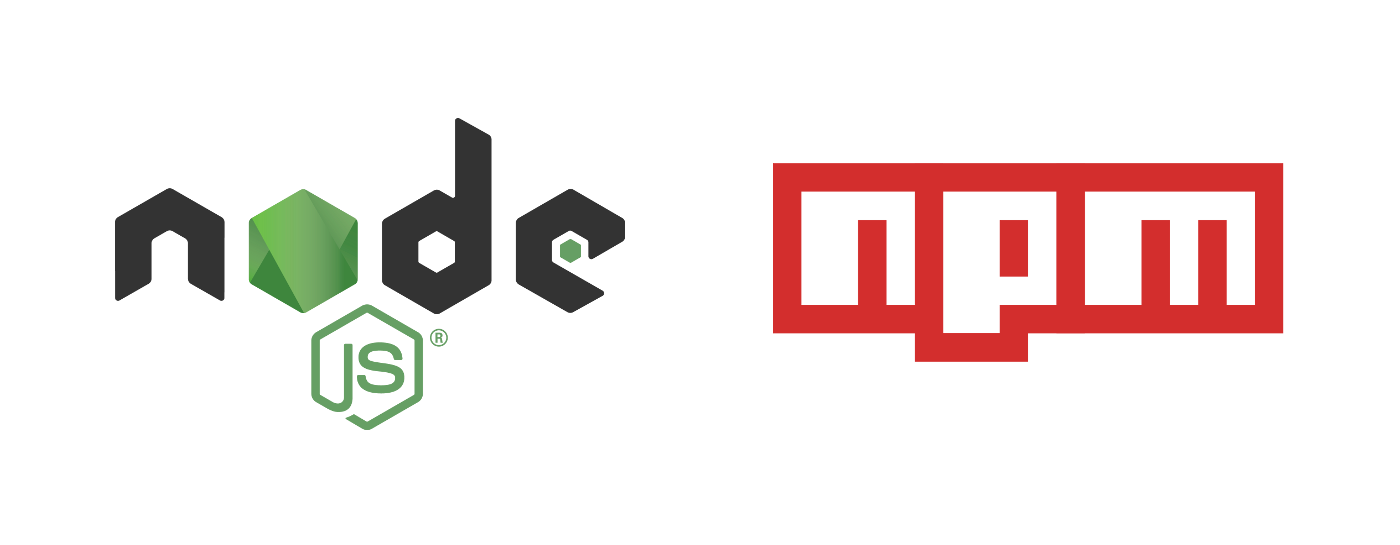
# Days\_66 Node.js nedir? gerekli kurulumlar \*\*work-space/node.js---> 30/08/2025

**KURULUM 🡪 Proje oluşturma:**

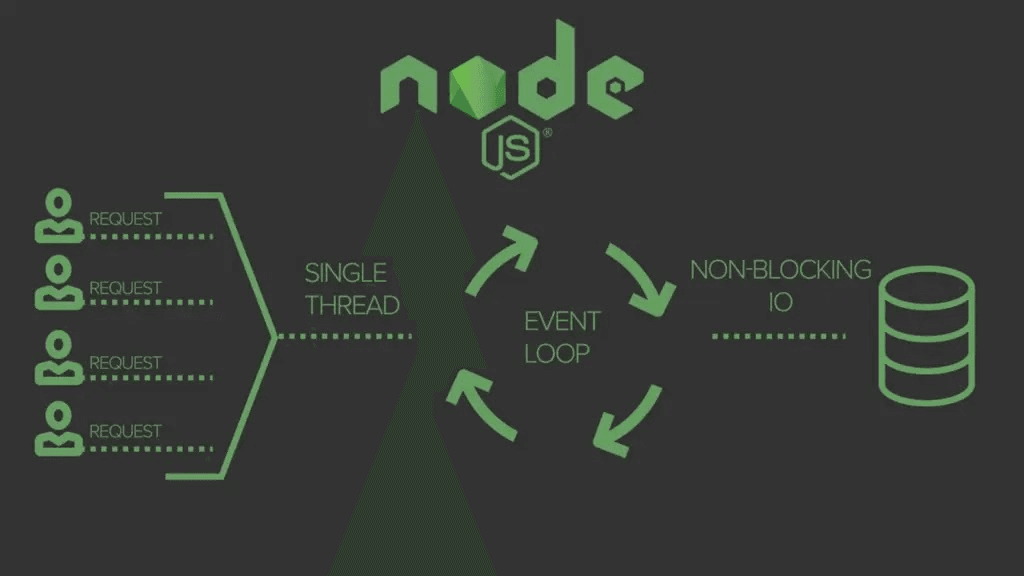
* npm init -y && npm install express && npm install -D typescript ts-node @types/node @types/express jest ts-jest @types/jest && npx tsc --init && npx ts-jest config:init
* npm i nodemon
* npm i ejs
* npm i bodey-parser



