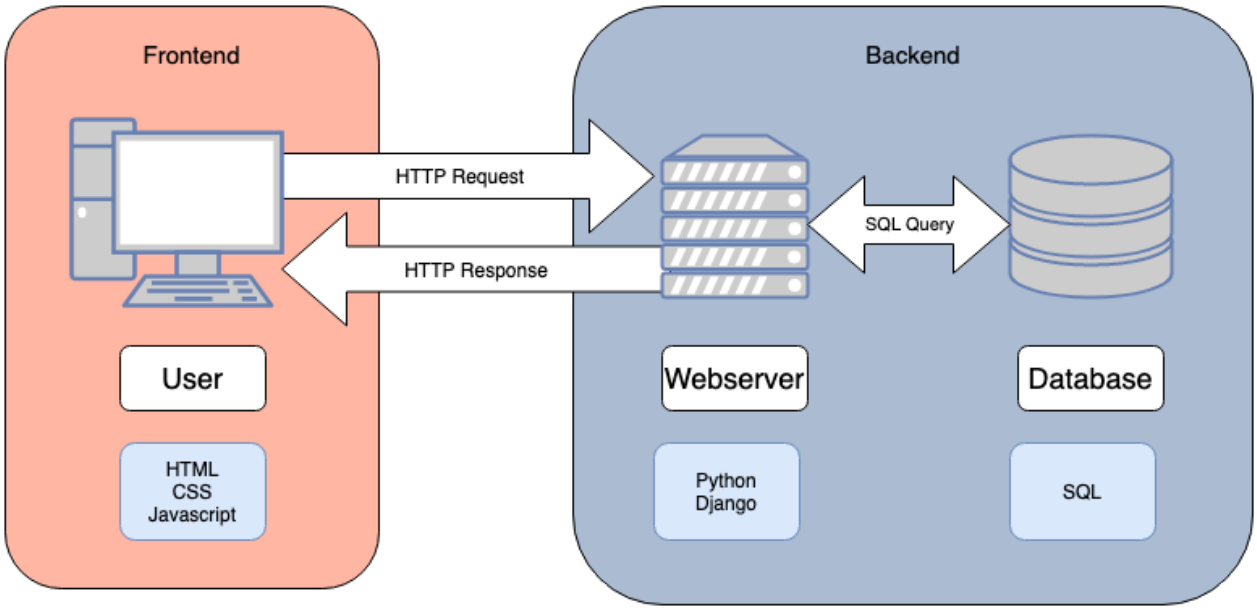
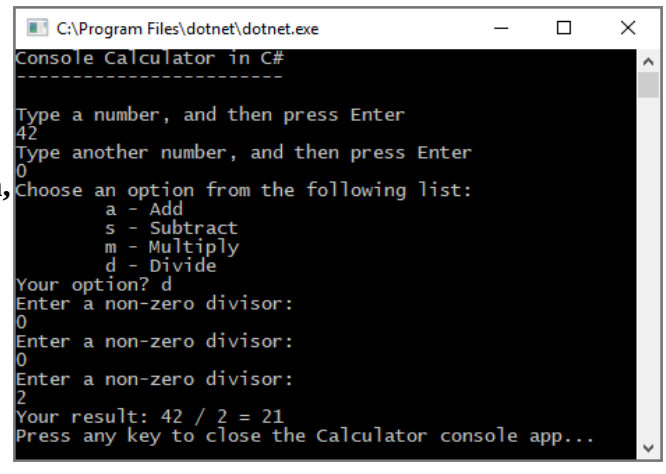
UYGULAMA MİMARİSİ 131



1-) BİLGİSAYAR PROGRAMI

Herkesin bildiği gibi bir bilgisayar programı, girdi (input) verilerini alıp, belirli işlemlerde geçirerek bir çıktı (output) üreten kod parçacığıdır. 2 tane sayı alıp 4 işlemden birini yapan, bir tane metin (string) ifadesini alıp, içindeki belirli karakterleri (char) silen, bir dosyayı okuyup içindeki herhangi bir kelimenin sayısını bulan vesaire şeyler bilgisayar programıdır.

