki adam bu bağlantıyı HTTP'ye çevirir. |
| **IP Spoofing** | Saldırgan, güvenilen bir sistem gibi davranır. |

**🔐 Nasıl Korunuruz?**

✅ **HTTPS** kullanmak  
✅ **SSL sertifikalarını doğrulamak**  
✅ **VPN kullanmak** (özellikle açık Wi-Fi'da)  
✅ **Güvenilir ağlar dışında dikkatli olmak**  
✅ **Public Wi-Fi'da asla banka, e-devlet gibi sitelere girmemek**  
✅ **JWT, OAuth gibi güvenli oturum yönetimi sistemleri kullanmak**

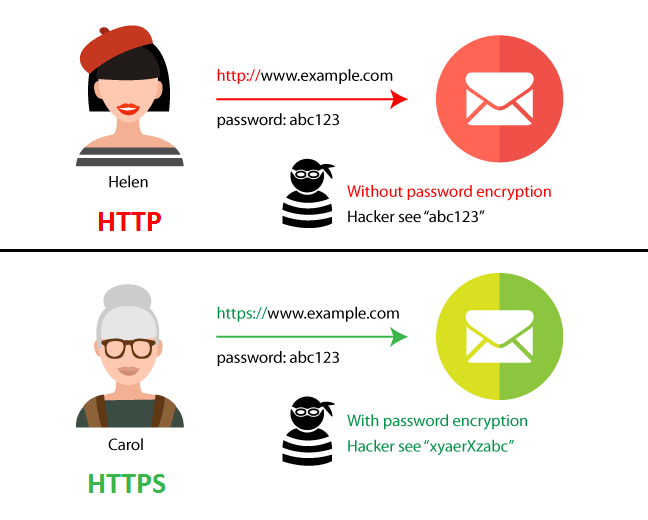
**👨‍💻 Yazılımcılar İçin Not**

Eğer bir **frontend/backend** geliştiriciysen, MITM saldırılarına karşı:

* API’lerde **HTTPS kullan**
* Token’ları (JWT vs.) tarayıcıda **HttpOnly Cookie** olarak saklamayı düşün
* CORS ve CSRF ayarlarını doğru yap
* İletişim trafiğini şifrele

**🧠 Özet**

**"Araya giren adam" saldırısı**, iki tarafın iletişimini gizlice dinleyen veya değiştiren üçüncü bir kişinin yaptığı bir siber saldırı türüdür.  
Amaç: veri çalmak, kimlik taklidi yapmak veya bilgiye zarar vermektir.



**HTTPS (Hypertext Transfer Protocol Secure)**

HTTPS, HTTP protokolünün güvenli bir sürümüdür. HTTPS, iletişim sırasında verilerin şifrelenmesini sağlar, böylece verilerin güvenli bir şekilde iletilmesini ve üçüncü tarafların erişimine karşı korunmasını sağlar. Bu protokol SSL (Secure Sockets Layer) veya TLS (Transport Layer Security) adı verilen bir güvenlik protokolü kullanır. HTTPS’nin en belirgin özelliği, veri şifreleme sürecidir. İstemci (web tarayıcısı) ve sunucu arasındaki iletişim, genel ve özel bir anahtar kullanarak şifrelenir. Bu, verilerin iletilirken şifrelenmesini ve yalnızca doğru alıcı tarafından çözülebilmesini sağlar. Bu sayede, bir kişi veya kötü niyetli bir saldırgan ağ üzerinden iletilen verilere eriştiğinde, şifrelenmiş verileri anlaması veya manipüle etmesi çok zordur, bunun yanı sıra bu HTTPS kullanan web sitelerinin yüzde yüz güvenli olduğunu ve verilerin erişilemez olduğu anlamına gelmiyor zira bilişim dünyasında hiçbir şey imkansız değildir.

[https://transform.tools/](https://transform.tools/%20%20)  sitesi ne için kullanılır?

Başak bir alternatif site de <https://app.quicktype.io/>

Bu site, geliştiricilerin bir kod ya da veri formatını başka bir formata dönüştürmelerine yardımcı olan çevrim içi bir "polyglot converter" aracıdır. Örneğin:

* JSON → TypeScript tipi
* TypeScript → Flow, düz JavaScript vs. vs.

**🔄 Neleri Dönüştürebilir?**

* **JSON**: YAML, OpenAPI, TypeScript tanımına
* **TypeScript**: Flow, JavaScript veya Zod şeması

**🎯 Kullanım Amacı**

* Farklı diller/formats arası pratik kod geçişleri sağlamak
* Bir formatı başka bir projede (örneğin React JSX veya Tailwind CSS) kullanmaya uygun hale getirmek
* Otomatik dönüşüm sayesinde zaman kazanmak

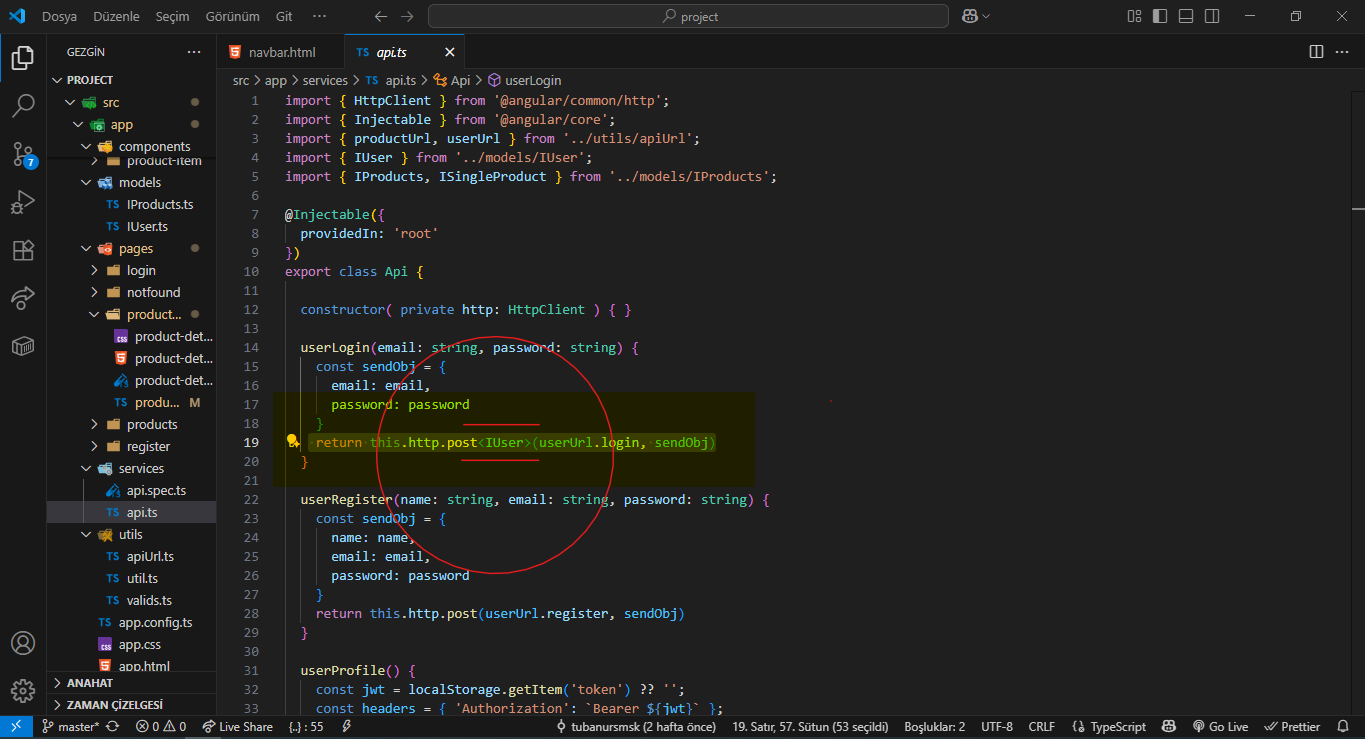
**🧠 Kimler İçin Faydalı?**

* **Frontend geliştiriciler**: CSS → Tailwind dönüşümleri, HTML → JSX dönüşümleri
* **Backend/data mühendisleri**: JSON → GraphQL, OpenAPI, TypeScript tip tanımı gibi dönüşümler
* **Tüm yazılımcılar**: Farklı formatlar arasında hızlı dönüşümlerle zamandan tasarruf etmek isteyenler

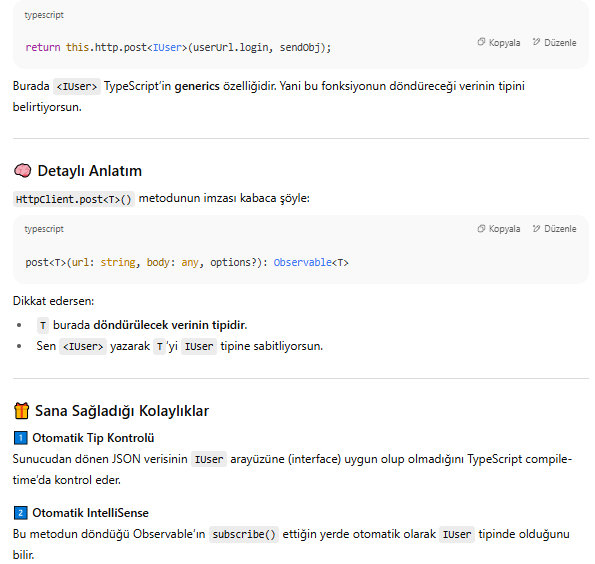
**✅ Özet**

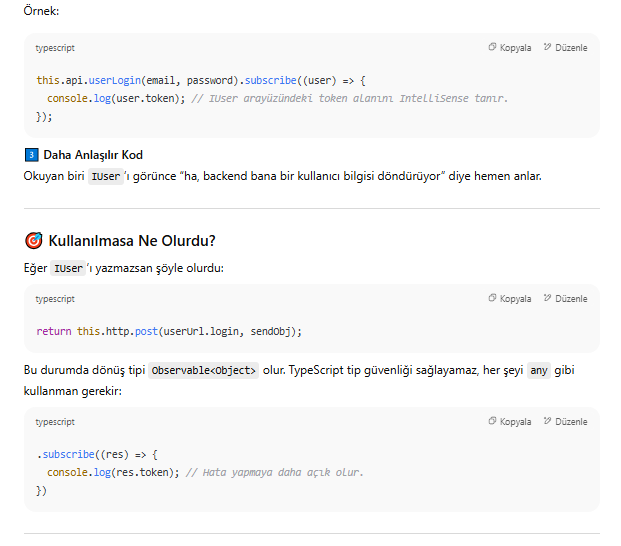
**transform.tools**, çok sayıda **kod ve veri formatı** arasında hızlıca dönüştürme yapabilen bir **web tabanlı araçtır**. Kod taşıma, API veya frontend framework entegrasyonu gibi işler için oldukça pratik bir kaynak sunar.

