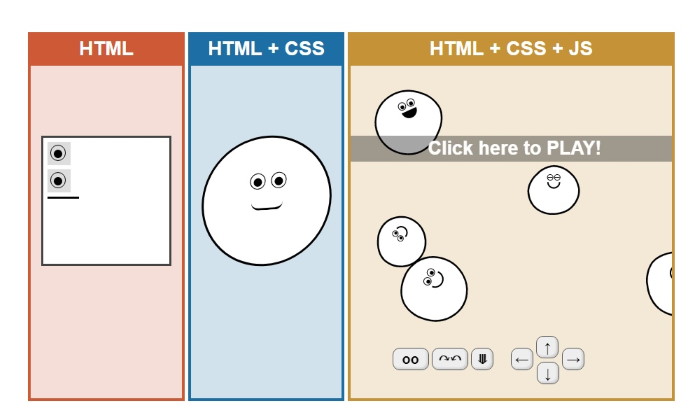
DERS NOTLARI VE TEKRAR TAKİBİ

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

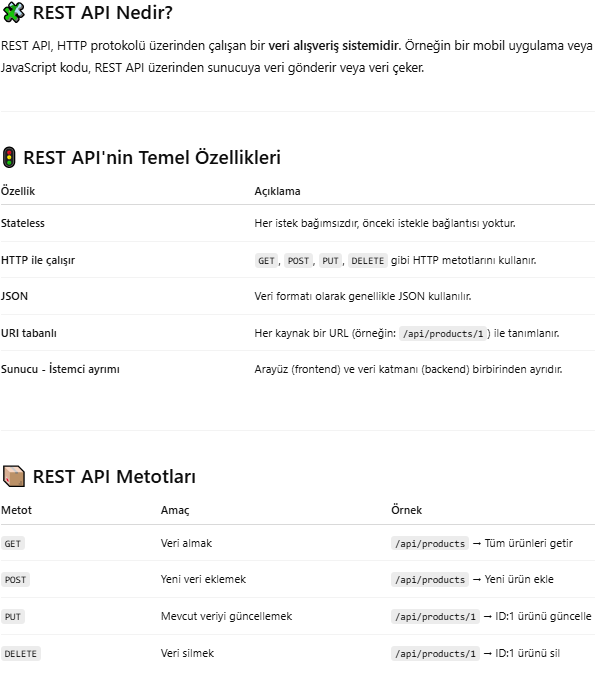
----------------------------------------------------------- \*\*\* ----------------------------------------------------------

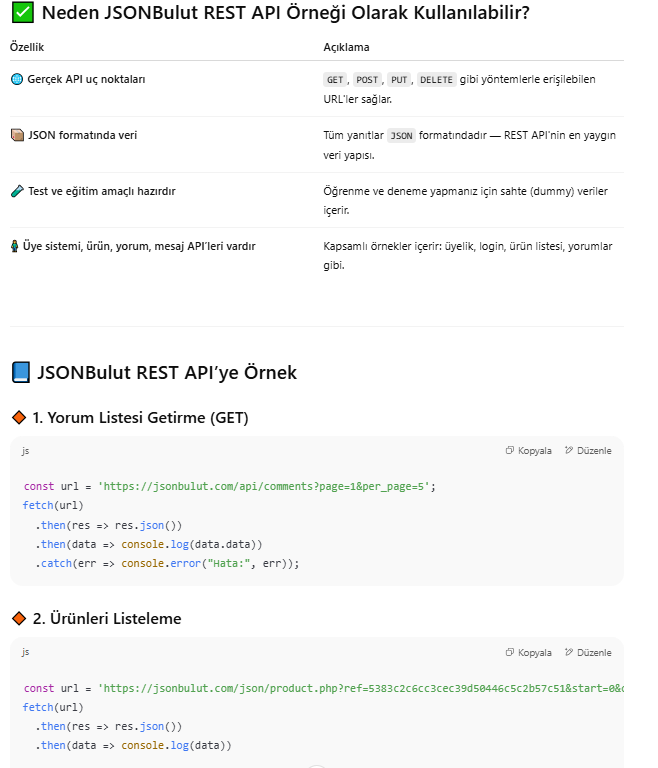
Bu kısımda ki notların kod dökümünün dosya yolu 🡪 belgeler/github/workspace/frontend/Days\_7