1. örneğimizde verilen **2 sayı** girdi, **4 işlemden biri** işlem ve bu **4 işlemden birinin sonucu** çıktıdır.
2. örneğimizde **metin** girdi, **içindeki belirli karakterleri silmek** işlem, **belirli karakterleri silinmiş metin** çıktıdır.
3. örneğimizde **bir dosya** girdi, **bu dosya içindeki bir kelimenin sayısını bulmak** işlem ve **bu kelimenin sayısı** çıktıdır.



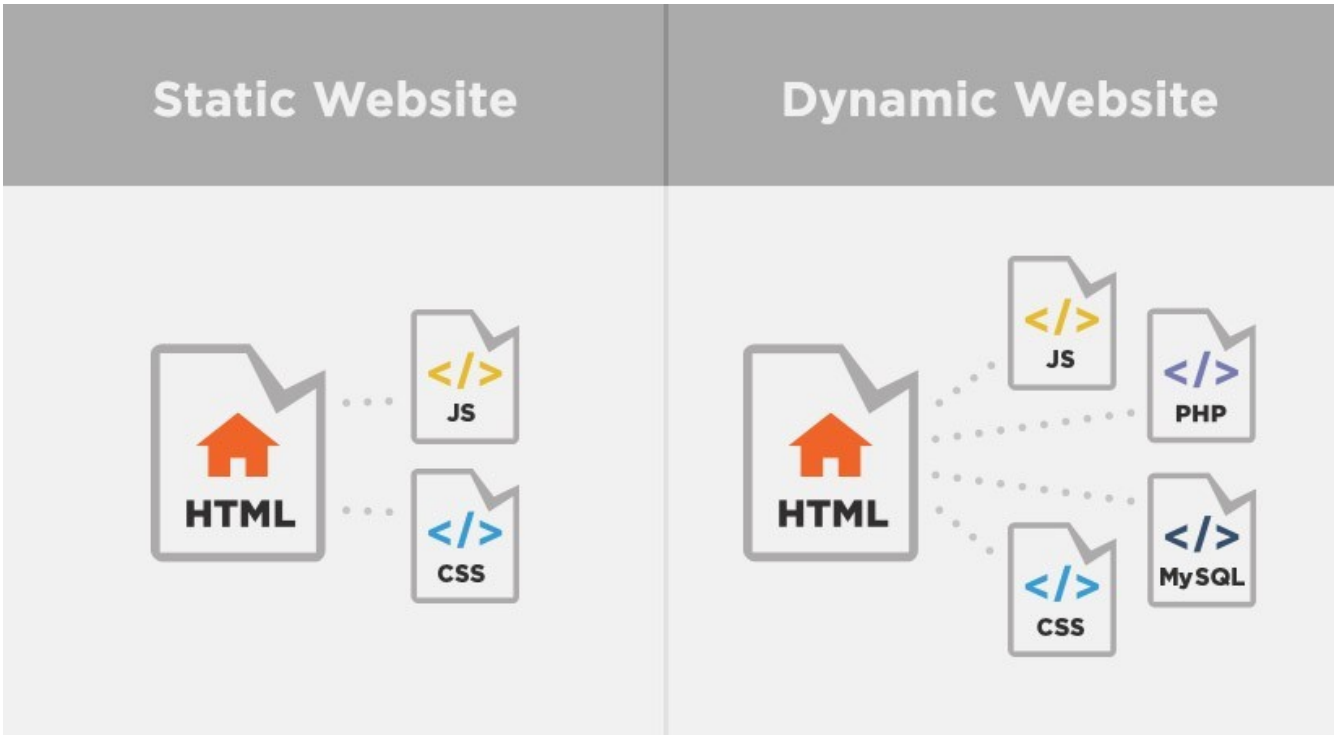
Bu üç örneğimizde temel olarak girdi, işlem ve çıktı kavramlarını örneklerle anlattık. Ancak bu girdilerin **nereden, kim tarafından, nasıl verileceğini,** işlemlerin **nerede, kim tarafından, nasıl yapılacağını** ve çıktının **nerede, kim tarafından, nasıl sunulacağını** anlatmadık.