🡪Mesela aldığı json formatını to TypeScript formatına kolayca çevirir…



<IUser> işlevi:





**✅ Özet: <IUser> burada şunu sağlar:**

* Dönen cevabın tipini garanti eder.
* Kod yazarken otomatik tamamlama (IntelliSense) çıkarır.
* Hataları daha derleme aşamasında fark ettirir.
* Kodun okunabilirliğini artırır.

Bu yüzden Angular’da **HTTP isteklerinde response tipi** mutlaka tanımlanır.

**🛡️ OWASP Nedir?**

**OWASP**, İngilizce açılımı **Open Web Application Security Project** olan,  
✅ **açık kaynaklı**,  
✅ **kar amacı gütmeyen**,  
✅ **web uygulama güvenliği** alanında dünyada en çok kabul gören topluluktur.

Kısaca:  
🔍 *Web uygulamalarının daha güvenli olmasını sağlamak için kılavuzlar, araçlar, dokümantasyon ve topluluk desteği sunan bir organizasyon.*

Web sitesi 👉 <https://owasp.org/>

**🎯 OWASP’ın Amacı**

* Web uygulamalarının güvenliğini **standartlaştırmak**
* Geliştiricilere, test uzmanlarına ve kurumlara rehberlik sağlamak
* Güvenlik açıklarının nasıl keşfedileceğini ve önleneceğini öğretmek

**🧠 En Çok Bilinen Çalışmaları**

**🔟 OWASP Top 10**

Web uygulamalarındaki en kritik 10 güvenlik riski listesidir.  
Her birkaç yılda bir güncellenir.  
Yazılımcılar için **birincil başvuru kaynağı**dır.

**🧩 OWASP Neden Önemli?**

✅ Web uygulama güvenliğinde **sektör standardıdır**.  
✅ Dünya çapında yaygın kabul görür.  
✅ Yasal ve uyumluluk gereksinimlerinde (ISO, PCI DSS, SOC2) OWASP Top 10 referans alınır.  
✅ Şirketler kodlarını OWASP Top 10’a göre test eder.  
✅ Yazılımcılara güvenli kod yazmayı öğretir.

**🎓 Kısaca Özet**

**OWASP = Web uygulamalarını güvenli geliştirmek için başucu kaynağı ve topluluk.**

Eğer yazılım geliştiriyorsan, OWASP Top 10’u öğrenmen **kesinlikle tavsiye edilir**.

**OWASP Top 10** listesini *son sürüm* (2021) Bu liste, dünyada en sık görülen ve en kritik web güvenlik risklerini kapsar:



**🛡️ OWASP Top 10 (2021)** En son sürümü 👉 <https://owasp.org/www-project-top-ten/>

**1️⃣ Broken Access Control**

🔹 **Yetki kontrolü hataları**

* Kullanıcı, erişmemesi gereken kaynağa erişir.
* Örnek: Bir kullanıcı başka bir kullanıcının profilini görebilir.
* Çözüm: Sıkı erişim kontrolleri ve testler.

**2️⃣ Cryptographic Failures**

🔹 **Şifreleme hataları**

* Hassas veriler (şifre, kredi kartı, kişisel bilgi) iyi korunmaz.
* Örnek: HTTPS yerine HTTP kullanmak.
* Çözüm: Modern şifreleme algoritmaları, TLS.

**3️⃣ Injection**

🔹 **En klasik ve tehlikeli açık**

* Kullanıcı girdileriyle veri tabanına veya başka sistemlere komut enjekte edilir.
* Örnek: SQL Injection, NoSQL Injection.
* Çözüm: Parametrik sorgular, input doğrulama.

**4️⃣ Insecure Design**

🔹 **Baştan yanlış tasarım**

* Güvenlik göz ardı edilerek sistem tasarlanır.
* Örnek: Güvensiz parola resetleme mekanizması.
* Çözüm: Tehdit modelleme, güvenlik mimarisi.

**5️⃣ Security Misconfiguration**

🔹 **Yanlış yapılandırmalar**

* Varsayılan şifreler, yanlış izinler, hatalı başlıklar.
* Örnek: Sunucunun debug modu açık kalması.
* Çözüm: Düzenli konfigürasyon denetimleri.

**6️⃣ Vulnerable and Outdated Components**

🔹 **Güvensiz kütüphaneler**

* Eski yazılım bileşenlerini kullanmak.
* Örnek: Güvenlik açığı olan jQuery sürümü.
* Çözüm: Sürekli güncelleme, bağımlılık yönetimi.

**7️⃣ Identification and Authentication Failures**

🔹 **Kimlik doğrulama sorunları**

* Kullanıcı doğrulama zafiyetleri.
* Örnek: Zayıf parola politikaları.
* Çözüm: Çok faktörlü kimlik doğrulama.

**8️⃣ Software and Data Integrity Failures**

🔹 **Yazılım ve veri bütünlüğü eksikliği**

* Kod veya veri manipülasyonu.
* Örnek: Paketlerin doğrulanmadan kurulması.
* Çözüm: Dijital imza, supply chain güvenliği.

**9️⃣ Security Logging and Monitoring Failures**

🔹 **Yetersiz loglama ve izleme**

* Saldırılar fark edilmez.
* Çözüm: Olay kaydı, alarmlar, izleme sistemleri.

**🔟 Server-Side Request Forgery (SSRF)**

🔹 **Sunucu tarafı istek sahteciliği**

* Saldırgan, sunucu üzerinden başka sistemlere istek attırır.
* Örnek: Metadata sunucusundan veri sızdırmak.
* Çözüm: İstek filtreleme, whitelist kullanımı.

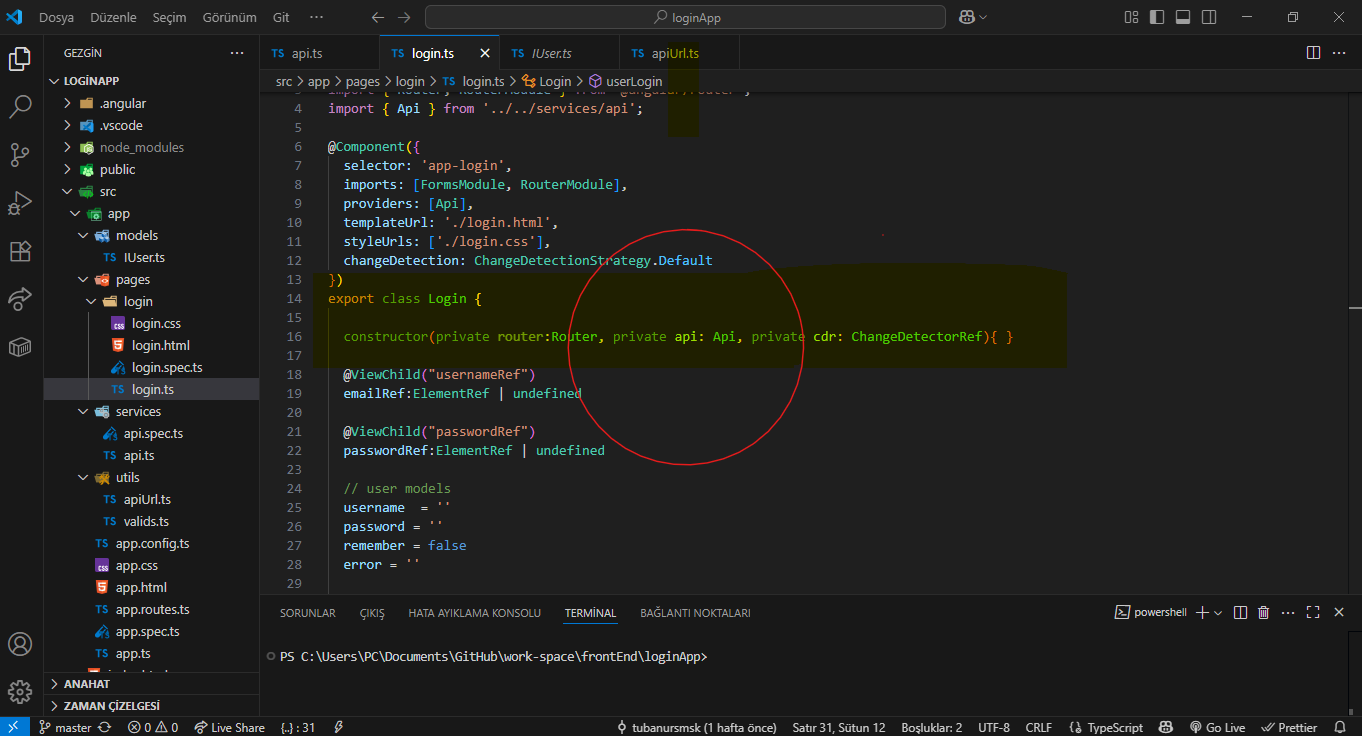
**📌 Özet Tablon**

| **#** | **Başlık** | **Kısaca** |
| --- | --- | --- |
| 1 | Broken Access Control | Yetki kontrol hataları |
| 2 | Cryptographic Failures | Şifreleme eksikleri |
| 3 | Injection | Girdi enjekte edilmesi |
| 4 | Insecure Design | Yanlış sistem tasarımı |
| 5 | Security Misconfiguration | Hatalı yapılandırmalar |
| 6 | Vulnerable Components | Eski güvensiz bileşenler |
| 7 | Auth Failures | Kimlik doğrulama sorunları |
| 8 | Integrity Failures | Kod ve veri bütünlüğü zafiyeti |
| 9 | Logging Failures | Yetersiz izleme ve kayıt |
| 10 | SSRF | Sunucu istek manipülasyonu |

🔍 **Neden öğrenmelisin?**

* Güvenli yazılım geliştirmek için en temel rehberdir.
* Birçok kurum, OWASP Top 10’a uyum zorunluluğu getirir.
* Yazılımda bilinçli olursan güvenlik açıklarını en aza indirirsin.

### Days\_55: tekrar – product sayfası tasarlama \*\*FrontEnd/Project---> 20/07/2025



Çizdiğim **private api: api** kodunu neden constructer içinde yazmak gerekli bir alt satıra tek yazsam neden olmuyor?

🟢 1️⃣ Burada constructor() içine yazılan ifade: **private api: Api** şu anlama gelir:

✅ Bir **sınıf üyesi (property)** oluştur.  
✅ Tipi Api olsun.  
✅ Angular Dependency Injection sistemi bu Api servisini otomatik versin.

**🟡 2️⃣ “Constructor dışında alt satıra yazsam olmuyor” dediğin durum**

Örnek olarak şu şekilde yazarsan:

**export class Login {**

**private api: Api;**

**constructor() { }**

**}**

Bu sefer **Dependency Injection yapmamış oluyorsun**.

Sen burada sadece:

"Bir api özelliğim var, tipi Api" demiş oluyorsun.

Ama **nasıl geleceğini**, **Angular’ın bu servisi otomatik vereceğini** söylememiş oluyorsun.

**🟣 3️⃣ Peki Dependency Injection nasıl çalışıyor?**

Angular constructor()’a bakar:

* Oradaki parametrelerin **tiplerini okur** (Api)
* Oradaki erişim belirleyiciyi (private, public) kullanarak **otomatik sınıf özelliği** oluşturur
* Uygulama başladığında Api servisini hazırlar ve verir

Yani constructor’da tanımlamazsan **enjekte edilmez**, undefined olur.