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

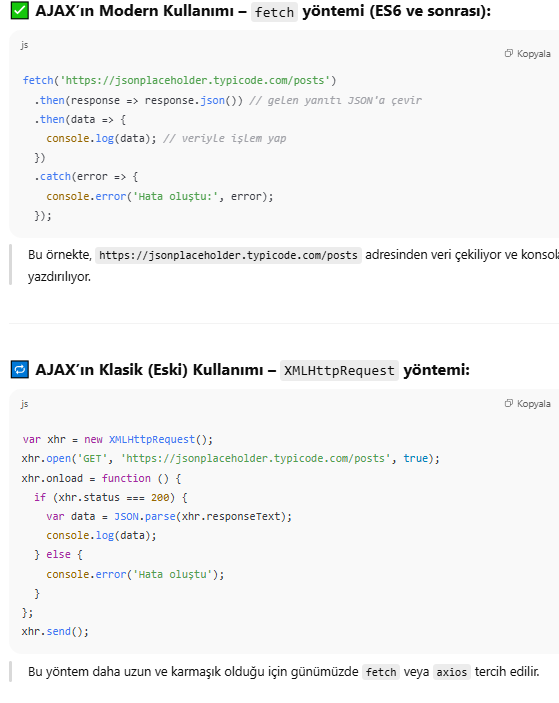
**🔐 Not:**

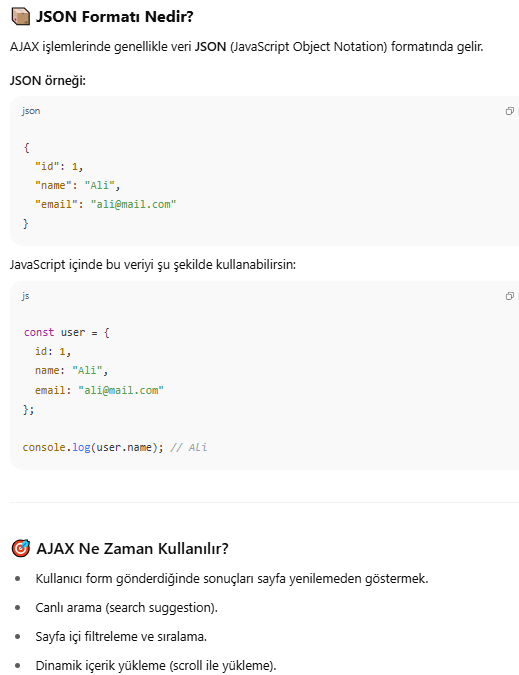
Bazı API'ler ref yani referans anahtarı (API key) gerektirir. JSONBulut da bu yöntemi kullanır. Ücretsiz üye olarak kendi ref anahtarını alıp testler yapabilirsin.

**🔧 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) |

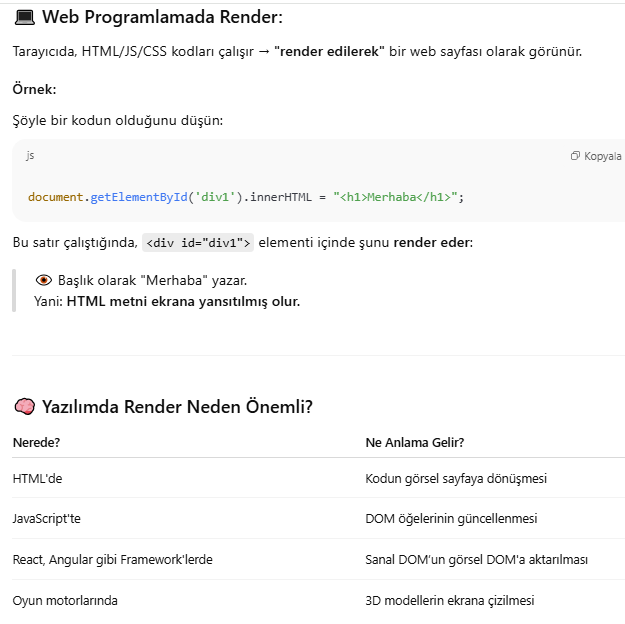
****

****

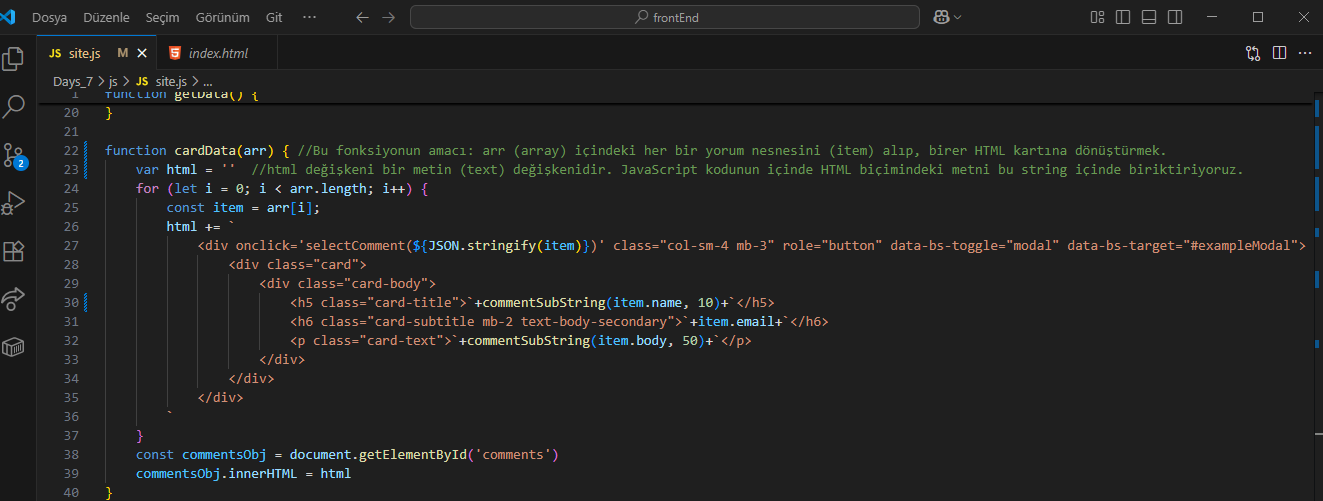
\*\*\* Render, kelime anlamı olarak:

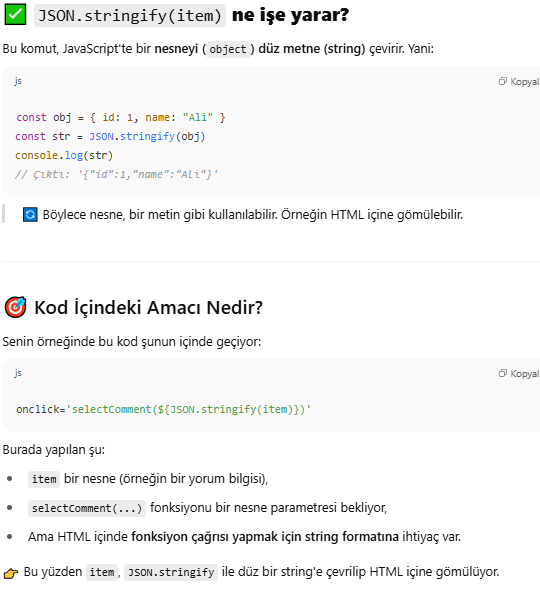
“Görselleştirmek” ya da “ekrana çizmek/göstermek” anlamına gelir.