Bilgisayar programlama ile başlangıç düzeyinde uğraştığımızda, yukarıda verdiğimiz 3 örneğe

benzer şeyler bir çok kere yaparız. Örneğin kendi yazdığımız bir programda bir dosyayı okuyup bu dosyada geçen bir kelimenin sayısını bulup, bu kelimenin sayısını metin analizi fonksiyonlarrına gönderebiliriz. Bu gibi durumlarda girdiyi programcı verir ve çıktıyı yine programcı kendi yazdığı başka bir işlemde kullanır. Veya basit bir konsol üzerinde çalışan uygulamada, kullanıcıdan konsol üzerinden 2 tane sayı isteyip bunları toplayıp çıktıyı yine konsol üzerine yazdırabiliriz. Bu programlar son kullanıcı için üretilecek çıktının ara aşamalarını yapan veya makul ve mantıklı bir sebeple kullanılmayacak küçük programlardır. Ancak web, mobil v.b platformlarda son kullanıcı için üretilen programlarda kullanıcıdan girdilerin ne şekilde toplanacağı, ne şekilde işleneceği ve ne şekilde çıktının gösterileceği önem kazanır. **İşte zurnanın zırt dediği yer burasıdır.**

2-) DATABASE, BACKEND FRONTEND V.B MUHABBETLER

Yine örnekler üzerinden gidelim. Elimizde 2 tane web sayfası var. 1.si **kaliteliporno.com**, diğeri **türkiyevipeskort.com**.