**Node.js'in Günümüzdeki Yeri ve Önemi**

**Node.js**, JavaScript'i sunucu tarafında (server-side) çalıştırmayı mümkün kılan, açık kaynaklı ve olay tabanlı (event-driven) bir çalışma ortamıdır. Google'ın Chrome V8 JavaScript motoru üzerine inşa edilmiştir. Geleneksel sunucu dillerinin (PHP, Java, Python gibi) aksine, Node.js eşzamansız (asynchronous) ve engellemeyen (non-blocking) I/O modeliyle öne çıkar. Bu model, özellikle gerçek zamanlı uygulamalar, mikro hizmetler ve yoğun veri akışının olduğu sistemler için idealdir.

Günümüzde Node.js, modern web geliştirmesinin en önemli bileşenlerinden biri haline gelmiştir. Artık sadece basit web sunucuları kurmakla kalmıyor, aynı zamanda karmaşık kurumsal uygulamalardan mobil uygulama backend'lerine, hatta masaüstü uygulamalarına kadar birçok alanda kullanılıyor.



**Node.js'i Bu Kadar Önemli Kılan Özellikler**

**1. Yüksek Performans ve Ölçeklenebilirlik**

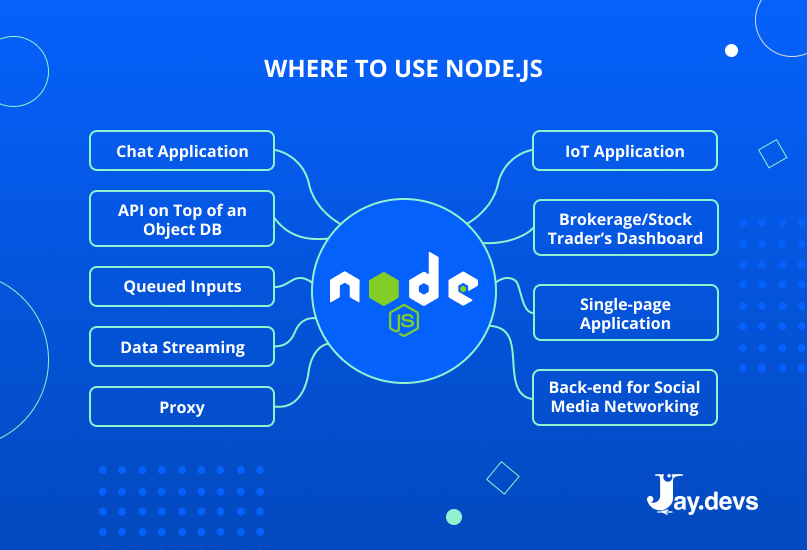
Node.js'in en büyük avantajı, eşzamansız mimarisidir. Bir istek geldiğinde, Node.js o isteği işlemini engellemeden arka plana atar ve bir sonraki isteği kabul etmeye devam eder. İşlem tamamlandığında, bir geri arama (callback) fonksiyonuyla sonuçları döner. Bu sayede, aynı anda binlerce bağlantıyı çok daha az sistem kaynağıyla yönetebilir. Bu özellik, özellikle anlık mesajlaşma, canlı yayın ve online oyunlar gibi gerçek zamanlı uygulamalar için hayati önem taşır.

**2. JavaScript Ekosistemi**

JavaScript, günümüzün en yaygın kullanılan programlama dillerinden biridir. Node.js sayesinde frontend (ön yüz) ve backend (arka yüz) geliştirme aynı dil üzerinde yapılabilir. Bu, geliştirici ekiplerin daha verimli çalışmasını sağlar ve kod paylaşımını kolaylaştırır. Ayrıca, **npm (Node Package Manager)** adı verilen devasa bir paket yöneticisi ekosistemine sahiptir. npm'de, hemen hemen her ihtiyaca yönelik milyonlarca açık kaynaklı kütüphane ve modül bulunur. Bu durum, geliştirme süreçlerini hızlandırır ve tekrarlayan işleri azaltır.