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.

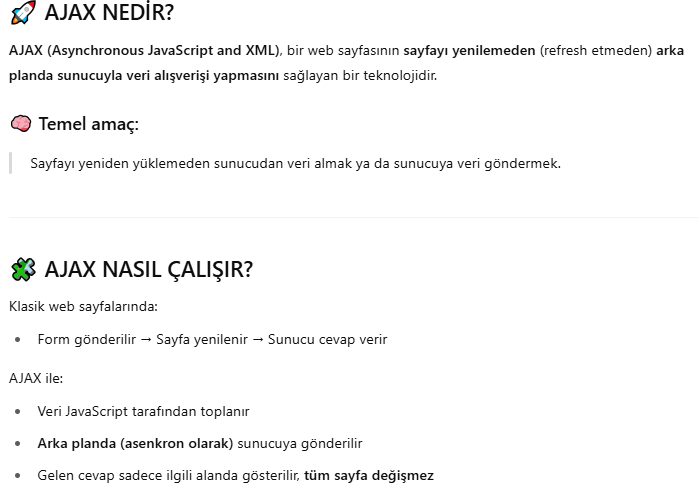
### 

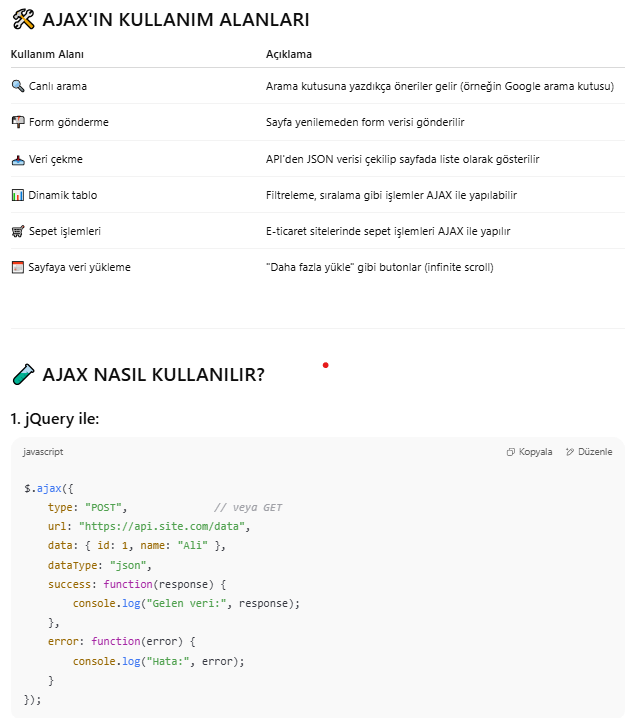
### 

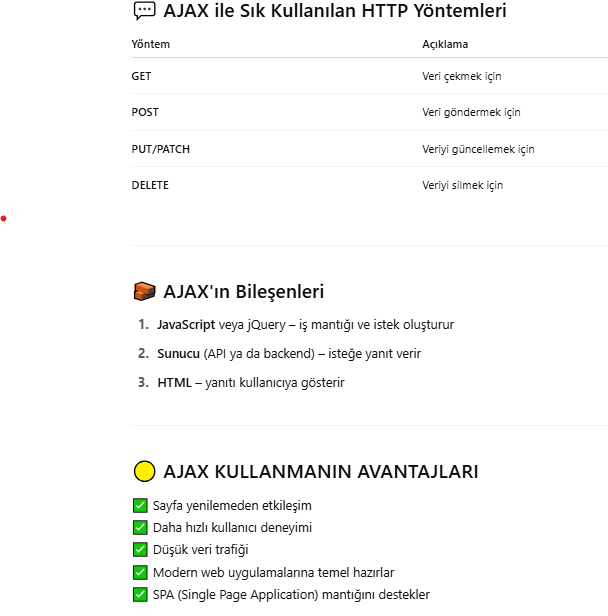
\*\*\*Bkz. bu dersin ekser notları FrontEnd/Days\_8 klasöründe tekrar et!

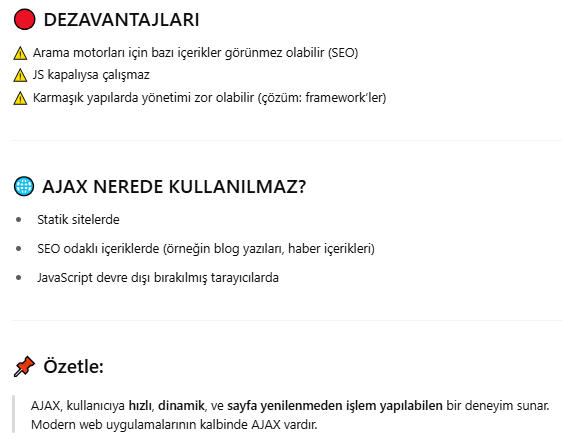
Days\_49: jquery - JavaScript - AJAX kullanımı \*\*FrontEnd/Days\_9---> 22/06/2025

