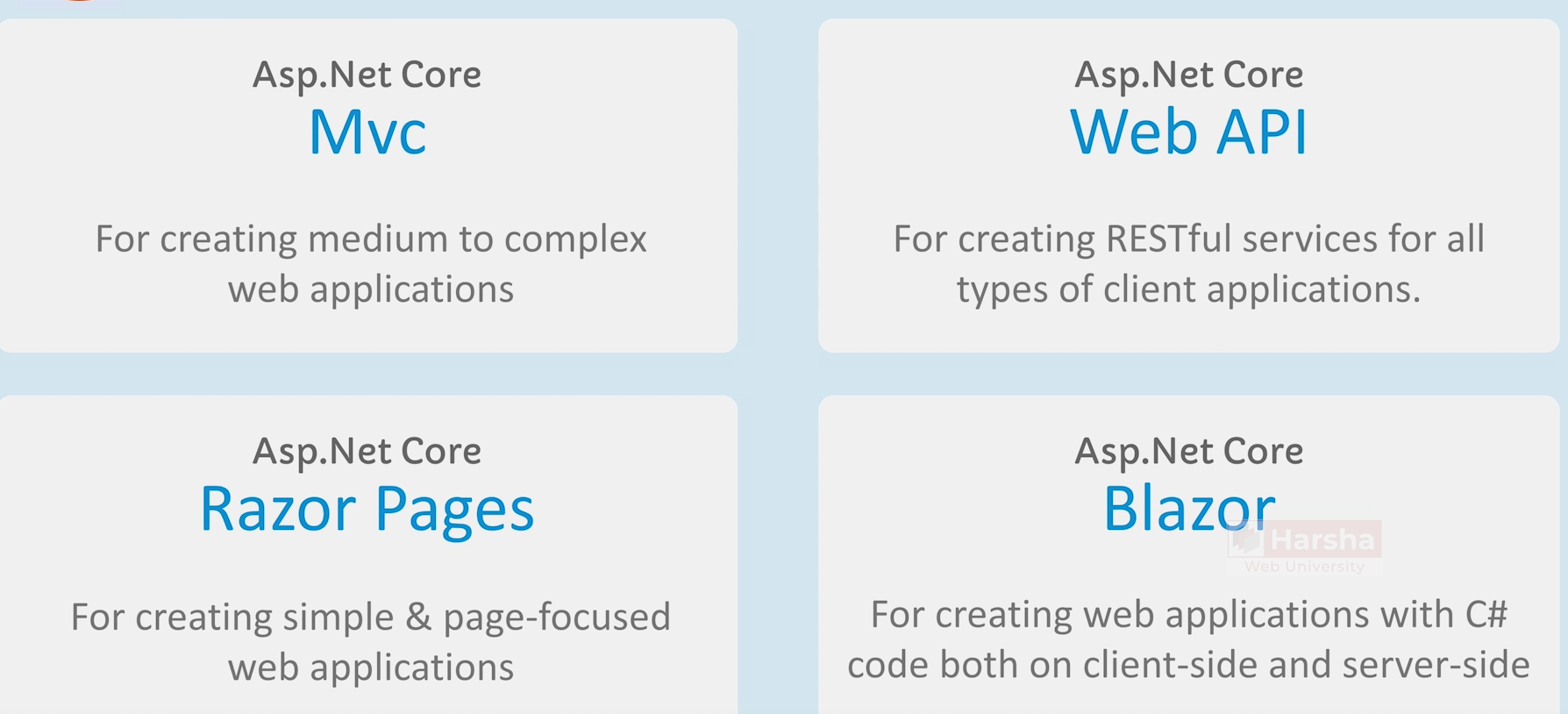
**Bölüm 1- Asp.Net Core’a Giriş**

Genellikle Devolopment için Mac ve Windows cihazlar tercih edilirken, sunucu tarafında Linux tercih edilir. Server tarafında requestleri alan ve responsları geri çeviren yazılımlar gerekir bunlar Kestrel,llS,Ngninx, docker gibi... Bu yazılımlara aynı zaman *“reserver proxies”* adı da verilir. Asp.Net Core varsayılan aplikasyon server’ı olarak Kestreli kullanır. Aynı zamanda Asp.Net Core cloud kısmında Azure’yi tamamen destekler. Asp.Net Core bünyesinde 4 temel yapıyı barındırır. MVC kısmında Model View ve Controller’ın hepsi Asp tabanlı yazılır. Web API kısmında Model ve Controller Asp ile, front end kısmı ise angular, react, xamarin vb frameworkler ile geliştirilebilir. Temelde restful servislerin yönetilmesinde kullanılır. Razor Page’ler basit ve sayfa odaklı web aplikasyonlarında daha çok tercih edilir. Hem sunucu hemde de client tarafında C# tercih edilecekse Asp.Net Core Blazor tercih edilir.