**3. Mikro Hizmet Mimarisi (Microservices)**

Günümüzdeki modern yazılım mimarilerinde büyük monolitik uygulamalar yerine, bağımsız ve küçük hizmetlerden oluşan mikro hizmetler tercih edilir. Node.js'in hafif yapısı ve modüler mimarisi, mikro hizmetler oluşturmak için mükemmel bir uyum sağlar. Her mikro hizmet, kendi Node.js sunucusunda çalışabilir ve farklı teknolojilerle bile iletişim kurabilir.



**Node.js'in Kullanım Alanları**

* **Gerçek Zamanlı Uygulamalar:** Anlık mesajlaşma, canlı sohbet uygulamaları, online oyunlar, gerçek zamanlı bildirim sistemleri. **Socket.IO** gibi kütüphaneler sayesinde bu tür uygulamaları geliştirmek oldukça kolaydır.
* **API Geliştirme:** RESTful API'ler ve GraphQL sunucuları oluşturmak için idealdir. **Express.js**, **Koa.js** ve **NestJS** gibi frameworkler, bu alanda sıkça kullanılır.
* **Mikro Hizmetler:** Büyük uygulamaları küçük, bağımsız ve yönetilebilir parçalara ayırmak için kullanılır.
* **Veri Yoğun Uygulamalar:** Veri akışının sürekli olduğu ve I/O işlemlerinin yoğun olduğu uygulamalar (örneğin, dosya yükleme servisleri, veri işleme boru hatları).
* **Tek Sayfa Uygulamaları (SPA) için Backend:** **React**, **Angular** veya **Vue.js** gibi frontend frameworkleri ile geliştirilen SPA'lar için güçlü bir backend sunar.
* **Komut Satırı Araçları:** Geliştiricilerin işlerini otomatikleştirmek için kullandığı araçlar (örneğin, **Grunt** ve **Gulp** gibi görev çalıştırıcılar).

**Node.js'in Avantajları ve Dezavantajları**

**Avantajları**

* **Yüksek Performans:** Eşzamansız yapısı sayesinde yüksek trafikli uygulamalar için idealdir.
* **Tek Dil:** JavaScript ile hem frontend hem de backend geliştirebilme imkanı sunar.
* **Geniş Ekosistem:** npm sayesinde geliştirme süreçleri oldukça hızlanır.
* **Ölçeklenebilirlik:** Küçük hizmetler halinde kolayca ölçeklenebilir.
* **Büyük Topluluk Desteği:** Aktif ve büyüyen bir geliştirici topluluğu vardır.

**Dezavantajları**

* **Yoğun İşlem Gerektiren Uygulamalar:** CPU-yoğun (örneğin, karmaşık matematiksel hesaplamalar) uygulamalarda eşzamanlı mimari avantaj sağlamaz ve performans sorunlarına yol açabilir.
* **Callback Cehennemi (Callback Hell):** Eşzamansız yapı, karmaşık iç içe geçmiş callback fonksiyonları nedeniyle kodun okunabilirliğini zorlaştırabilir. Bu sorun **async/await** gibi modern yaklaşımlarla büyük ölçüde çözülmüştür.
* **Olgun Olmayan Frameworkler:** Diğer dillerdeki gibi oturmuş, standart bir framework ekosistemi hala gelişmekte olan bir yapıdadır.