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

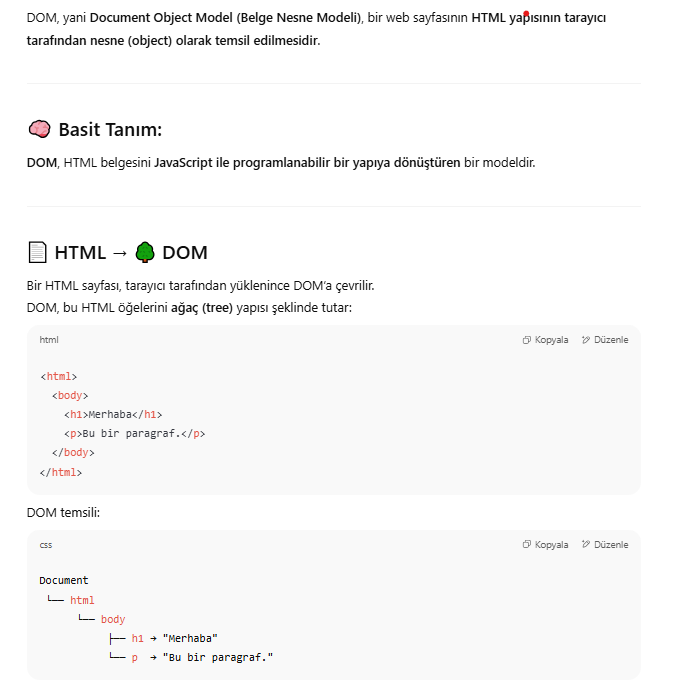


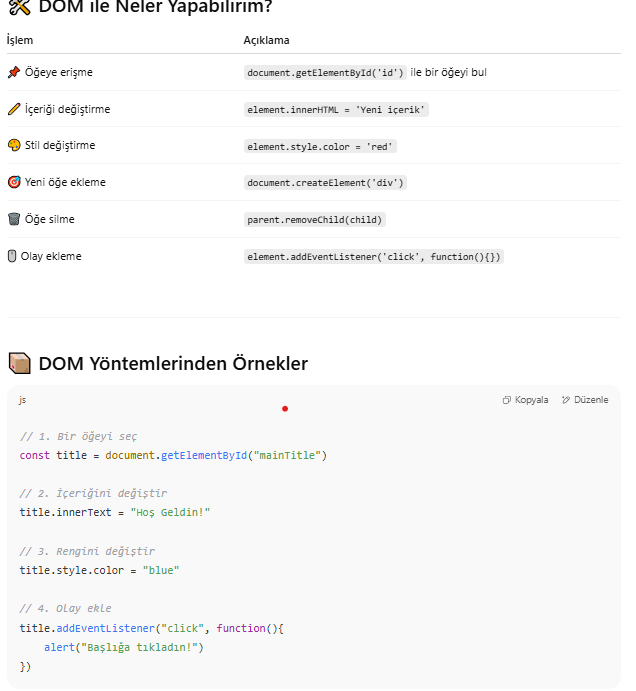


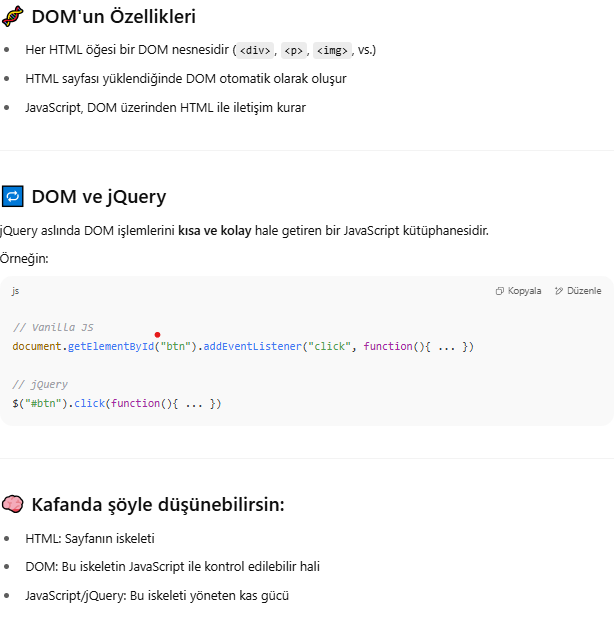




### DOM NEDİR? JavaScript ya da jQuery ile **sayfadaki öğelere erişmek**, **değiştirmek**, **silmek** ya da **yeni öğeler eklemek** için DOM kullanılır.







**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 |
|  |  |

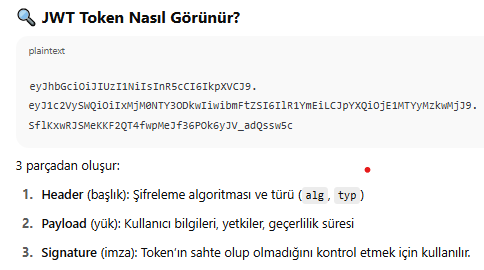
### 🔐 JWT Token Nedir? Bkz.-->Days\_9/login.js/39.kod satırı

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>

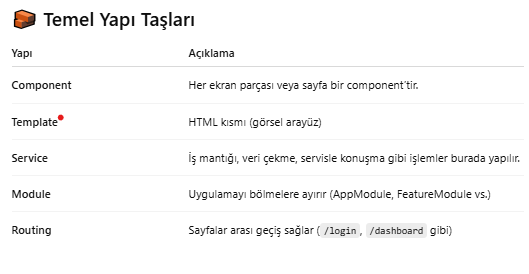
### **🔧** ANGULAR NEDİR?