**🧠 4️⃣ Şu yapı ile eşdeğerdir**

Sen şunu yazdığında**:**

**constructor(private api: Api) {}**

**Angular şu işlemi otomatik yapar:**

**typescript**

**KopyalaDüzenle**

**private api: Api;**

**constructor(api: Api) {**

**this.api = api;**

**}**

Bu yüzden constructor’da tanımlamazsan, Angular **senin yerine o atamayı yapmaz**.

**🧩 Nedir Bu Dependency Injection (DI)?**

**Dependency Injection** kabaca şu demek:

Bir sınıfın ihtiyaç duyduğu bağımlılıkları (ör. servisleri) **sen elle new ile yaratmak yerine** Angular’ın otomatik oluşturup sınıfa vermesi.

Yani:

* Senin Api servisine ihtiyacın var.
* Angular uygulaması başlarken bu servisi **tek sefer yaratır (singleton)**.
* O servisi sana otomatik yollar.

**🛠️ Klasik JavaScript’te nasıldı?**

Normalde saf JS’de bir sınıf başka sınıfı kullanmak istese şöyle yaparsın:

**class MyClass {**

**constructor() {**

**this.api = new Api();**

**}**

**}**

Burada Api’yi  **sen oluşturuyorsun**.  
Bu:

* Test etmesi zor,
* Yönetmesi zor,
* Tekrarlı,
* Çoğu framework’te tavsiye edilmeyen bir yöntemdir.

**🟢 Angular’da Dependency Injection Ne Sağlar?**

✅ Angular Api servisini **uygulama başlarken tek sefer yaratır**.  
✅ Tüm ihtiyaç duyan sınıflara otomatik **aynı örneği (singleton) verir**.  
✅ Sen new Api() yazmak zorunda kalmazsın.  
✅ Kodun test edilebilir, sürdürülebilir olur.

**🔍 Nasıl Çalışıyor?**

Sen constructor’a şöyle yazıyorsun: 🡪**constructor(private api: Api) { }**

Bu satırda **Angular şu işlemleri yapar**:

1️⃣ Uygulamanın modülünde Api servisinin @Injectable() işareti olduğuna bakar.  
2️⃣ providedIn: 'root' olduğu için uygulama başladığında **bir kez yaratır**.  
3️⃣ Login bileşenini (Login component) oluştururken constructor’ına otomatik Api nesnesini gönderir.  
4️⃣ private api: Api yazarak aynı anda:

* Bir **sınıf özelliği** (this.api) tanımlamış olursun.
* this.api artık kullanılabilir hale gelir ve new Api() yapmana gerek kalmaz.

**✨ Örnek Akış**

**Senin component’in:**

**export class Login {**

**constructor(private api: Api) { }**

**}**

**Angular çalışma zamanı:**

✅ Api servisi yaratılır:

**const apiInstance = new Api();**

✅ Login bileşeni yaratılır:

**const loginComponent = new Login(apiInstance);**

**Senin hiç farkında olman gerekmez.**  
Sadece this.api diyerek servisi kullanırsın.

**🧠 Neden private yazıyoruz?**

Şöyle bir yazarsan:

**constructor(api: Api) { }**

Artık bir sınıf özelliği oluşmaz.  
this.api diye çağırmaya çalışırsan hata alırsın.

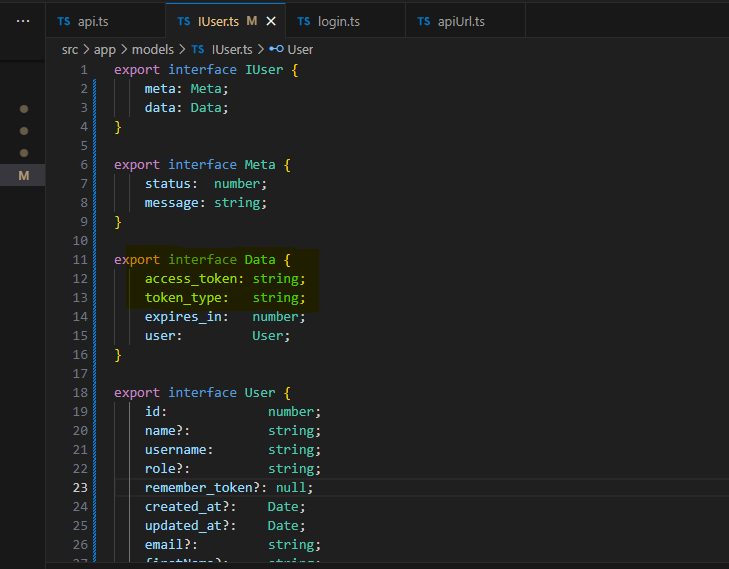
private api: Api dediğinde:

* this.api property’sini **Angular senin için tanımlar.**
* Hem injection yapılır, hem de sınıf içinde erişilebilir olur.

**🎯 Özet**

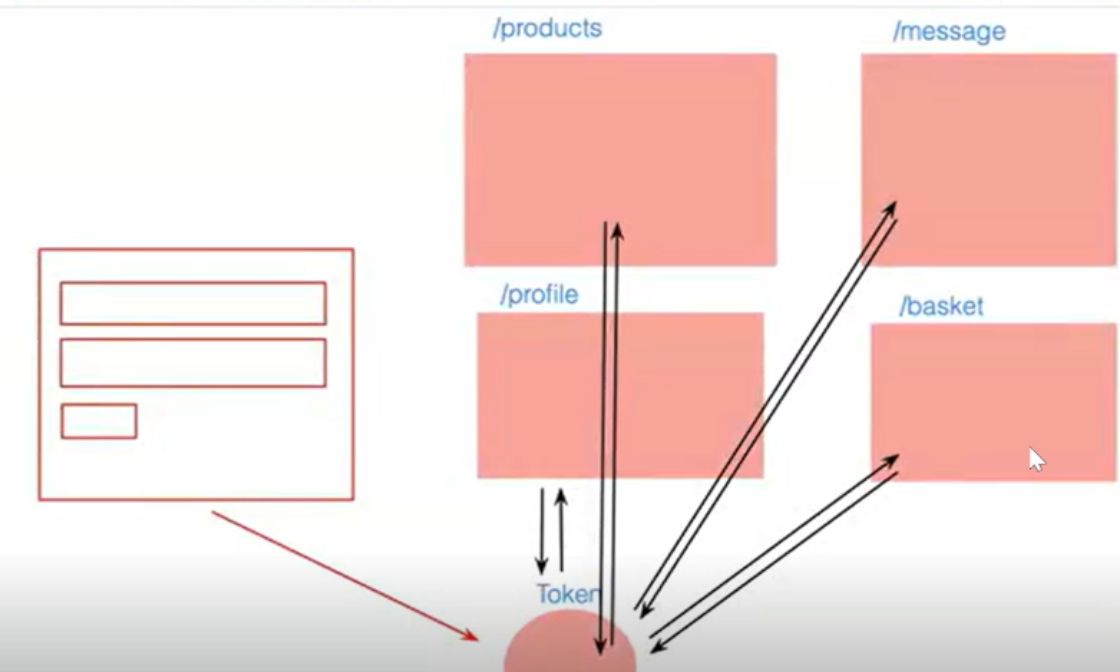
✅ **Dependency Injection** = "Bağımlılıkları otomatik oluştur ve ver."  
✅ constructor(private api: Api) yazmak = "Bana Api servisinin örneğini otomatik olarak sağla ve bu sınıfın içinde this.api diye kullanayım."

**📌 1️⃣ Access Token Nedir?**



**Access Token**, bir kullanıcının sisteme giriş yaptıktan sonra hangi kaynaklara erişebileceğini gösteren dijital bir anahtardır.  
Özetle:  
✅ Kullanıcının kimliğini ispat eder.  
✅ Kullanıcıya hangi işlemlerin izinli olduğunu belirtir.  
✅ Genelde JWT (JSON Web Token) formatında olur.  
✅ Sunucu tarafından üretilir ve istemciye (tarayıcı veya mobil uygulama) gönderilir.  
✅ Daha sonra yapılan API isteklerinde Authorization başlığına eklenir:

**Authorization: Bearer <access\_token>**



**📌 2️⃣ Senin Kodundaki Yapı**

**interface Data**

Şu alanlar var:

* access\_token: string;  
  Kullanıcı oturum açtığında backend’in ürettiği token (anahtar).
* token\_type: string;  
  Genelde "Bearer" olur. Yani *Bearer token* ile taşınır.
* expires\_in: number;  
  Token’ın kaç saniye geçerli olduğunu gösterir (ör. 3600 saniye = 1 saat).
* user: User;  
  Token ile ilişkili kullanıcı bilgileri.

Bu yapı, sunucudan dönen JSON cevabı temsil eder. Örnek bir cevap şöyle olabilir:

**{**

**"meta": {**

**"status": 200,**

**"message": "Login successful"**

**},**

**"data": {**

**"access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",**

**"token\_type": "Bearer",**

**"expires\_in": 3600,**

**"user": {**

**"id": 5,**

**"username": "john\_doe",**

**"email": "john@example.com"**

**}**

**}**

**}**

**📌 3️⃣ Access Token İşleyişi (Adım Adım)**

1. **Kullanıcı giriş yapar.**
   * Kullanıcı adı ve şifre backend’e gönderilir.
2. **Backend doğrular.**
   * Şifre doğruysa token üretir.
3. **Token döner.**
   * Yukarıdaki access\_token, token\_type, expires\_in bilgilerinin olduğu JSON gönderir.
4. **Frontend saklar.**
   * Angular uygulaman token’ı localStorage veya sessionStorage içine kaydeder:

**localStorage.setItem('token', access\_token);**

1. **Korunan isteklere token eklenir.**
   * Örneğin profil bilgisi çekeceğinde header’a eklenir:

**const headers = { Authorization: `Bearer ${token}` };**

**this.http.get('/api/user/profile', { headers });**

1. **Backend token’ı doğrular.**
   * İmzasını kontrol eder, süresini kontrol eder.
   * Token geçerliyse isteği kabul eder.
   * Token geçersizse 401 Unauthorized döner.
2. **Token süresi dolarsa yeniden giriş gerekir.**

**📌 4️⃣ Access Token ile Refresh Token Farkı**

* **Access Token**
  + Kısa ömürlüdür. (Örn. 1 saat)
  + Her istekte kullanılır.
* **Refresh Token**
  + Daha uzun ömürlüdür. (Örn. 1 hafta)
  + Access Token süresi dolunca yeni token almak için kullanılır.
  + Genelde backend’den gelen cevapta ayrıca döner.

Senin örneğinde sadece access\_token tanımlı, *refresh\_token* yok.

**📌 5️⃣ Özet**

Bu interface:

* Kullanıcının giriş yaptıktan sonra tüm yetki bilgilerini kapsayan veriyi tanımlar.
* Access Token, tüm korumalı API çağrılarının anahtarıdır.
* Token’ı header’da taşımak zorundasın.
* Süresi dolarsa login’i yenilemelisin.