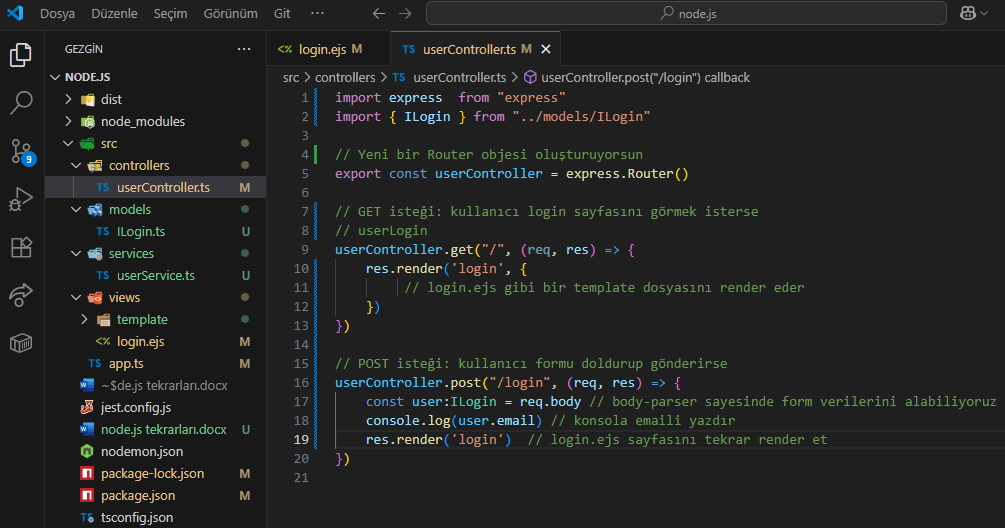
**Özetle Node.js Nedir?**

* **Tanım**: Node.js, tarayıcı dışında JavaScript çalıştırmamızı sağlayan bir **çalışma ortamıdır (runtime)**.
* **Çekirdeği**: Google’ın Chrome’da kullandığı **V8 JavaScript motoru** üzerine kuruludur.
* **Özellikleri**:
  + Tek iş parçacıklı (single-threaded) ama **event-driven** mimariye sahiptir.
  + **Non-blocking I/O** sayesinde çok sayıda istemciyle aynı anda iletişim kurabilir (özellikle web sunucuları için güçlüdür).
  + Sunucu tarafında JavaScript ile API’ler, web uygulamaları, gerçek zamanlı uygulamalar (chat, oyun, IoT vb.) geliştirmemizi sağlar.

**Express.js Nedir?**

* Express, **Node.js için geliştirilmiş bir web uygulama çatısı (framework)**’dır.
* Node.js’in ham yapısını daha kolay ve düzenli kullanabilmemizi sağlar.
* Sağladıkları:
  + **Router sistemi** → URL’leri ve HTTP isteklerini yönetme
  + **Middleware desteği** → istek/cevap sürecinde araya girip işlem yapabilme
  + **Template engine entegrasyonu** → res.render() ile HTML şablonlarını server-side render etme
  + Daha okunabilir ve modüler bir yapı

**Koddaki Örnek Üzerinden Açıklama**



**Burada olan şeyler:**

1. express.Router() → küçük bir **mini uygulama** gibi çalışır. Belirli path’ler için get, post, put, delete gibi route’lar tanımlayabilirsin.
   * Sonra bu userController’ı app.ts veya server.ts dosyanda app.use("/users", userController) diyerek ana uygulamaya bağlarsın.
2. res.render('login') → Express’in template engine ile (mesela **EJS**) HTML sayfasını oluşturup istemciye göndermesi.
3. req.body → Kullanıcının gönderdiği form verilerini almak için. Bunun çalışması için Express’te body-parser ya da express.json() / express.urlencoded() middleware’i gerekir.

**Genel Akış**

* Kullanıcı /users/ adresine gider → GET / çalışır → login sayfası açılır.
* Kullanıcı formu doldurup gönderir → POST /login çalışır → bilgiler backend’e gelir → log basılır → tekrar login sayfası render edilir.

**Body-parser nedir?**

* body-parser, Express.js uygulamalarında **HTTP isteği ile gelen verileri (request body)** kolayca okumak için kullanılan bir **middleware**’dir.
* Normalde Node.js, gelen ham HTTP isteğini **string/buffer** halinde bırakır. Bu ham veriyi anlamlı hale getirmek (JSON, form verisi vs.) için body-parser veya Express’in kendi gömülü fonksiyonları kullanılır.

**Ne İşe Yarar?**

1. **JSON verisi** (application/json) gönderildiğinde, req.body içinden doğrudan JS objesi olarak okunmasını sağlar.
2. **Form verisi** (application/x-www-form-urlencoded) gönderildiğinde, req.body içinden key-value olarak alınmasını sağlar.

**Kullanımı**

Express 4.16+ sürümünden itibaren body-parser’ın çoğu özelliği Express’in içine gömüldü. Yani artık şunu yazmak yeterli:

**userController.post("/login", (req, res) => {**

**const user: ILogin = req.body** // işte burası body-parser/express.json sayesinde çalışıyor

**console.log(user.email)**

**res.render('login')**

**})**

👉 Yani özetle:  
Body-parser = gelen request body verilerini (JSON, form vs.) okunabilir hale getiren bir Express middleware’i.