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.

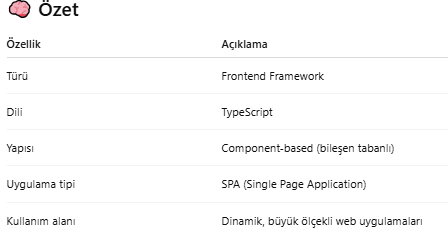
****

**⚙️ 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?**

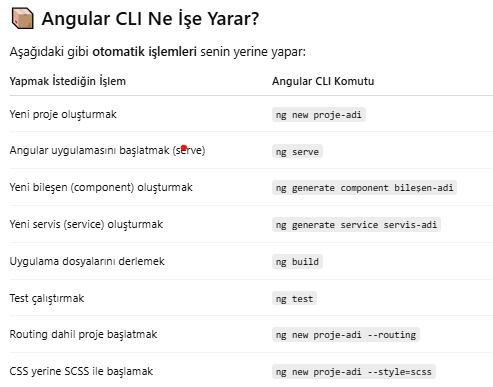
* Node.js ve npm
* Angular CLI (npm install -g @angular/cli)
* Kod editörü (Visual Studio Code gibi)
* Tarayıcı (Chrome, Edge vs.)



**🧭 Angular CLI Nedir?**

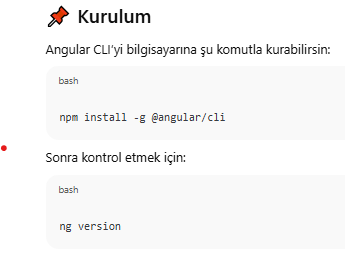
Angular CLI, "**Command Line Interface**" yani **Komut Satırı Arayüzü** anlamına gelir ve Angular projelerini kolayca oluşturmak, yönetmek ve geliştirmek için kullanılan bir araçtır.

**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

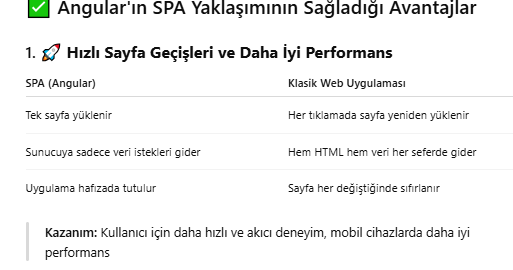
**🎯 Özet**

**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**

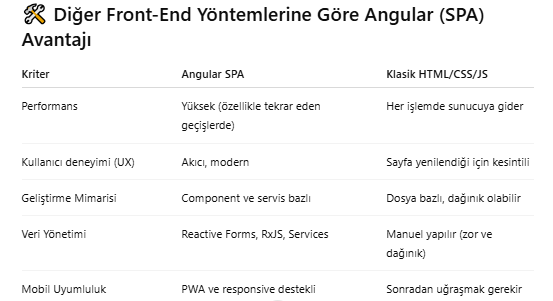
* Angular RouterModule sayesinde /home, /about gibi rotaları sayfa yenilemeden yönetir.
* Kullanıcı bir bağlantıya tıkladığında yalnızca içerik değişir, sayfa yeniden yüklenmez.