**📌 localStorage Nedir?**

JavaScript’te tarayıcı üzerinde **kalıcı veri saklamak** için kullanılan bir yapıdır.

🔹 Özellikleri:  
✅ Tarayıcı kapatılsa bile veriler silinmez.  
✅ Anahtar-değer (key-value) şeklinde veri saklar.  
✅ Her şey *string* olarak depolanır.  
✅ Kapasitesi genelde 5-10 MB civarındadır.

**📌 Kullanımı**

Veri Kaydetme

**localStorage.setItem("kullaniciAdi", "tuba");**

**Anahtar: kullaniciAdi  
Değer: "tuba"**

Veri Okuma

**const ad = localStorage.getItem("kullaniciAdi");**

**console.log(ad); // "tuba"**

Veri Silme

**localStorage.removeItem("kullaniciAdi");**

Tüm Verileri Temizleme

**localStorage.clear();**

**📌 Access Token İçin Kullanım**

**localStorage.setItem("token", access\_token);**

Böylece kullanıcı token’ını tarayıcıda saklayıp, sonraki isteklerde header’a ekleyebilirsin.

Eğer **oturumu kapatınca silinsin** istiyorsan sessionStorage kullanırsın, o da benzer ama tarayıcı sekmesi kapatıldığında veriler kaybolur.

📌 Header kasıt Nedir?

Bir HTTP isteği gönderirken (mesela fetch, HttpClient, axios), **header** dediğimiz kısım, isteğin üst bilgi bölümüdür.  
Header’da sunucuya ekstra bilgiler gönderirsin:

✅ Kimlik doğrulama bilgisi  
✅ İçeriğin tipi (Content-Type)  
✅ Dil bilgisi vb.

Örneğin **JWT access token** kullandığında, header’a şöyle bir bilgi koyarsın:

**Authorization: Bearer <access\_token>**

Sunucu gelen isteğin header’ına bakar, orada geçerli bir token var mı kontrol eder.

**📌 Örnek: Angular HttpClient ile Token Header’a Ekleme**

Senin access\_token’ını localStorage’dan alıp header’a ekleyip istek atmak:

**const token = localStorage.getItem("token");**

**this.http.get("https://api.site.com/products", {**

**headers: {**

**Authorization: `Bearer ${token}`**

**}**

**}).subscribe(res => {**

**console.log(res);**

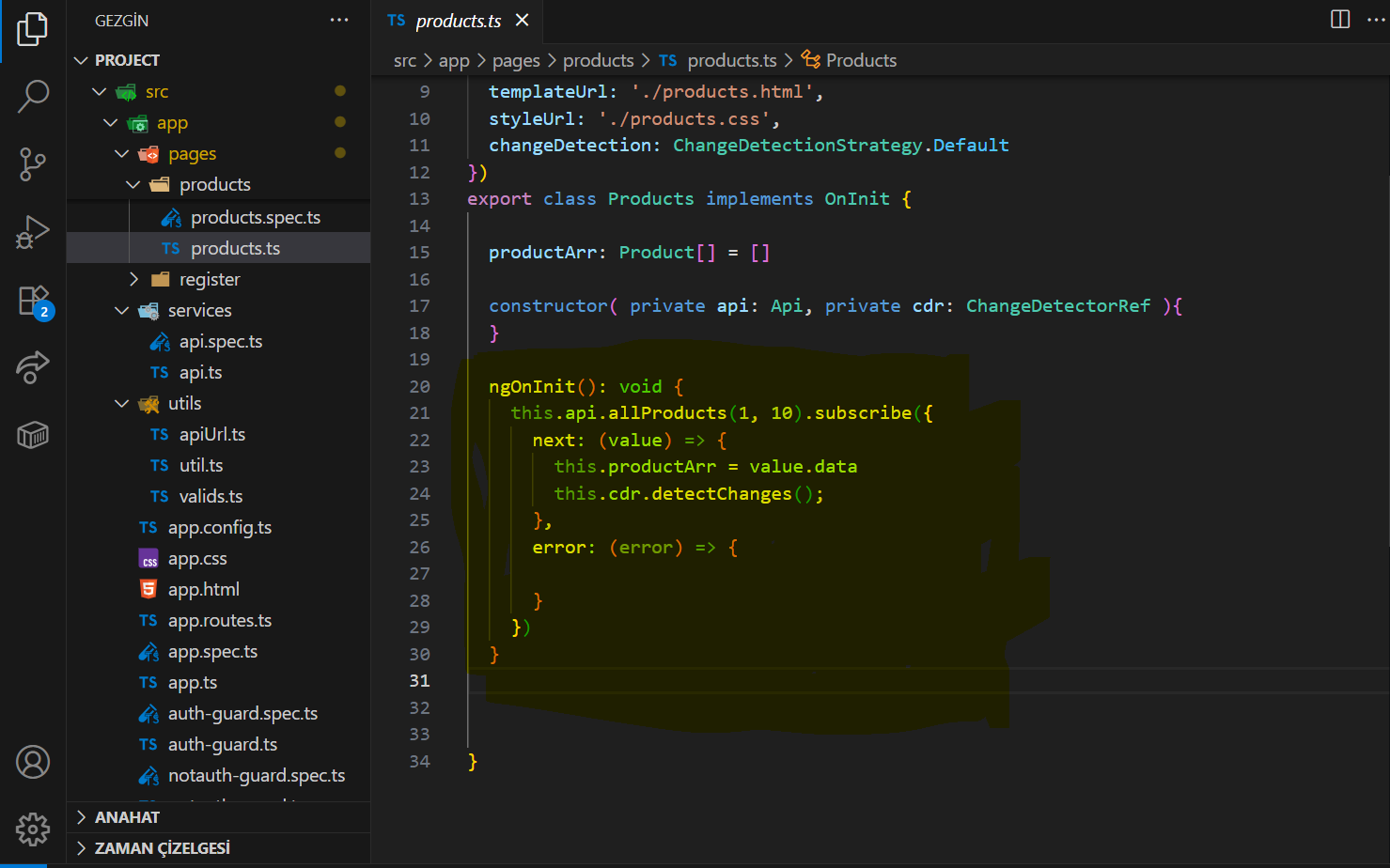
**});**

Burada:

* Authorization header’ı, token’ı sunucuya iletir.
* Sunucu da “Tamam, bu kullanıcı yetkili” diyerek cevabı döner.

Özetle:  
🔹 **localStorage:** Token’ı saklar  
🔹 **Header:** Token’ı her istekte sunucuya taşır

### Days\_56: Angularda guard yöntemi - products sayfası geliştirmeye devam \*\*FrontEnd/Project---> 26/07/2025



**ngOnInit(): void**

* Angular’da bir component yüklendiğinde ilk çalışan metottur.
* Angular OnInit arayüzünü (interface) kullanarak bileşen yüklendiğinde yapılacak işleri burada tanımlarsın.

**this.api.allProducts(1, 10)**

* Api servisinden allProducts metodunu çağırıyor.
* Bu metod büyük ihtimalle sunucudan ürün verilerini getiren bir HTTP GET isteği yapıyor.
* Parametreler:
  + 1: İlk sayfa (pagination için)
  + 10: Sayfa başına 10 ürün (limit)

Bu method muhtemelen şu şekilde tanımlıdır:

**allProducts(page: number, per\_page: number): Observable<IProducts> { ... }**

**.subscribe({ ... })**

* Angular'da HttpClient ile yapılan istekler bir **Observable** döner.
* subscribe() metodu ile bu isteğe abone olunur ve sonuçlar alınır.

**✅ next: (value) => { ... }**

* HTTP isteği **başarılı** olursa çalışır.
* value.data: Gelen verinin içindeki ürün listesi (value API'den gelen response objesi).
* this.productArr = value.data: Bu liste, bileşendeki productArr dizisine atanıyor.

**this.cdr.detectChanges();**

* ChangeDetectorRef servisi kullanılarak Angular’a **“Veri değişti, DOM’u yeniden kontrol et”** komutu veriliyor.
* Bu, bazı durumlarda Angular’ın otomatik fark edemediği değişiklikleri DOM’a yansıtmak için kullanılır (özellikle OnPush change detection stratejisi kullanıldığında).

**❌ error: (error) => { ... }**

* Eğer HTTP isteği **başarısız** olursa (örneğin ağ hatası, 404, 500 vb.) bu blok çalışır.
* Şu an boş bırakılmış, ama genellikle kullanıcıya uyarı vermek, loglama yapmak veya alternatif işlem yürütmek için kullanılır.

**🎯 Özet:**

Bu kod, bileşen yüklendiğinde:

1. İlk 10 ürünü API’den çeker,
2. Gelen ürünleri productArr dizisine atar,
3. DOM’un güncellenmesini manuel olarak tetikler (detectChanges()),
4. Hata olursa error bloğunda işlem yapılabilir.

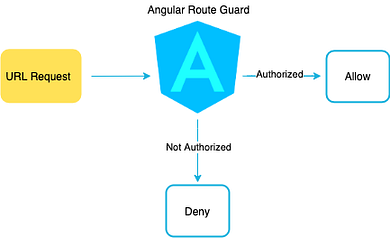
**🔐 Angular Guard Nedir?**

Guard'lar, **kullanıcının belirli bir sayfaya erişimini engellemek veya yönlendirmek** için kullanılır.

**Guard (Koruma)**, Angular'da bir kullanıcı bir sayfaya gitmeden önce bir kontrol yapılmasını sağlar. Örneğin:

* Giriş yapmamış kullanıcıyı /dashboard sayfasına sokmamak,
* Admin olmayan kullanıcıyı /admin paneline yönlendirmemek,
* Sayfadan çıkmadan önce "form verileriniz silinecek" diye uyarmak...

Bu tarz işlemler **guard’larla** yapılır.



**🧩 Guard Türleri**

| **Guard Türü** | **Ne zaman çalışır?** |
| --- | --- |
| CanActivate | Bir route’a girilmeden önce kontrol yapar. |
| CanActivateChild | Alt route'lara girilmeden önce kontrol yapar. |
| CanDeactivate | Route değişmeden önce (sayfadan çıkarken) kontrol yapar. |
| CanLoad | Lazy-loaded modül yüklenmeden önce çalışır. |
| Resolve | Sayfa açılmadan veri getirir (önceden fetch). |

**🛠️ Örnek: CanActivate Guard ile Giriş Kontrolü**

**Guard Oluşturma**

Terminalde:

**ng generate guard auth**

Veya kısa:

**ng g guard auth 🡪**Bu komut sana auth.guard.ts adında bir dosya oluşturur.

Angular’da guard sistemi, uygulamanın güvenliğini ve yönlendirme mantığını kontrol etmek için çok güçlü bir araçtır. Aşağıda detaylı şekilde Angular'da Guard (özellikle CanActivate) kullanım örneği:

**🧱 Guard Türleri**

| **Guard Adı** | **Ne Zaman Kullanılır?** |
| --- | --- |
| CanActivate | Bir route’a girilmeden önce kontrol |
| CanActivateChild | Bir route’un çocuk route’ları (alt sayfaları) için kontrol |
| CanDeactivate | Sayfadan çıkılmadan önce uyarı göstermek |
| CanLoad | Lazy-loaded modüllerin yüklenmesini engellemek |
| Resolve | Sayfa açılmadan önce veri getirmek |

Biz şu an **CanActivate**'i detaylı öğreneceğiz.