EJS: “**Node.js’de neden <%= %> gibi EJS tagları kullanılıyor?**” sorunun cevabı aslında **sunucu taraflı HTML üretme ihtiyacından** geliyor.

**🚀 Neden Böyle Bir Yapı Var?**

**1. Dinamik HTML üretmek için**

* Normalde HTML statiktir → içine yazdığın şey hep aynıdır.
* Ama web uygulamalarında çoğu zaman değişken veriler göstermek gerekir (ör. kullanıcı adı, ürün listesi, yorumlar vs).
* İşte <%= %> sayesinde **JavaScript değişkenlerini HTML içine gömebiliyorsun**.

**2. JavaScript’i doğrudan HTML’de çalıştırmak için**

* EJS, HTML’in içine **JavaScript kodu yazmana izin verir**.
* Bu sayede if/else, for döngüleri, listeleme gibi şeyleri HTML tarafında kolayca yaparsın.

**3. MVC mantığına uyduğu için**

* Node.js + Express genelde **MVC (Model–View–Controller)** mantığında kullanılır:
  + **Model** → veri (ör. veritabanından gelen kullanıcı bilgisi)
  + **Controller** → veriyi alıp işleyip şablona yollar
  + **View (EJS)** → o veriyi HTML’e yerleştirir

Yani <%= %> aslında **“controller’dan gelen veriyi view’da göstermek”** işlevini yapıyor.

**4. Güvenlik (Escape mantığı)**

* <%= %> → XSS saldırılarına karşı veriyi **escape ederek** basar (HTML kodu çalıştırmaz).
* <%- %> → escape etmeden HTML’i direkt basar.

Böylece geliştirici isterse güvenli şekilde veri gösterebilir, isterse HTML render ettirebilir.

**✅ Özet**

Node.js’de EJS (ve benzeri templating engine’lerdeki tag yapısı):

* **HTML’i dinamik hale getirmek** için,
* **JavaScript kodunu doğrudan şablon içinde çalıştırmak** için,
* **MVC düzenine uygun “veri → view” akışını sağlamak** için kullanılıyor.