**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

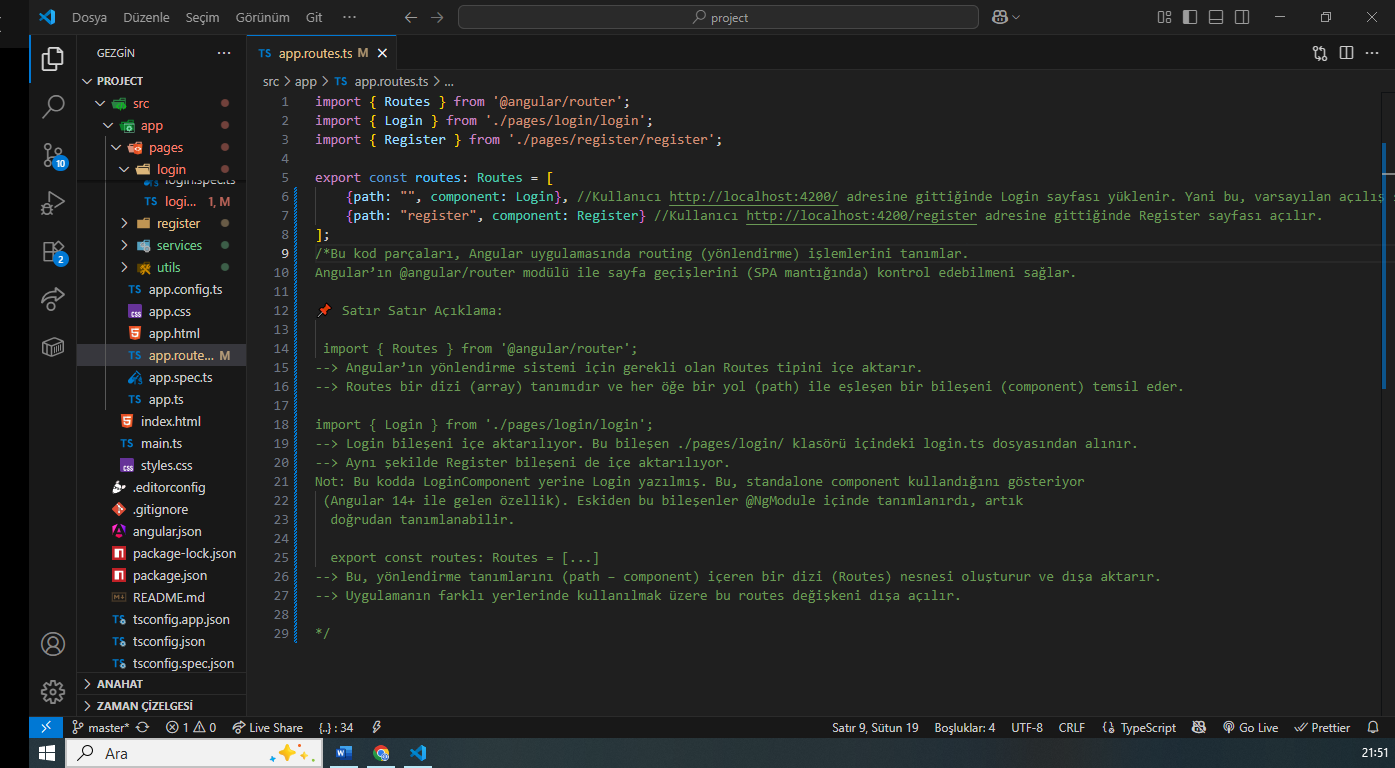


**📌 Özetle:**

**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)

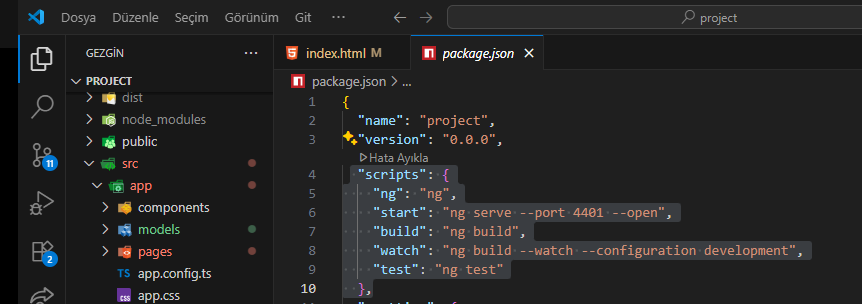
Dersten kod örnekleri:



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

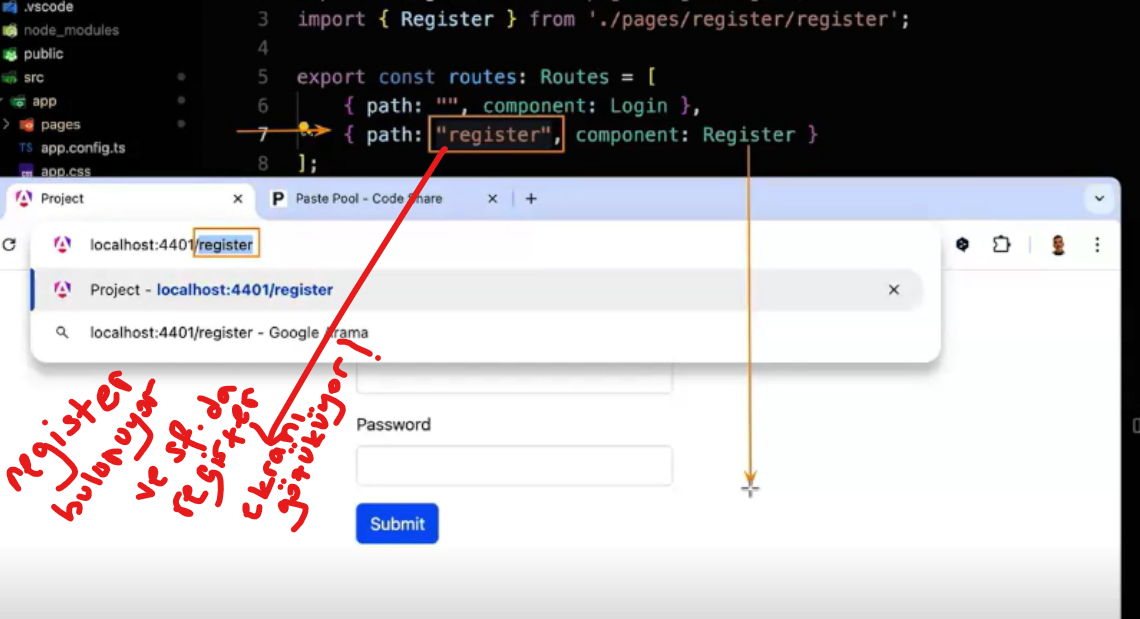
🡪Login veya register sayfası nasıl açılır?

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.