**🎯 Senaryo: Giriş Yapmamış Kullanıcıyı /home sayfasına sokma!**

**📁 Proje Dosya Yapısı (örnek)**

src/

├── app/

│ ├── pages/

│ │ ├── home/

│ │ └── login/

│ ├── services/

│ │ └── auth.service.ts

│ ├── guards/

│ │ └── auth.guard.ts

│ ├── app-routing.module.ts

│ └── app.module.ts

**1️⃣ AuthService Oluştur (Kullanıcı giriş yapmış mı kontrolü)**

**// services/auth.service.ts**

**import { Injectable } from '@angular/core';**

**@Injectable({**

**providedIn: 'root'**

**})**

**export class AuthService {**

**isAuthenticated(): boolean {**

**return !!localStorage.getItem('token'); // token varsa giriş yapılmış**

**}**

**}**

**2️⃣ Guard Oluştur**

Terminalden şu komutu gir:

**ng g guard guards/auth**

Bu komut sana auth.guard.ts dosyasını oluşturur.

**// guards/auth.guard.ts**

**import { Injectable } from '@angular/core';**

**import { CanActivate, Router } from '@angular/router';**

**import { AuthService } from '../services/auth.service';**

**@Injectable({**

**providedIn: 'root'**

**})**

**export class AuthGuard implements CanActivate {**

**constructor(**

**private authService: AuthService,**

**private router: Router**

**) {}**

**canActivate(): boolean {**

**if (this.authService.isAuthenticated()) {**

**return true; // Kullanıcı giriş yaptıysa erişime izin ver**

**} else {**

**this.router.navigate(['/login']); // Giriş yapmadıysa login sayfasına yönlendir**

**return false;**

**}**

**}**

**}**

**3️⃣ Guard’ı Route’a Ekle**

**// app-routing.module.ts**

**import { NgModule } from '@angular/core';**

**import { RouterModule, Routes } from '@angular/router';**

**import { HomeComponent } from './pages/home/home.component';**

**import { LoginComponent } from './pages/login/login.component';**

**import { AuthGuard } from './guards/auth.guard';**

**const routes: Routes = [**

**{ path: 'home', component: HomeComponent, canActivate: [AuthGuard] },**

**{ path: 'login', component: LoginComponent },**

**{ path: '\*\*', redirectTo: 'login' }**

**];**

**@NgModule({**

**imports: [RouterModule.forRoot(routes)],**

**exports: [RouterModule]**

**})**

**export class AppRoutingModule {}**

**4️⃣ Login Sayfasında Token Kaydet (örnek)**

**// login.component.ts**

**loginUser() {**

**localStorage.setItem('token', 'sahte-jwt-token');**

**this.router.navigate(['/home']);**

**}**

**🔄 Logout Yani Çıkış**

**logoutUser() {**

**localStorage.removeItem('token');**

**this.router.navigate(['/login']);**

**}**

**✅ Uygulama Akışı**

| **Durum** | **Ne Olur?** |
| --- | --- |
| Kullanıcı /home'a gitmek ister | Eğer token varsa girer, yoksa login'e yönlendirilir |
| Kullanıcı giriş yaptıktan sonra | Token localStorage'a kaydedilir, artık tüm korumalı sayfalara erişebilir |
| Kullanıcı logout yaparsa | Token silinir, artık korumalı sayfalara gidemez |

**🧪 Test Etmek İçin**

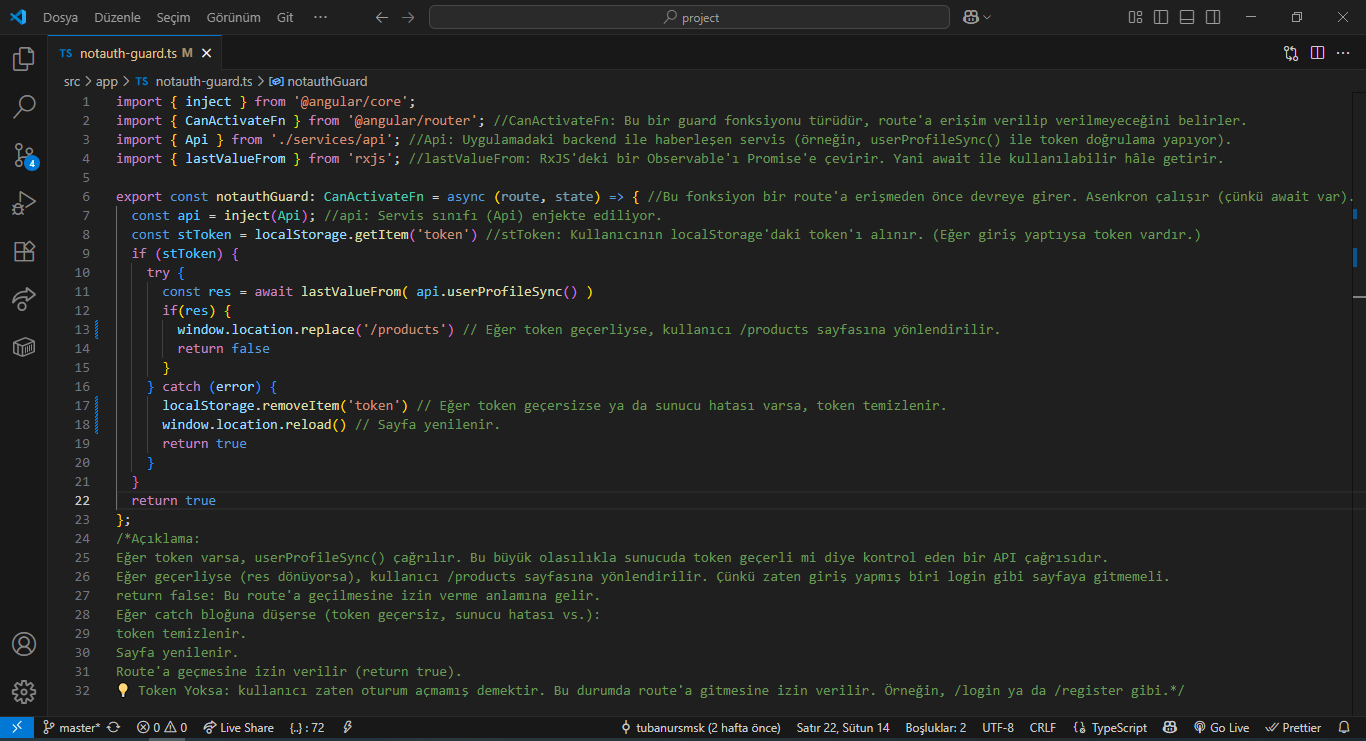
* localStorage.clear() komutuyla token'ı silip deneyebilirsin.
* Giriş yaptıktan sonra token'ı görmek için tarayıcı konsolundan localStorage.getItem('token') yazabilirsin.

**📌 Özetle**

* CanActivate → route'a girmeden önce kontrol.
* Guard içinde token ya da kullanıcı bilgisi kontrol edilir.
* Giriş yapılmadıysa yönlendirme yapılır.
* Angular'da güvenlik açısından önemli bir yapı taşını oluşturur.

### Days\_57: Buraya kadar gelinen Angular konularının tekrarı \*\*FrontEnd/Project---> 27/07/2025

\*\*\*notauth-guard dosyası



**Observable ve Promise’in Türkçe karşılığı**

Observable → “Gözlemlenebilir” veya “İzlenebilir akış”  
(zamanla birden fazla veri üreten mekanizma)

Promise → “Söz” veya “Gelecek”  
(ileride bir defa sonuç dönecek söz)

**🔄 lastValueFrom Nedir?**

lastValueFrom, RxJS (Reactive Extensions for JavaScript) kütüphanesinin bir fonksiyonudur. Bu fonksiyon, **bir Observable'ı Promise'e dönüştürür**.

**1️⃣ Observable’ı Promise’e dönüştürmek nedir?**

Angular’da (ve RxJS’de) **HTTP istekleri** ve benzeri asenkron işler Observable tipinde döner.  
Ama bazen Promise tipine ihtiyaç duyabilirsin (örneğin async/await ile kullanmak için).

👉 Observable → Promise dönüşümü şu anlama gelir:

* **Observable**, birçok değeri (stream/akış) zamanla üretebilir.
* **Promise**, sadece **bir kere** değer üretir ve biter.

Dolayısıyla Observable’ı Promise’e dönüştürdüğünde:

* Observable’ın sadece **ilk değeri** alınır.
* async/await ile rahat kullanılabilir.

**✅ Neden lastValueFrom’a Gerek Duyarız?**

Observable, Angular'da özellikle HttpClient gibi servislerden veri alırken kullanılır. Ancak Observable, Promise gibi await ile doğrudan kullanılmaz.  
Dolayısıyla, eğer async/await yapısını kullanmak istiyorsan, Observable'ı önce bir Promise'e çevirmelisin. İşte burada devreye lastValueFrom girer.

**🧠 Neden "last" (son) değeri?**

Çünkü Observable’lar çoklu değer yayınlayabilir. Ama Promise’ler sadece **tek bir değer döndürür**.  
lastValueFrom, yayınlanan son değeri alır. Eğer Observable hemen tamamlanıyorsa, zaten sadece bir değer yayınlar.

**🆚 firstValueFrom ile Farkı**

* firstValueFrom: İlk değeri alır.
* lastValueFrom: Son değeri alır (özellikle birden fazla değer yayınlanması mümkünse önemlidir).