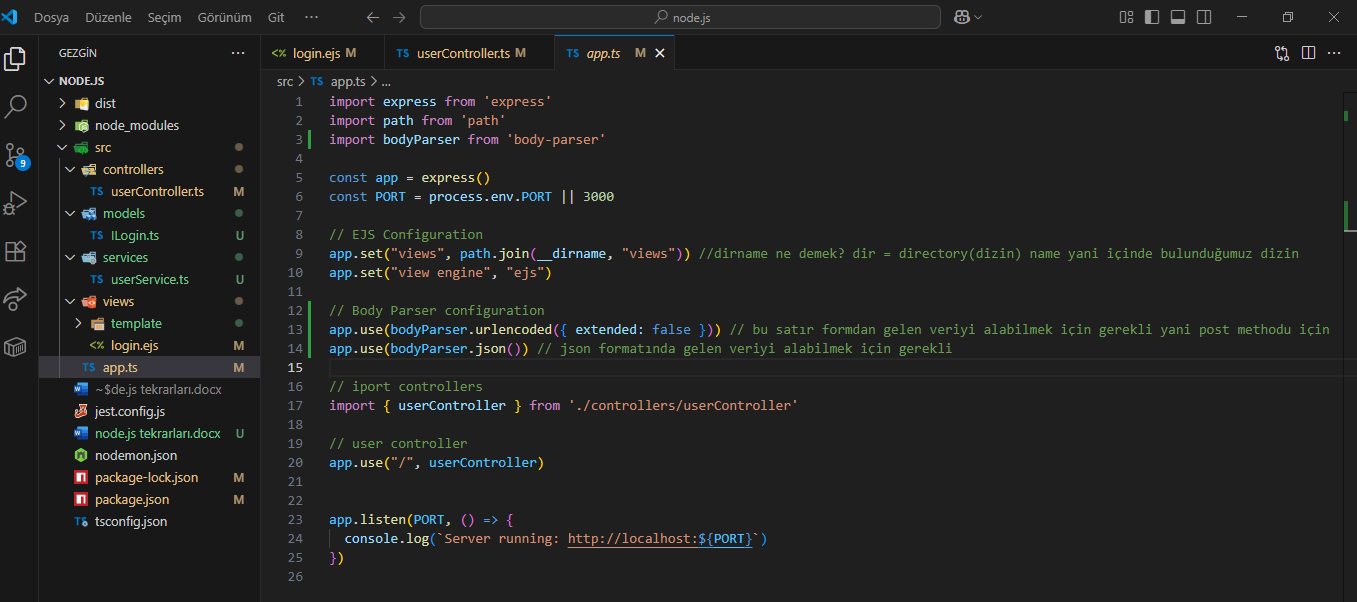
**📌 Escape Nedir?**

Escape etmek, Veriyi olduğu gibi değil, **özel karakterlerini dönüştürerek** yazdırmak demektir.

**Escape etmek** = özel HTML/JS karakterlerini güvenli eşdeğerlerine dönüştürmek

* + < → &lt;
  + > → &gt;
  + & → &amp;
  + " → &quot;
* <%= %> → escape eder (güvenli).
* <%- %> → escape etmez (HTML çalışır).

App.ts dosyası:



**1. Import Kısımları**

* **express** → Web sunucusu oluşturmak için ana framework.
* **path** → Dosya ve klasör yollarını yönetmek için Node.js'in yerleşik modülü.
* **body-parser** → POST veya PUT isteklerindeki verileri (req.body) yakalamak için kullanılan middleware.

**2. Sunucu Tanımı**

**const app = express()**

**const PORT = process.env.PORT || 3000**

* app → Express uygulaması.
* PORT → Eğer ortam değişkeninde (process.env.PORT) bir port varsa onu kullanır, yoksa **3000** portunu kullanır.

**3. EJS Yapılandırması**

**app.set("views", path.join(\_\_dirname, "views"))**

**app.set("view engine", "ejs")**

* views → EJS şablonlarının bulunduğu klasörü tanımlıyor.
  + \_\_dirname → app.ts dosyasının bulunduğu dizin.
  + path.join(\_\_dirname, "views") → src/views klasörünü işaret ediyor.
* view engine → EJS’yi template engine olarak ayarlıyor.

**4. Body Parser Yapılandırması**

**app.use(bodyParser.urlencoded({ extended: false }))**

**app.use(bodyParser.json())**

* **urlencoded** → application/x-www-form-urlencoded tipinde gelen verileri çözmek için (formlardan gelen veriler).
  + extended: false → sade key-value eşleşmeleri için.
* **json** → application/json tipinde gelen verileri çözmek için.

Böylece req.body içinden email, password gibi verilere erişebilirsin.

**5. Controller İçe Aktarma**

**import { userController } from './controllers/userController'**

* userController → Kullanıcı işlemleri (/login vs.) için yazdığın router.
* controllers/userController.ts dosyasındaki route tanımlarını burada uygulamana bağlıyorsun.

**6. Router Kullanımı**

**app.use("/", userController)**

* / ile başlayan tüm istekler userController tarafından işlenecek.
* Örn: /login isteği userController.post('/login', ...) metoduna düşecek.

**7. Sunucu Başlatma**

**app.listen(PORT, () => {**

**console.log(`Server running: http://localhost:${PORT}`)**

**})**

* Belirlenen portta Express sunucusu çalıştırılıyor.
* Terminale “Server running: [http://localhost:3000”](http://localhost:3000%E2%80%9D) yazdırıyor.

**📌 Özetle**

Bu app.ts şunları yapıyor:

1. Express app başlatıyor.
2. Görünüm motoru olarak EJS’yi ayarlıyor.
3. Gelen form ve json verilerini çözmek için body-parser ekliyor.
4. Kullanıcı controller’ını (userController) uygulamaya bağlıyor.
5. 3000 portunda sunucuyu ayağa kaldırıyor.

# Days\_67 Login-Register işlemleri \*\*work-space/node.js---> 31/08/2025

EJS’de Dinamiklik

Ejs ile statik olan html’yi dinamik hale getiriyoruz. Mesela html içinde if döngüsü yazıp toplama işlemi yaptıramazsın. Html sadece tasarım kısmıdır. Tema motoru tarafından ejs tagları ile anlamlandırıp ve sistemi daha dinamik hale getiriyor. Yani:

* **HTML** = tasarım iskeleti
* **EJS** = bu tasarımı **dinamik hale getiren motor**

**🔹 HTML**

* **Statik bir yapıdır.**
* Yani sadece yazdığın şekilde tarayıcıya gider.
* İçinde değişken, koşul (if), döngü (for) çalışmaz.
* Görevi tamamen **sayfanın iskeletini ve tasarımını** vermek.