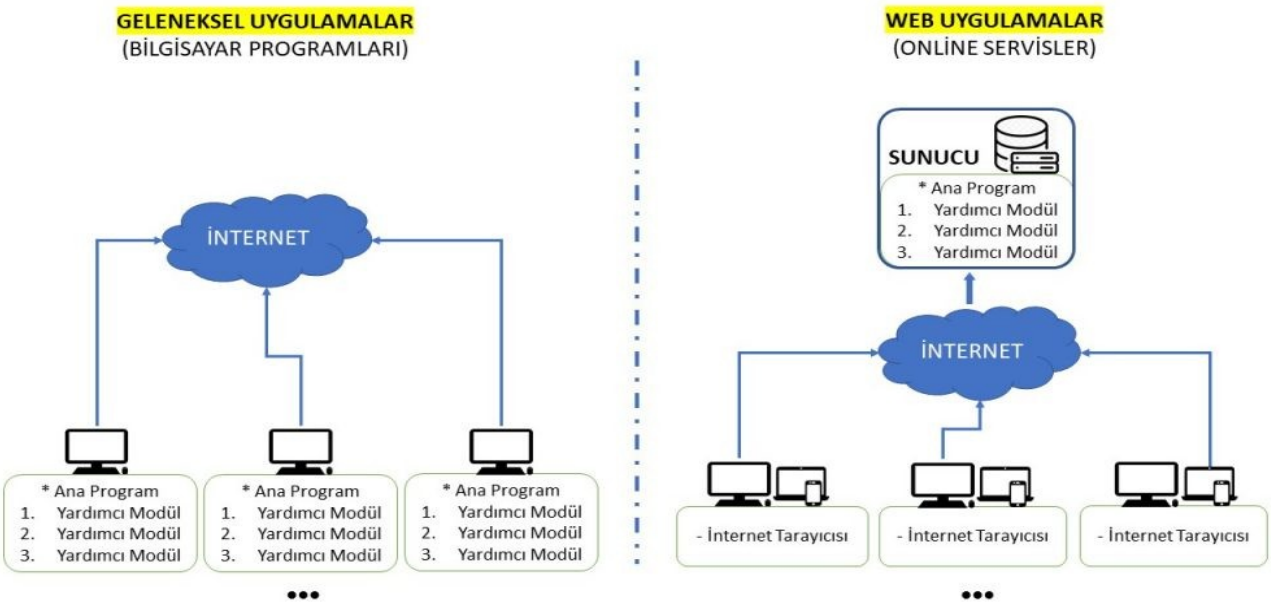


türkiyevipeskort.com adlı sitemizde seks emekçisi ablalarımızın fotoğraflarını, onları anlatan yazıları ve hizmet satın almak için ulaşmamız gereken telefon numalarını görüyoruz. Ancak kayıt olma, giriş yapma, çıkış yapma, eskort ablalarımızı puanlama, onlar hakkında bir metin kutusuna yorum yazma ve eğer böbrek avi ise böbrek avi diye belirtme, pzevenklerine küfürlü mesaj atma gibi işlemleri yapamıyoruz. Jargon kulanırsak bu **static**, yani **kullanıcı etkileşimi olmayan veya sınırlı olan** bir web sayfası. İşte bu tip web sayfaları genelde internet kafelerin camındaki websitesi yapılır ilanlarında website olarak adlandırılan şeylere denir. Basit **HTML** dosyalarından ibarettir. Biraz **CSS** ile pezevenk numaraları renklendirilmiş, biraz **JavaScript** her bir seks emekçisi abla için birkaç fotoğraftan oluşmuş bir slayt show yapılmış olabilir. Ancak kullanıcıdan bir veri almaz, bu verileri bir yerde işlemez ve bu işlenen verilerin çıktısını web sayfasında göstermez. Tüm türkiyeden, ortadoğu ve balkanlardan giren 1-2 saatlik seks hizmeti alma umudu taşıyan insanlar veya vpn ile porno sitesine girmeyi bilmeyen ergenler aynı içeriği görür. Adı üstünde **static** web sayfasıdır. Bundan 10 sene önce, almancı takılıp

ülkeye kesin dönüş yapmış internet kafeci 50 liraya bunu yapar. Bu sayfaya sağ tıklayıp kaynağı **view page source'**a tıklarsak sayfanın html dosyasının içeriğini görüntüleriz, bunun ötesinde bir numarası yoktur. Buna websitesi diyip geçeriz.

Şimdi **kaliteliporno.com**'a göz atalım. Bu güzide web sayfamızda o günün en çok izlenen pornolarının thumbnailleri listelenir. Pornolara tıkladığımızda porno videosu puanla, yorum yap bunların yanında puanlamak ve yorum yapmak için üyelik gerekir yazılarının görürüz. Geriye tıkladığımızda tekrar anasayfaya döneriz ve kayıt ol, giriş yap butonları bulunur. Kayıt ol'a tıkladığımızda yeni bir sayfa açılır ve bizden kullanıcı adı, şifre, mail adresi vesaire ister. Giriş yap butonuna tıkladığımızda bizden kullanıcı adı ve şifre ister. Giriş yaparsak profil sayfamızda en çok izlediğimiz pornoları, puanladığımız pornoları görebiliriz.

Yukarıda zaten kendimizi ele verdik. Bizden belirli bilgiler yani **girdi(input)** ister(örneğin kullanıcı adı ve şifremizi alır) . Bu girdileri alır **işlemden geçirir** (kayıtlı mıyız değil miyiz diye kontrol eder.) Ve bu işlemin sonucuda bizim için bir çıktı üretir. (Bizi profil sayfamıza yönlendirir.) Yani kullanıcı etkileşimine cevap verir, buna göre çıktı üretir. Ben ile başka bir kullanıcının favorilediği pornolar farklı, kullanıcı adı ve şifremiz farklı olduğundan **profil sayfamız farklıdır**. İşte bunlar dinamik websitesi yani **web uygulaması** dediğimiz mevzulardır. Ana sayfadaki porno thumbnailleri, butonlar html elemanlarıdır ve yine basit html sayfalarında bulunurlar. Ancak bu websitesi kullanıcı adı ve şifre form alanları ve giriş yap butonu ile bir web sunucusu ile bağlantı kurar. Bu web sunucusu python, js, java v.b dillerle yazılmış bazı programlar vastasıyla bağlantı isteğine göre işlem yapar. Yine örnek üzerinden gidersek, giriş yap butonuna tıkladığımızda kullanıcı adı ve şifre alanına yazdığımız veriler bir **HTTP POST** isteği ile bir web sunucusuna gider. Bu web sunucusunda bu **POST isteğini** işleyecek örneğin **python flask** ile yazılmış bir fonksiyon bulunur. Bu fonksiyon kullanıcı adı kısmına girdiğimiz veri bir veritabanında kayıtlı mı değil mi diye kontrol eder. Varsa bu kullanıcı adına karşılık gelen şifre ile bizim yazdığımız şifre aynı mı diye kontrol eder. Eğer o da aynı ise web sayfasına bir giriş başarılı mesajı gönderir. Bu mesajı alan websitemiz bizi profil sayfamıza yönlendirir. İşte bu işlemleri yapabilme kapasitesine sahip olan websiteleri dinamik websiteleri yani web uygulamalarıdır.