**🔍 Ne Zaman Kullanılır?**

* Servisten gelen cevabı await ile almak istediğinde
* async fonksiyon içinde Observable ile uğraşmak istemediğinde
* Daha okunaklı bir try/catch yapısı kurmak istediğinde

🡪Notauth-guard.ts dosyasına bakalım…

**Observable – Promise Karşılaştırma Tablosu**

| **Özellik** | **Observable (Gözlemlenebilir)** | **Promise (Söz)** |
| --- | --- | --- |
| Veri sayısı | Birden fazla değer üretebilir (stream) | Sadece **bir değer** üretir |
| İptal | İptal edilebilir (unsubscribe()) | İptal edilemez |
| Lazy/Eager | Lazy (subscribe edilmeden çalışmaz) | Eager (oluşturulunca hemen çalışır) |
| Kullanım | subscribe() ile | then() / catch() veya async/await |

Days\_58: JWT - Token denetimi - LogOut işlemleri \*\*FrontEnd/Project---> 02/08/2025

**🧠 Angular'da Kullanıcı Oturumu (Session) Yönetimi Nedir?**

Bir kullanıcı giriş yaptığında onun kimliğini geçici olarak hatırlamak istersin. Bu, genelde:

* 🔐 **JWT (JSON Web Token)** ile
* 📦 **LocalStorage** / **SessionStorage** kullanılarak
* 🌐 Backend ile haberleşerek yapılır

**💡 Terimlerin Anlamı**

| **Terim** | **Açıklama** |
| --- | --- |
| **Session** | Sunucu taraflı kullanıcı oturumu (örn. ASP.NET, PHP'de olur) |
| **Token** | Giriş sonrası backend’in verdiği şifreli kullanıcı kimlik doğrulama verisi |
| **LocalStorage** | Kalıcı, kullanıcı silene kadar tarayıcıda kalan veri alanı |
| **SessionStorage** | Sekme kapanınca silinen geçici depolama alanı |

**🔐 Tipik Oturum Akışı (JWT + Angular)**

**1. 🧍 Kullanıcı Giriş Yapar**

**this.api.login(email, password).subscribe((res: any) => {**

**localStorage.setItem('token', res.token);**

**this.router.navigate(['/dashboard']);**

**});**

**2. ✅ Token LocalStorage’a Kaydedilir**

* Tarayıcıda şunu görebilirsin:

**localStorage.getItem('token');**

**3. 🔍 Oturum Kontrolü (Guard ile)**

**export const authGuard: CanActivateFn = (route, state) => {**

**const stToken = localStorage.getItem('token');**

**if (!stToken) {**

**window.location.replace('/login');**

**return false;**

**}**

**return true;**

**};**

**4. 📤 Backend’e Her İstekle Token Gönderilir**

**headers: {**

**Authorization: `Bearer ${localStorage.getItem('token')}`**

**}**

**5. 🚪 Kullanıcı Çıkış Yapar**

**localStorage.removeItem('token');**

**this.router.navigate(['/login']);**

**📦 LocalStorage vs SessionStorage**

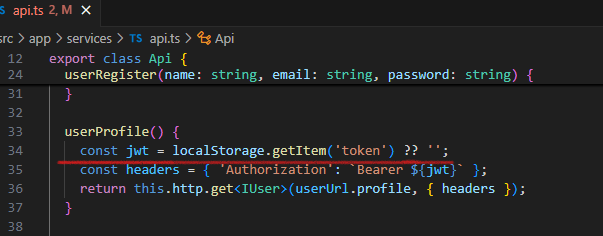
| **Özellik** | **LocalStorage** | **SessionStorage** |
| --- | --- | --- |
| Süreklilik | Kalıcı (kapatınca bile durur) | Sekme kapatılınca silinir |
| Depolama limiti | ~5-10 MB | ~5 MB |
| Güvenlik | XSS’ye karşı savunmasızdır | Aynı |

Angular'da genellikle localStorage kullanılır çünkü kullanıcı sayfayı kapatıp açsa bile oturumu korumak istenir.

**🎯 Özet**

Angular tarafında “session” kelimesi:

* Genellikle token bazlı oturum anlamına gelir
* localStorage ya da sessionStorage ile token saklanır
* Guard, interceptor ve API çağrılarıyla kontrol edilir.



 Altı çizili kodun yazım amacı 🡪 localStorage'a git token varsa al yoksa boş string al. Eğer değer null olsa hata verirdi. Şimdi token yoksa boş string almış olacak.



CanActivateFn sade ve fonksiyonel guard yazmanın modern yoludur (Angular 15+).

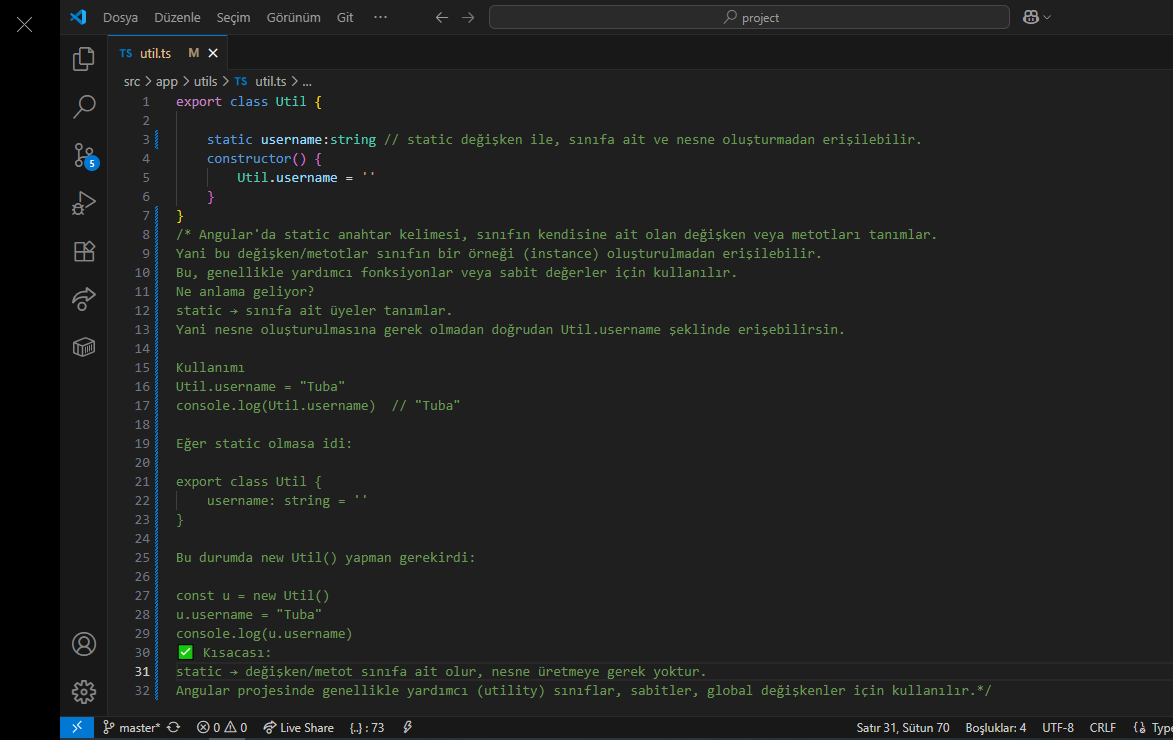
Token kontrolü yaparken mantığın doğru olması gerekir: token varsa true, yoksa false.

Yanlış: if (!token) return true → bu, yetkisiz kişiye izin vermek olur.

window.location.replace('/login') ile yönlendirme yapılabilir ama

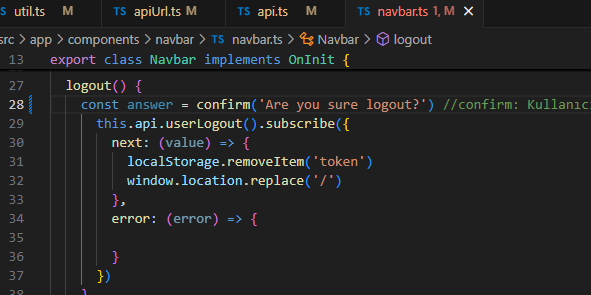
Angular’da Router.navigate() kullanmak daha doğrudur.

\*\*\*Angular altında Statik anahtarı ne işe yarar?



### Days\_59: LogOut işlemleri – product-detail.html sayfa tasarımı \*\*FrontEnd/Project---> 03/08/2025

\*\*LogOut işlemi, arka tarafta jwt’nin öldürülme işleminin servis tarafına bağlanmasıyla olur.



\*\*confirm: Kullanıcıya bir onay penceresi açar. Eğer kullanıcı "OK" butonuna basarsa true, "Cancel" butonuna basarsa false döner.

**\*\*localStorage.removeItem('token')** localStorage'dan token'ı siliyor.

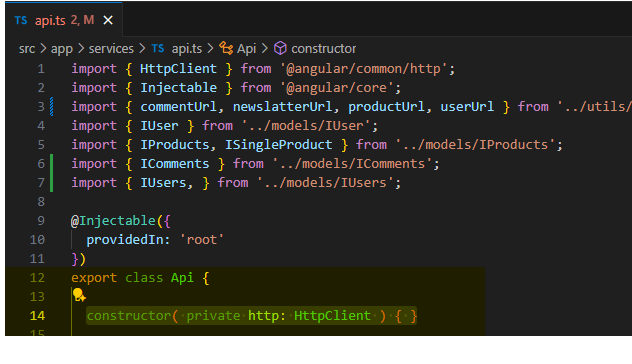
\*\***window.location.replace('/'**) Kullanıcıyı anasayfaya yönlendiriyor. Bu, mevcut sayfayı yeni bir URL ile değiştirir.

### Days\_60: Buraya kadar gelinen Angular konularının tekrarı \*\*FrontEnd/Project---> 09/08/2025

\*\*\*SORULAR\*\*\*

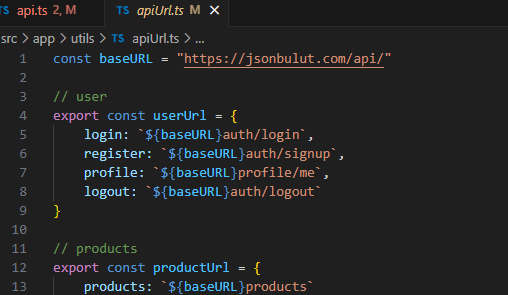
1-Api nedir?

2-Apiye giden ve onu kontrol eden yapı nedir?



3-userUrl bağlantısını neden login içerisine aktardık?

Tekrarlı yazımı önleyip merkezi şekilde yönetebilmek için. Böylelikle birden fazla yerde kullanılabilir. Hata riski de azalmış olur.



4-Modelleme yaparken mesela IUser.ts dosyasını nasıl oluşturuyoruz?

Quicktype uygulaması ile JSON formatında yazılan kodu TypeScript formatına dönüştürüp oluşan kodu ilgili model dosyasına aktarırız. (vscode üzerinden yardımcı excention ile de açabiliriz ctrl+shift+p🡺quicktype)

5- userLogin içinde sendObj nesnesi kurup parametreleri key - value şeklinde çağırmak ile return de parametreleri çağırmak aynı işlemi ve aynı işlem maliyetini verir mi?



Maliyet yoktur çünkü iki yazımda da süslü parantez ile obje oluşturuldu işlem tür olarak aynı lakin üstteki yazım daha derli toplu ve kısa yoldur.

6- Servislerde kaç türlü gönderim yöntemi vardır? 3 tane yöntem vardır.

1🡪 HTTP POST**:**

this.http.post(...) kullanıldığı için HTTP POST isteği yapılıyor. Bu yöntem JSON formatında çalışır. Aşağıdaki örnek kod yazımında path variable (ör. /login/ahmet/1234) veya query string (ör./login?email=ahmet&password=1234) kullanılmamış.