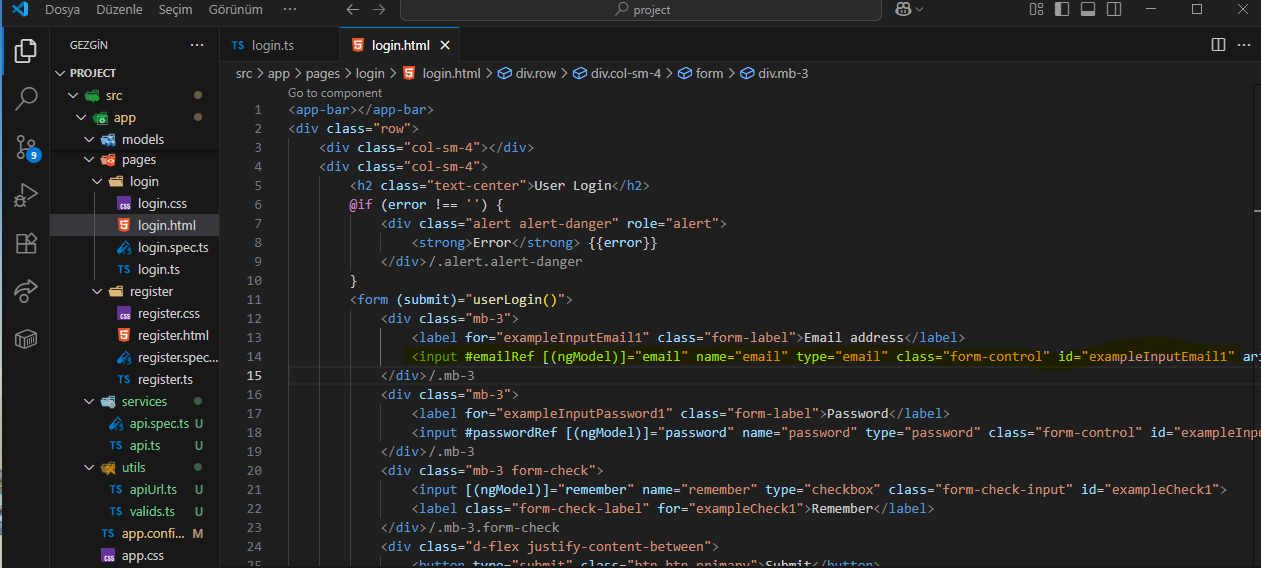
🡪 path nedir? url’de ki kısmı ifade eder.



🡪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**

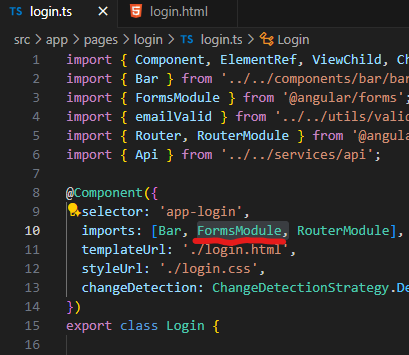
* Bu, Angular’daki **template reference variable**’dır.
* HTML tarafında bu input'a **referans vermek** için kullanılır.
* Bu sayede TypeScript dosyasına gerek kalmadan bu input'u HTML'de tanımlı başka yerlerde doğrudan kullanabilirsin.

**🔹 [(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')**

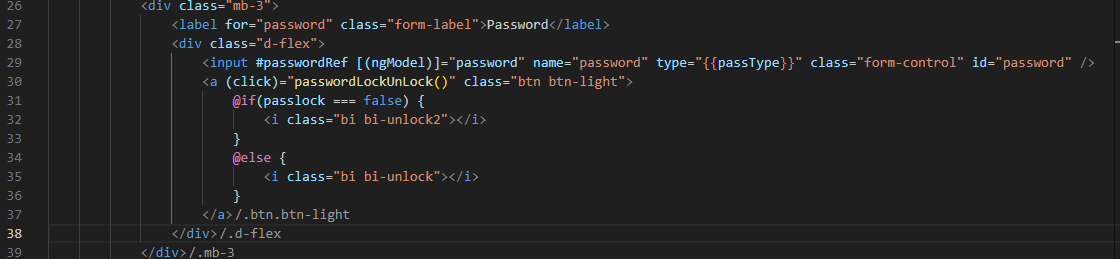
Bu ne demek?

* 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>**

**}**

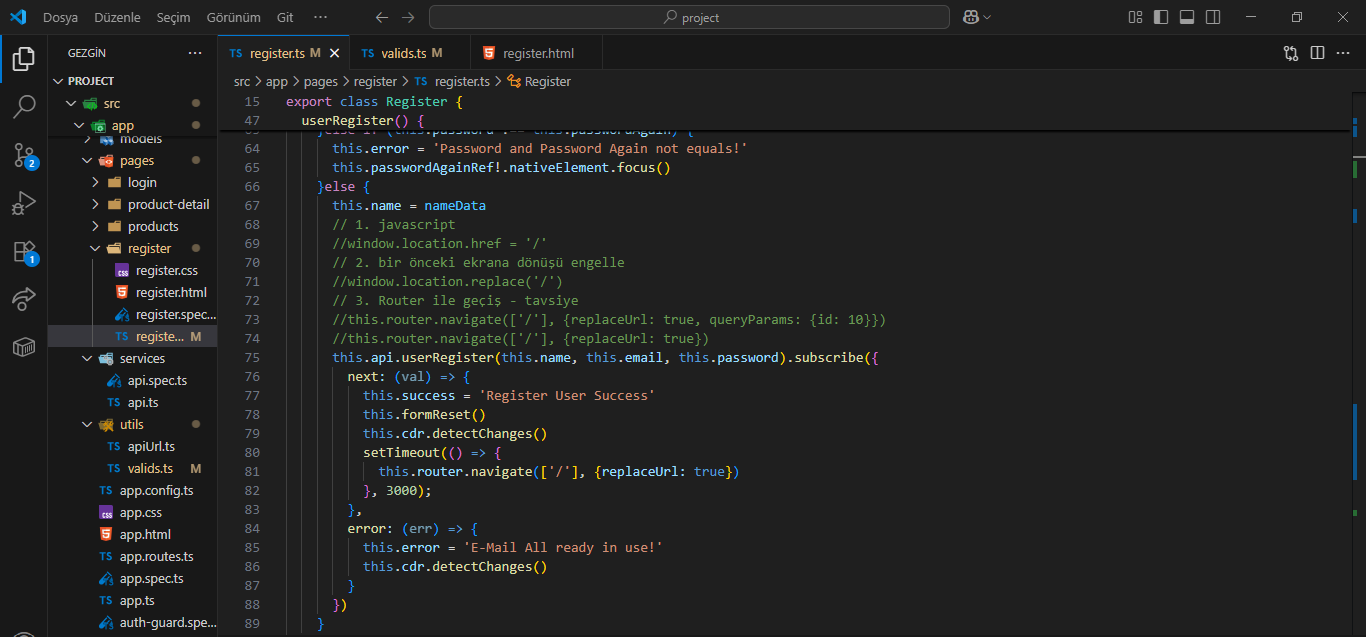
✅ Bu yazım daha okunabilir, temiz ve **HTML'e daha yakın**.