**🔹 EJS (Embedded JavaScript)**

* HTML’nin içine **JavaScript gömmemizi** sağlar.
* Bir “template engine” yani **şablon motorudur**.
* <%= %>, <% %>, <%- %> gibi özel tag’lar kullanarak:
  + **Değişkenleri yazdırabilir**
  + **Koşullar** (if/else) çalıştırabilir
  + **Döngüler** (for/each) çalıştırabilir
  + **Parçalı view’leri include** edebilirsin

Node.js’te include mantığı:

**EJS** kullanırken sayfa parçalarını tekrar tekrar yazmamak için include mantığı kullanılır

**🔹 Normal HTML’de Durum**

Normalde HTML’de **aynı header, footer, menü** kodlarını her dosyada yazmak zorunda kalırsın.  
Bu hem zahmetli hem de değişiklik yapınca her dosyada tek tek güncellemen gerekir.

**🔹 EJS’de Include Mantığı**

EJS sana include özelliği veriyor:

* Ortak parçaları (örneğin: **header.ejs, footer.ejs, navbar.ejs**) ayrı dosyaya ayırabilirsin.
* Sonra include ile istediğin yerde çağırırsın.

**🔹 Örnek Kullanım**

📂 Klasör yapısı:

views/

├── template/

│ ├── boostrap.ejs

├── login.ejs

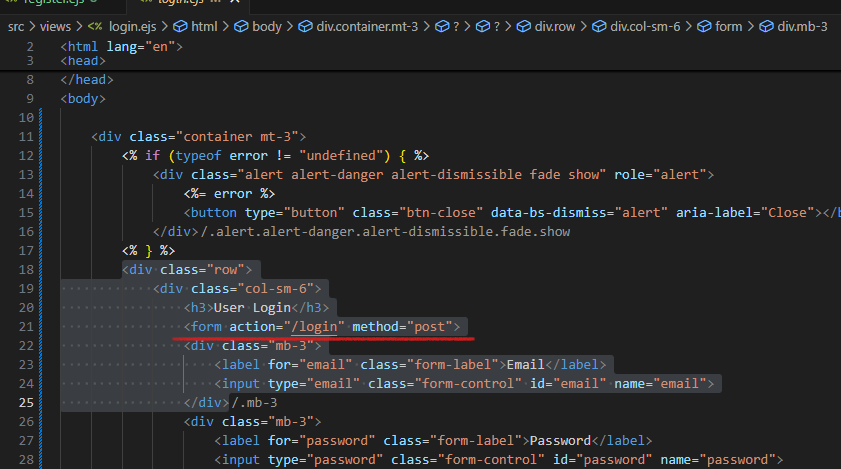
├── register.ejs

**🔹 Buradaki Farklar**

* <%- include(...) %> → include edilen dosyanın içeriğini **render edip ekler**.
* Boostrap.ejs dosyasını 20 sayfa da kullansan sadece **1 dosya** üzerinde değişiklik yaparsın.

👉 Özet:  
**include** = tekrar eden kodları **parçalara ayırıp yeniden kullanabilmeni sağlayan mantık**.

Login.ejs



Formlarda kullanılan **GET** ve **POST** metodlarının farkı hem **güvenlik** hem de **kullanım amacı** ile ilgilidir.

**🔹 1. GET Methodu**

* **Veriyi URL üzerinden gönderir** (?email=abc@mail.com&password=1234 gibi).
* URL'de görüneceği için herkes görebilir, loglarda saklanır, paylaşılabilir.
* **Güvenli değil**, o yüzden **şifre, kimlik bilgisi** gibi kritik şeyler için **ASLA kullanılmaz**.

✅ Kullanım alanları:

* **Arama/Filtreleme** = GET
* **Ürün linki paylaşımı** = GET

**🔹 2. POST Methodu**

* **Veriyi HTTP body** kısmında gönderir (URL'de görünmez).
* **Daha güvenlidir** çünkü URL’ye yazılmaz (ama tabii HTTPS kullanılmazsa yine ağda yakalanabilir).
* Büyük veriler gönderilebilir (form, dosya upload).