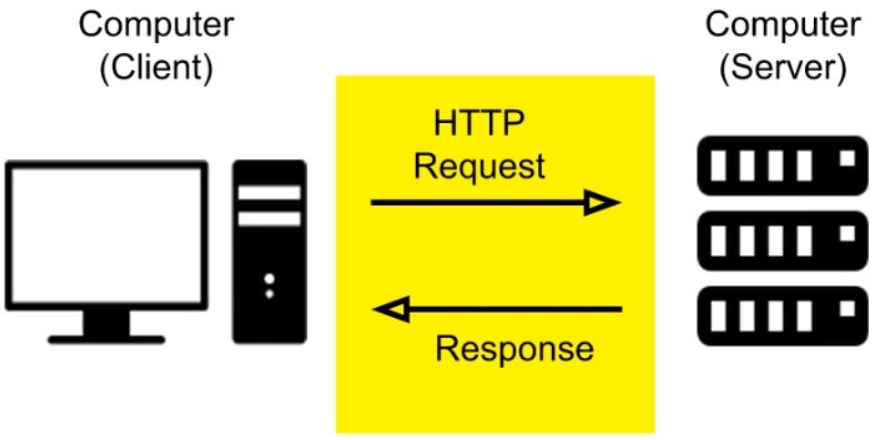


Yukarı verdiğimiz örnekler üzerinden hızlıca backend, frontend ve database mevzularını tanımlayalım. Porno thumbnaillerimizin, giriş yap butonunun olduğu html dosyası, buna şekil veren css ve javascript dosyaları güzide sitemizin frontend'didir. Giriş yap butonuna tıkladığımızda http post isteğini alan, bu isteğin içindeki kullanıcı adı ve şifreyi database'de kayıtı mı diye kontrol eden ve girişin başarılı olup olmadığına dair front-end'e mesaj (response) gönderen kısım backend'dir. Uygulama ile ilgili verileri tutan kısım, database'dir.

Front-end : Kullanıcıdan girdilerin alınıdğı ve kullanıcıya çıktıların gösterildiği kısımdır.
Back-end: Kullanıcıdan alınan girdilerin HTTP istekleri(**request**) ile gönderildiği, bu isteklerer göre bu veriler ile çeşitli işlemler yapan ve çıktı üretip front-end'e HTTP mesajı (**response**) gönderen kısımdır.
Database: Kullanıcıdan alınan ve bu veriler uyarınca üretilen verilerin gerektiğinde depolandığı kısımdır.

HTTP Request-Response Cycle : HTTP protokolü ile haberleşen uygulama katmanları arasında kurulan yapıdır. Örneğin frontend ve backend katmanı arasında. Frontend'den backend'e yapılan bir http request'ine, server tarafından bir response üretilir ve frontend'e geri gönderilir. Frontend'den gönderilen request mesajının ve backend'den gönderilen response mesajının formatı değişebilir. Bu formata göre mimari adlandırılır. Bu format **JSON** ise **REST**, **XML** ile **SOAP** olarak adlandırılır.

Basit olarak anlatırsak, bir uygulamanın katmanları arasında veya farklı uygulamanın katmanları arasında haberleşmek için http protokolü kullanılabilir. Bu haberleşme HTTP metodları kullanılarak Request-Response cycle kurularak yapılır. Her Request'e bir Response dönülür.



HTTP Metodları : Uygulama katmanları (örneğin backend ve frontend) arasında HTTP protokolü kullanılarak haberleşme sağlanıyorsa, yapıacak işlemler bu protokolün farklı metodları ile yapılır.

GET: Server'dan veri istekleri **GET** metodu ile yapılır. Örneğin bir web uygulaması, kayıtlı tüm kullanıcılarına erişmek için server'a bir get request'i yollar. Server bu get request'ine tüm kayıtlı kullanıcıları reponse olarak gönderir.