Bunun yerine JSON gönderim yolu seçilmiş:

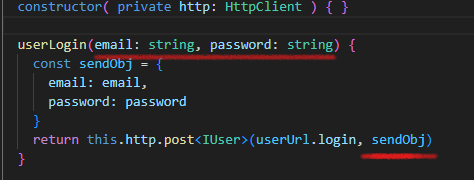
**Js** (apide yazan kısım)

**{**

**"email": "kullanici@mail.com",**

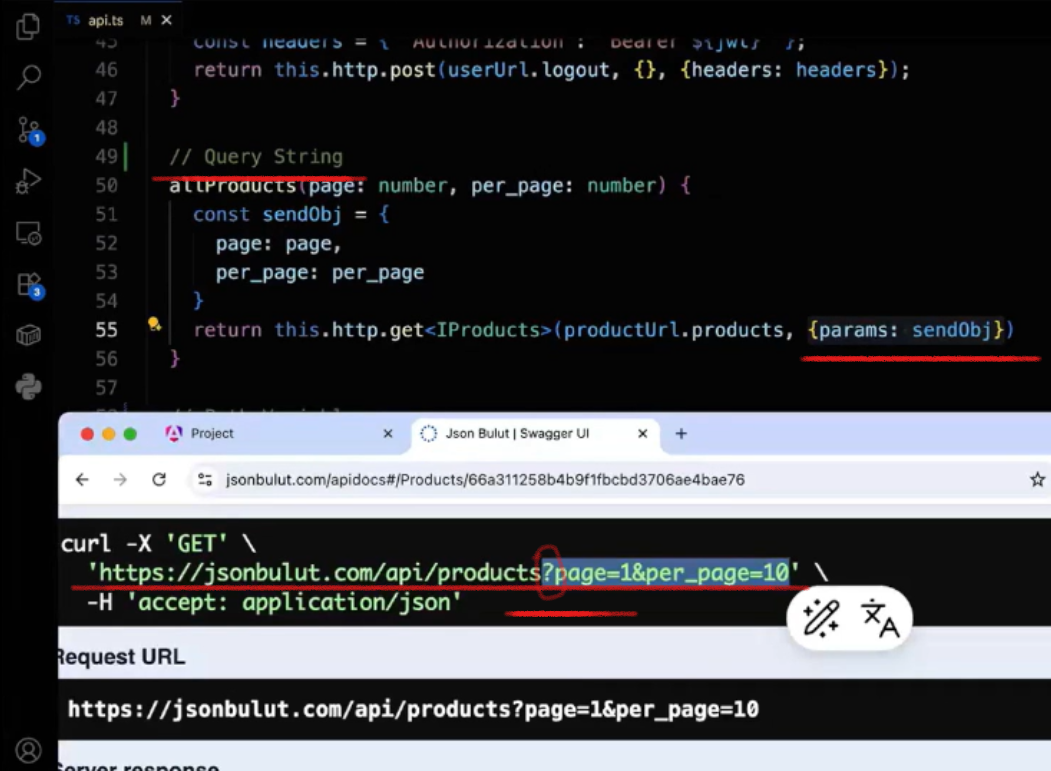
**"password": "123456"**

**}**

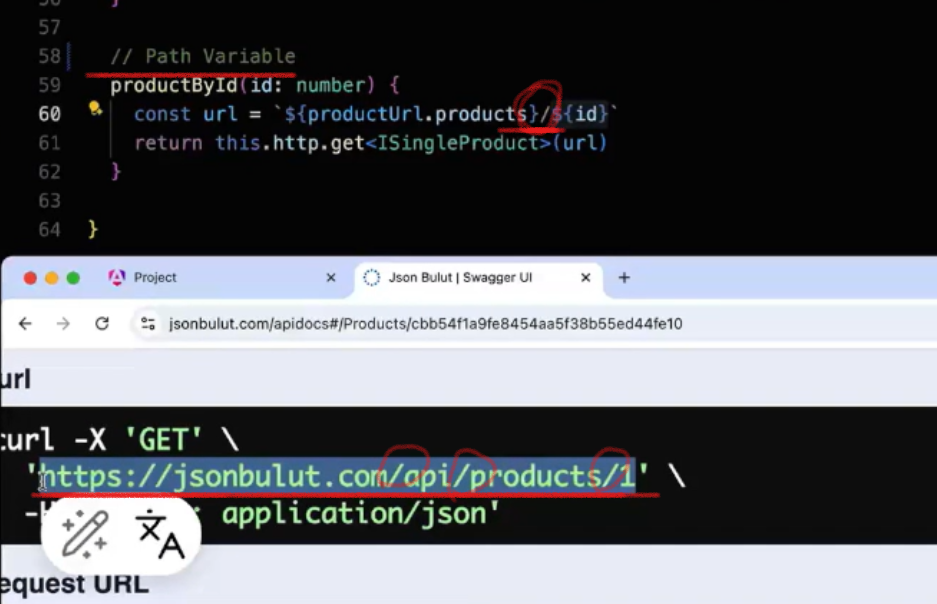


2🡪QUERY STRİNG: Gönderilen sorgunun ? işareti ile farklı parametrik davranışları vardır.

Burada sendObj’yi yukardaki gibi direk yamak yerine params içine attık sebebi ise query string parametre bekliyor .

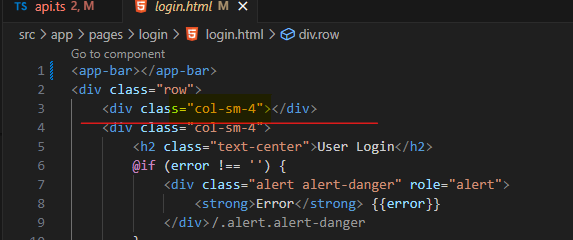


3🡪 PATH VARİABLE: Tekil data gönderiminde kullanılır.

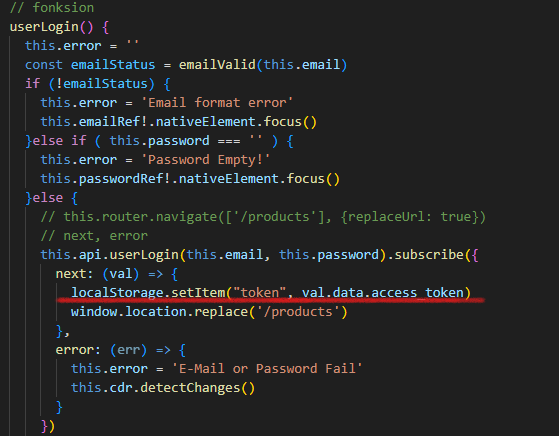


7- uygulamanın her ekran boyutuna uyum sağlamasını hangi kodlar ile yaparız?

GRİT yöntemi ile



8- localStorage içinde token neden saklandı, amaç nedir?

(login.ts dosyası)

Güvenli sayfaların doğrulanabilirliğini saklamak adına giriş yapan kişinin bilgisini burada saklanır. Bilgilerin saklanma amacı ise bir sonraki işlemlerde kullanıcının varlık denetiminin kolaylıkla sağlamak için. Her kullanıcı rolünün kapsamı dahilinde uygulamada işlem yapar bu bilgilerde localStrorage içinde token’lar ile tutulur.

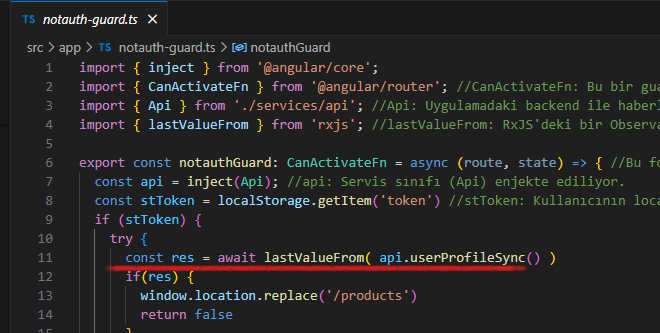
**\*\*window.location.replace('/products') 🡪** burada replace’in amacıda kullanıcı giriş işlmeni yaptı ve sayfalar arası geçiş yaparken geri gelmeyi (yani login sayfasına) engelliyor çünkü sistemde şu an aktif. Aktifken tekrar login işlemi yapmak saçma olurdu. Ama url üzerinden login’ gidebiliriz. Bunu engellemek içinde guard (notGuard: oturum açık mı değil mi?) yöntemi tanımlanır.

(Bkz. authGuard: oturumun varlığını denetler bu kısmın detaylı anlatımı için Days\_56 notlarına bak.)

9- guard metodu nasıl kurulur?

terminalde: ng g guardAdi

10- Bu kısımda a-senkron await neden kullanıldı?



Bir a-senkron işlem gerçekleştiğinde kendinden sonraki kod parçası bu fonksiyonun (lastValueFrom() çalışmasını beklemez. Bunun sonucunda gelecek olan cevap doğrulanmadan diğer fonksiyonun içindeki değerler(if(stToken) olan kısım) işlenmeye başlar. Await ile fonksiyona önce servise git userProfile’dan emin olduktan sonra aşağıdaki kodu çalıştırır.

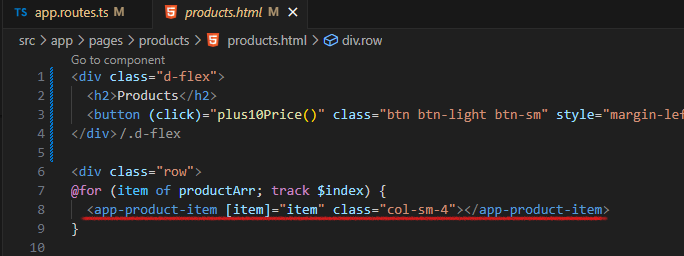
Burada await a- senkron sitemi bekletip kodu doğrulayan lastValueFrom ile cevabın ne zaman geldiğinden emin oluruz.

11-ngOnInit() ne zaman çalışır?

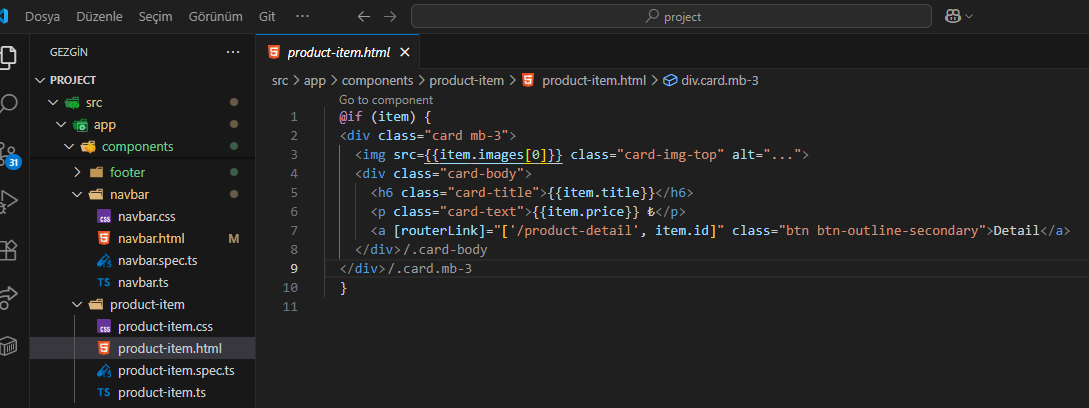
Angular’da her componentin bir **yaşam döngüsü (lifecycle)** vardır.  
ngOnInit() bu yaşam döngüsünün en çok kullanılan araçlarından biridir.