**📦 Özet**

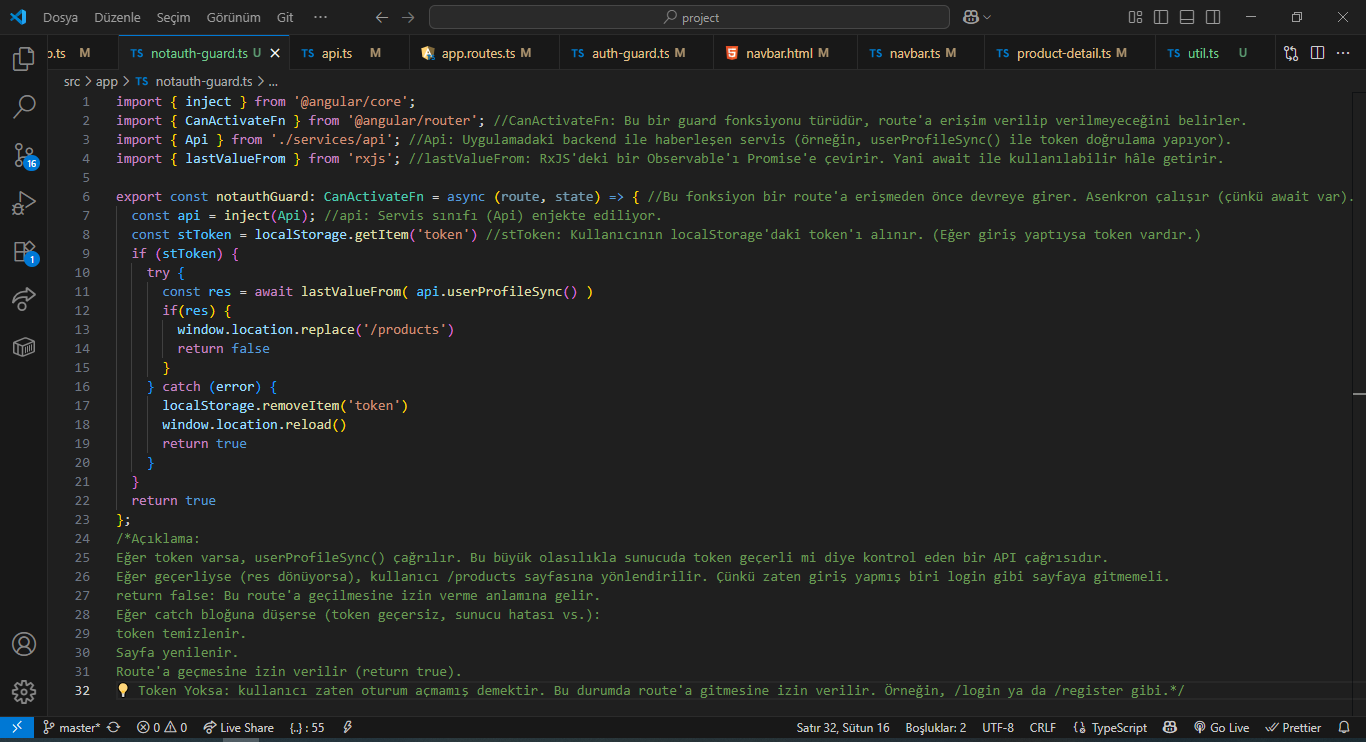
| **Sembol** | **Anlamı** |
| --- | --- |
| @if (...) {} | Koşullu görünüm |
| @else {} | Alternatif içerik |
| @for (...) {} | Döngüyle içerik üretme |
| @ işareti | Angular 17+’nin yapısal kontrol sözdizimi başlatıcısı |

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

### Days\_53: Angularda input selector \*\*FrontEnd/Project---> 12/07/2025



### Days\_57: notfound sayfası tasarımı + navbar içeriği \*\*FrontEnd/Project---> 12/07/2025



**🔁 Observable Nedir?**

**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.

**🔄 lastValueFrom Nedir?**

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

**✅ Neden 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…

Days\_58: çıkış işlemleri \*\*FrontEnd/Project---> 12/07/2025

**🧠 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.

**🧠 Gelişmiş: Sunucu Taraflı Session Kullanımı**

Eğer **sunucu taraflı session** (örneğin ASP.NET, Spring, Laravel) kullanıyorsan:

* Angular sadece HttpOnly Cookie içeren talepler gönderir.
* Angular, session ID’yi kendisi tutmaz.
* Bu yapıda frontend sadece **yetkilendirme gerektiren endpointlere** istek atar.

Ama Angular tarafında yine de kullanıcı arayüzünde durumu kontrol etmek için local/sessionStorage kullanılır.

**🎯 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