✅ Kullanım alanları:

* Kullanıcı girişleri (login)
* Kayıt olma (register)
* Form ile veri kaydetme (yorum ekleme, sipariş verme, dosya yükleme vb.)

**🔹 Bir örneğine uyarlayalım:**

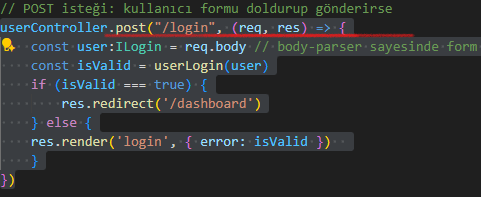
* **Login formu** → POST kullanılmalı. Çünkü email+şifreyi **gizlemek** gerekir.
* **Sahibinden ürün linki** → GET kullanılmalı. Çünkü linki arkadaşına gönderdiğinde sayfa açılmalı, içinde kritik bilgi yok, sadece ürünün id’si veya adı var (/ilan/12345).
* **Arama kutusu (Google, sahibinden arama çubuğu)** → GET. Çünkü aradığın kelime (?q=iphone) URL’ye yazılsa sorun olmaz, hatta iyi olur (bookmark, paylaşım yapılabilir).

**🔹 Özet Tablo**

| **Özellik** | **GET** | **POST** |
| --- | --- | --- |
| Veri nereye gider? | URL query string | HTTP body |
| Güvenlik | Düşük (URL’de görünüyor) | Daha güvenli (body’de saklı) |
| Veri boyutu | Limitli (URL uzunluğu) | Daha büyük veri/dosya gönderilir |
| Bookmark | Yapılabilir | Yapılamaz |
| Kullanım alanı | Arama, filtreleme, paylaşılabilir linkler | Login, register, yorum, ödeme, dosya yükleme |

(\*\* 🔖 Bookmark (Türkçe: Yer imi))

userController.ts



Node.js / Express’te controller metodlarının parametreleri genelde şu şekilde olur:

**userController.post("/login", (req, res) => { ... })**

**1. req → Request (İstek)**

Kullanıcının **sunucuya gönderdiği tüm bilgileri** taşır.

* Formdan gelen veriler (ör. email, password) → req.body
* URL parametreleri (ör. /users/5) → req.params
* Query string (/search?keyword=node) → req.query
* Header bilgileri (ör. Authorization, Content-Type) → req.headers
* Cookie bilgileri → req.cookies

📌 örnekte:

const user: ILogin = req.body

Formdaki email ve password değerleri burada yakalanıyor.

**2. res → Response (Yanıt)**

Sunucunun **kullanıcıya geri döndüğü cevabı** temsil eder.

* **HTML sayfası render etme** → res.render("sayfa")
* **JSON veri döndürme** → res.json({ message: "ok" })
* **Yönlendirme (redirect)** → res.redirect("/dashboard")
* **Basit text / status döndürme** → res.send("Merhaba"), res.status(404).send("Bulunamadı")

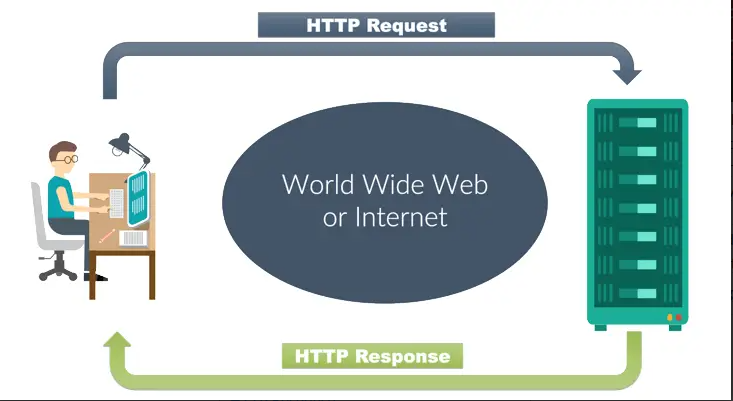
📌 örnekte:

**res.redirect('/dashboard')**

Başarılı girişte kullanıcıyı /dashboard sayfasına yönlendiriyor.

**res.render('login', { error: isValid })**

Eğer hata varsa login sayfasını tekrar gösteriyor, error değişkenini sayfaya gönderiyor.



✅ Özetle:

* req = Kullanıcıdan gelen istek (data, headers, parametreler).
* res = Sunucudan giden cevap (html, json, redirect, status).