* constructor() → component oluşturulduğunda (dependency injection vs.) çalışır.
* ngOnInit() → DOM hazır olduğunda yani component tam olarak yüklenip hazır olduğunda çalışır.,

12- Burada servisten gelen datanın product-item gibi bir componentten etkilenmesini neden istedik, bu kodun işlevi nedir?

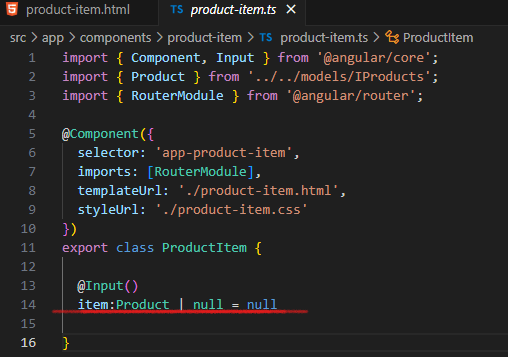


Bir yerde yazıp Farklı yerlerde kullanabilmek için navbar benzeri gibi. Tasarım değiştiğinde bu tasarımı kullanan her yer otomatik değişmiş olacak. İtem’ bir çok yerde çağıracağımız için işlem kolaylığı olması açısından bu yazım şeklini tercih ederiz. Ayrıca componet klasöründe prodouct-item adında item için klasör oluştururuz.



🡪bir üst fotoğraftaki kod bloğunda yazan @for(…) sayesinde de verileri import ettik. İnput anahatar kelimesi ile de veriler sağlıklı bir şekilde alınmış oldu.

İnput ile şunu diyoruz: sana bir item gelir ve **ilk değeri** producttır veya null olabilir. İlk değerin boş olma sebebi ise başlangıç item değeri olmadığı durumlarda olabilme ihtimalidir. Bunu tanımlayarak sistemin patlamasının önüne geçmiş oluruz.



### Days\_61: Tekrar – product-detail-comments işlemleri \*\*FrontEnd/Project---> 10/08/2025

\*\*\* Tekrar yapılarak product-detail üzerine comments eklendi..

### Days\_62: Projeye footer ekleme -db.json üretimi \*\*FrontEnd/Project---> 16/08/2025

json-server, Angular öğrenirken çok sık kullanılan mini bir backend simülatörüdür

**🔹 json-server Nedir?**

* Basit bir **Node.js paketi**dir.
* Elindeki db.json dosyasını alır, onu **REST API** gibi çalıştırır.
* Böylece sen sanki gerçek bir sunucudan veri alıyormuş gibi Angular ile HTTP istekleri yapabilirsin.

**🔹 Nasıl Çalışır?**

1. Projende db.json adında bir dosya oluşturursun (senin newsletter örneğin gibi).

{

**"newsletter": [**

**{ "id": "1", "email": "ali@mail.com" },**

**{ "id": "2", "email": "ayse@mail.com" }**

**]**

**}**

1. Terminalden paketi kurarsın:

**npm install -g json-server**

1. Varsayılan olarak http://localhost:\*\*\*\*? adresinde çalışır.
2. Artık aşağıdaki endpoint’ler hazır olur:
   * GET http://localhost:3000/newsletter → tüm kayıtları getirir
   * GET http://localhost:3000/newsletter/1 → id=1 olan kaydı getirir
   * POST http://localhost:3000/newsletter → yeni kayıt ekler
   * PUT http://localhost:3000/newsletter/1 → id=1 olan kaydı günceller
   * DELETE http://localhost:3000/newsletter/1 → id=1 olan kaydı siler

👉 Görüyorsun, sıfır backend kodu yazmadan sana tam bir REST API sunuyor.

**🔹 Senin Kodunda Kullanım**

Sen Api servisini yazıp this.api.newsletterControl() çağırdığında aslında şunu yapıyorsun:

this.http.get("http://localhost:3000/newsletter?email=" + this.email)

* Bu istek json-server’a gider.
* json-server db.json dosyasında newsletter dizisini kontrol eder ve eşleşenleri döner.

Yeni kayıt eklediğinde (POST) → db.json dosyasına otomatik yazılır.

**🔹 Avantajları**

✅ Hızlıca backend taklidi yapabilirsin.  
✅ Öğrenme/test aşamasında backend ekibine ihtiyaç duymazsın.  
✅ CRUD işlemlerini birebir görürsün.

**🔹 Sınırları**

❌ Gerçek SQL sorguları yok (JOIN, GROUP BY, TRANSACTION vs.).  
❌ Kullanıcı yetkilendirme, kimlik doğrulama gibi gelişmiş özellikler yok.  
❌ Büyük projeler için uygun değil, sadece **mock / prototip** amaçlı.

Yani kısacası:  
👉 json-server = **çalışma masasında küçük bir test mutfağı**  
👉 Gerçek Backend = **restoranın profesyonel mutfağı**

\*\*\*Angular projesinde veri tabanı olarak db.json paketi eklendi. Burada database işlemleri diğer dillerde olduğu gibi daha geniş kapsamlı değil Angular’da **db.json kullanımı** ile, “gerçek” veritabanı (SQL, MongoDB gibi) arasındaki farkları ve mantığını:

**1. Angular’ın Rolü**

* Angular bir **frontend (ön yüz) framework’üdür**.
* Görevi: kullanıcıya arayüz sunmak, butonları, formları, sayfaları yönetmek.
* Angular **doğrudan veritabanına bağlanmaz** ❌ (yani MySQL, MongoDB, MSSQL gibi sistemlerle direkt konuşamaz).
* Bunun yerine **API (backend)** üzerinden veri alır veya gönderir.

👉 Yani Angular = istemci (client).  
👉 Database = sunucu tarafında olur.  
👉 Angular → API → Database zinciri işler.

**2. db.json ile Ne Yapıyorsun?**

Senin db.json dosyan:

**{**

**"newsletter": [**

**{ "id": "2", "email": "ayse@mail.com" },**

**{ "id": "3", "email": "mehmet@mail.com" }**

**]**

**}**

* Bu aslında **gerçek bir veritabanı değil** → düz bir JSON dosyası.
* Bunun üzerine genelde json-server paketi çalıştırılır.
* json-server bu dosyayı alır ve **sanki bir REST API varmış gibi** çalıştırır.
* Yani sen Angular’da http.get("http://localhost:3000/newsletter") dediğinde, json-server sana bu JSON içeriğini döner.

👉 Bu yüzden Angular tarafında “gerçek database sorguları” (SQL yazmak, join yapmak, trigger çalıştırmak vb.) göremezsin. Çünkü sadece **fake API** üzerinden CRUD (Create-Read-Update-Delete) yapıyorsun.

**3. Senin Kodunda Olan Mantık**

Senin Footer bileşeninde:

* fncEmailSend() çağrıldığında önce email formatı kontrol ediliyor ✅
* Sonra this.api.newsletterControl(this.email) ile **json-server’daki newsletter dizisine** bakılıyor.
* Eğer e-posta yoksa → this.fncNewslatterAdd() ile API’ye POST isteği atılıyor.
* json-server bunu alıp db.json içine yeni kaydı yazıyor.

👉 Burada yaptığın işlem aslında bir “**fake backend**” simülasyonu.  
👉 Gerçek bir sistemde db.json yerine **.NET, Node.js, Java, PHP gibi bir backend** olur, onlar da MySQL/MongoDB ile çalışır.

**4. Özet Mantık**

* Angular tek başına database işlemi yapmaz ❌
* JSON-server gibi paketlerle **demo / mock veri tabanı** yapılabilir ✅
* Gerçek dünyada: Angular (Frontend) ↔ API (Backend) ↔ Database (SQL/Mongo)

**5. Avantaj – Dezavantaj**

✅ Avantajları:

* Öğrenme kolaylığı
* Hızlı prototip geliştirme
* Backend olmadan test yapabilme

❌ Dezavantajları:

* Karmaşık SQL sorguları çalıştıramazsın
* Kimlik doğrulama, transaction, ilişkisel tablolar vs. yok
* Yalnızca küçük testler/prototipler için uygundur

<https://mocki.io/fake-json-api> 👉 **Biz bu işlemleri neden yaptık, uygulamadaki amacı nedir?**

Bir uygulamada **API entegrasyonu** yapmamızın amacı:

* **Veriyi kullanıcıya getirmek** (ürün listesi, profil bilgisi, yorumlar vs.)
* **Kullanıcının yaptığı işlemleri kaydetmek** (login, register, sepet ekleme vs.)
* **Frontend ile Backend arasında köprü kurmak**

Yani aslında:

* **Frontend (Angular)**: Görsellik, kullanıcı ile etkileşim.
* **Backend (API)**: Veri saklama, iş kuralları, güvenlik.

Arada **HttpClient ile API çağrıları** yapıyoruz ki **kullanıcı gerçek verilerle çalışsın**.

**🔹 Neden Mock / Sahte API?**

Projeyi öğrenme sürecinde biz backend yazmadık. Ama uygulamamızın şu işlere ihtiyacı vardı:

* Ürün listesi göstermek
* Detay sayfası açmak
* Kullanıcı login/register yapmak

Backend olmayınca frontend boşta kalırdı. İşte bu yüzden:

1. **Mock API** veya **json-server** gibi sahte API kullandık.  
   → Böylece gerçekmiş gibi veri çekebildik.
2. **Service (Api sınıfı)** yazdık.  
   → Component’ler tek tek URL bilmek zorunda kalmadı, temiz bir katman oluştu.

**🔹 Uygulamadaki Hedefler**

* **Pratik yapmak** → Angular’da HttpClient, service, subscribe, observable kavramlarını öğrenmek.
* **Gerçek senaryoyu simüle etmek** → Sanki bir e-ticaret sitesinde çalışıyormuşuz gibi ürün listesi, login, register gibi akışları kurduk.
* **Katmanlı yapı kurmak** → Gerçek hayatta da frontend–backend ayrımı böyle yapılıyor, biz de küçük ölçekte uyguladık.
* **Test edilebilirlik** → Mock API sayesinde senaryoları rahat test edebildik.

**🔹 Gerçek Hayatta Amacı**

Büyük sitelerde bu mantığın karşılığı şudur:

* Frontend ekibi → Angular/React/Vue yazıyor.
* Backend ekibi → Java/.NET/Node/Python ile API yazıyor.
* Test süreci → Mock API veya staging ortamı ile başlıyor, gerçek API’ye geçişle bitiyor.

Bizim projede bu süreci **öğrenme amaçlı küçük ölçekte canlandırdık**.

✅ **Özet:**  
Bu işlemleri yaptık çünkü:

* Uygulamanın **gerçek verilerle çalışıyormuş gibi davranmasını** sağladık.
* Angular’da **API çağrısı yapmayı, veriyi ekranda göstermeyi** öğrendik.
* Gerçek projelere hazırlık yapmış olduk.

### Days\_63: Projeye users sayfası ekleme \*\*FrontEnd/Project---> 17/08/2025

<https://medium.com/@ilaydakosar/postman-nedir-ve-nas%C4%B1l-kullan%C4%B1l%C4%B1r-6d5096b2a284>