POST: Server'a yapılacak veri yüklemeleri **POST** metodu ile yapılır. Örneğin bir web uygulamasına kayıt olmak istediğimizde, kullanıcı bilgilerimizi gireriz. **Kayıt ol** butonuna tıkladığımızda web uygulaması server'a bu kullanıcı bilgilerimizi içeren bir post request'i atar. Server bu request'i alır ve kullanıcıyı gerekli yere (database, dosya, cache v.b) kaydeder. Ardından bir kayıt başarılı **response'i** gönderir.

PUT: Server'da yapılacak veri güncellemeleri **PUT** metodu ile yapılır. Örneğin bir web uygulamasında kullanıcı adımızı güncellemek istersek, yeni kullanıcı adımızı gireriz ve güncelle butonuna tıklarız. Güncelle butonuna tıkladığımız anda web uygulaması server'a yeni kullanıcı adımız ile birlikte bir put isteği gönderir. Bu put isteğini alan server gerekli depolama alanında(db, dosya, cache) gerekli veri güncellemelerini yapar ve geriye bir veri güncelleme başarılı **response'i** gönderir.

DELETE: Server'da yapılacak veri silme işlemleri **DELETE** metodu ile yapılır. Örneğin bir web uygulamasında **sonra izleye** eklediğimiz bir pornoyu silmek istediğimizde sil butonuna tıklarız. Bu butona tıkladığımız anda web uygulaması server'a silmek istediğimiz porno ile ilgilli veriyi ile birlikte delete request'i yapar. Server bu request'e göre gerekli veri silme işlemini gerekli kayıt alanında yapar ver geriye kayıt silme başarılı **response'i** gönderir.

HTTP method	Resource URI	Description
GET	/users	gets a list of users
GET	/users/1234	gets a user identified by '1234'
POST	/users	creates a new user
PUT	/users/1234	updates a user identified by '1234'
DELETE	/users	deletes all users
DELETE	/users/1234	deletes a user identified by '1234'

3-) REST API ESPRİSİ

Uygulamanın katmanları (frontend ve backend) arasında iletişim kurmak için önemli bir protokolün HTTP olduğunu, uygulama katmanları arasında **HTTP Request ve Response cycle** kurulabileceğini ve çeşitli HTTP methodları kullanılarak bu haberleşmenin sağlanabileceğini biliyoruz. Bu request ve response mesajlarının bir formatı vardır. Örneğin sunucuya x kullanıcısının izlediği tüm pornoları gönder diye bir requesti GET metodu ile atabiliriz. Requestimiz içinde kullanılacak metodun **GET** olduğunu ve **kullanıcı id**'sini gönderdiğimizde server'da yer alan ilgili program bunu alır ve gerekli kayıt dosyasından bu kullanıcının izlediği tüm filmleri okur. Ve **response** olarak geriye bu filmlerin listesini göndermesi gerekir. İşte bu listeyi bu **JSON** olarak gönderirse elimizde canavar gibi bir rest api var diyebilir. JSON açılımı **JavaScript Object Notation** olan bir standarttır. Kısaca verileri bir JavaScript nesnesi olarak formatlar. **Türkçesi 2 tane süslü {} parantez arasında key ve karşılığında value olarak. Python'daki dictionary yapısı ile aynı şekilde.**

Örneğin ilto id’li kullanıcının izlediği 3 tane porno var. Biz bunları frontend’de listelemek için server’a **GET Request** attığımızda server database’e iltonun izlediği pornoları getir şeklinde bir query atar ve db ona muhtemelen ilto’nun izlediği pornoların adlarını içeren bir liste döndürür.

[“My father new helpless wife”, “2 black bitch and fish cips”, “Şahin K ile samanlıktan kaldıramadım”]

Server bu listeyi frontend’e şu şekilde gönderirse (**Response**)

{
“user” : “ilto”,
“fav-porn” : [“My father new helpless wife”, “2 black bitch and fish cips”, “Şahin K ile samanlıktan kaldıramadım”]
}

tebrikler GET çağrımıza cevap veren bir REST api’miz var. Vatana millete hayırlı uğurlu olsun. Bu frontend de bu verileri göstermek için rahatça kullanabiliriz.