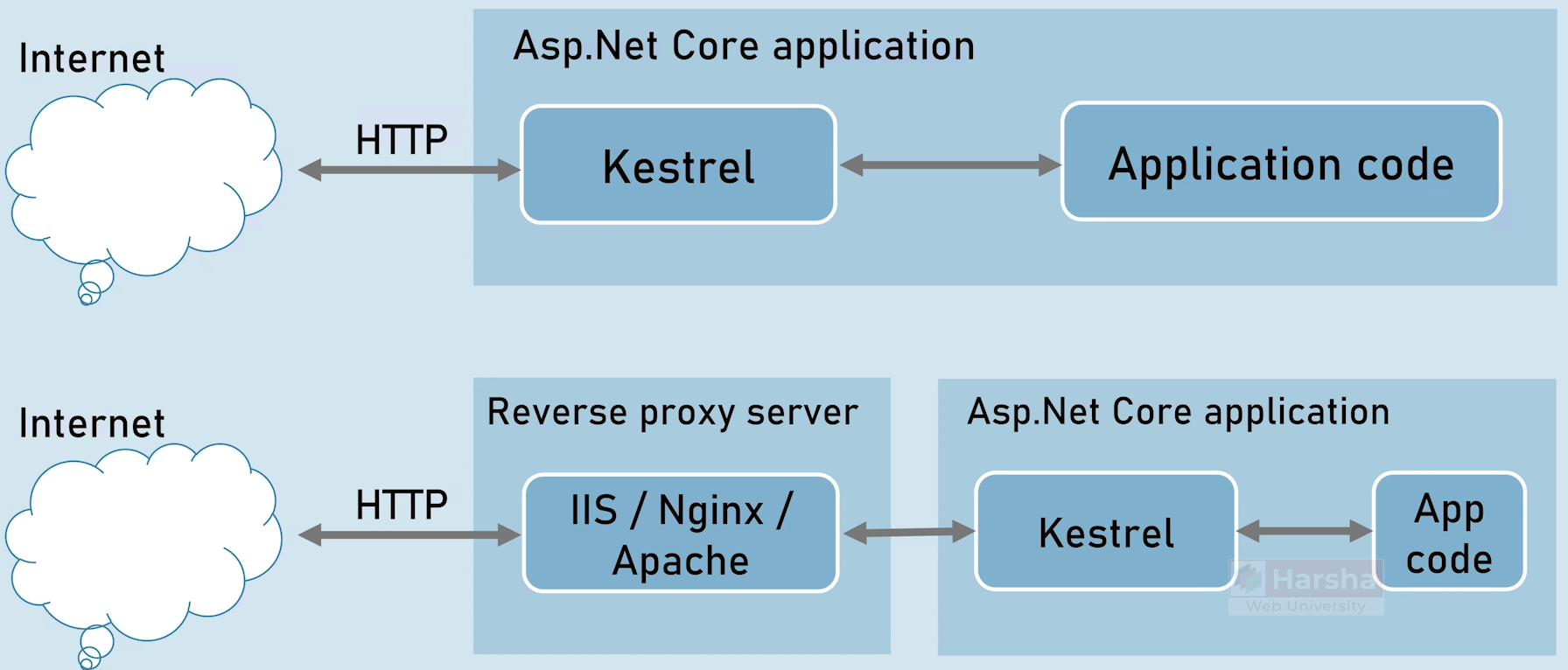
Şekil -Asp.NetCore Mimarisi

**Bölüm2-İlk Bakış**

Bir proje oluşturduğumuzda Program.cs içerisinde; Web Application Builder varsayılan olarak konfigürasyon enviroment ve varsayılan servisleri yükler (API,URL,Connection String vb). Builder konfigüre edildikten sonra app ile web builderlerarını entry pointleri yönlendirmek için kullanılabiliriz. Run() metodu ile serverı çalıştırırız kullanmazsak server çalışmaz.

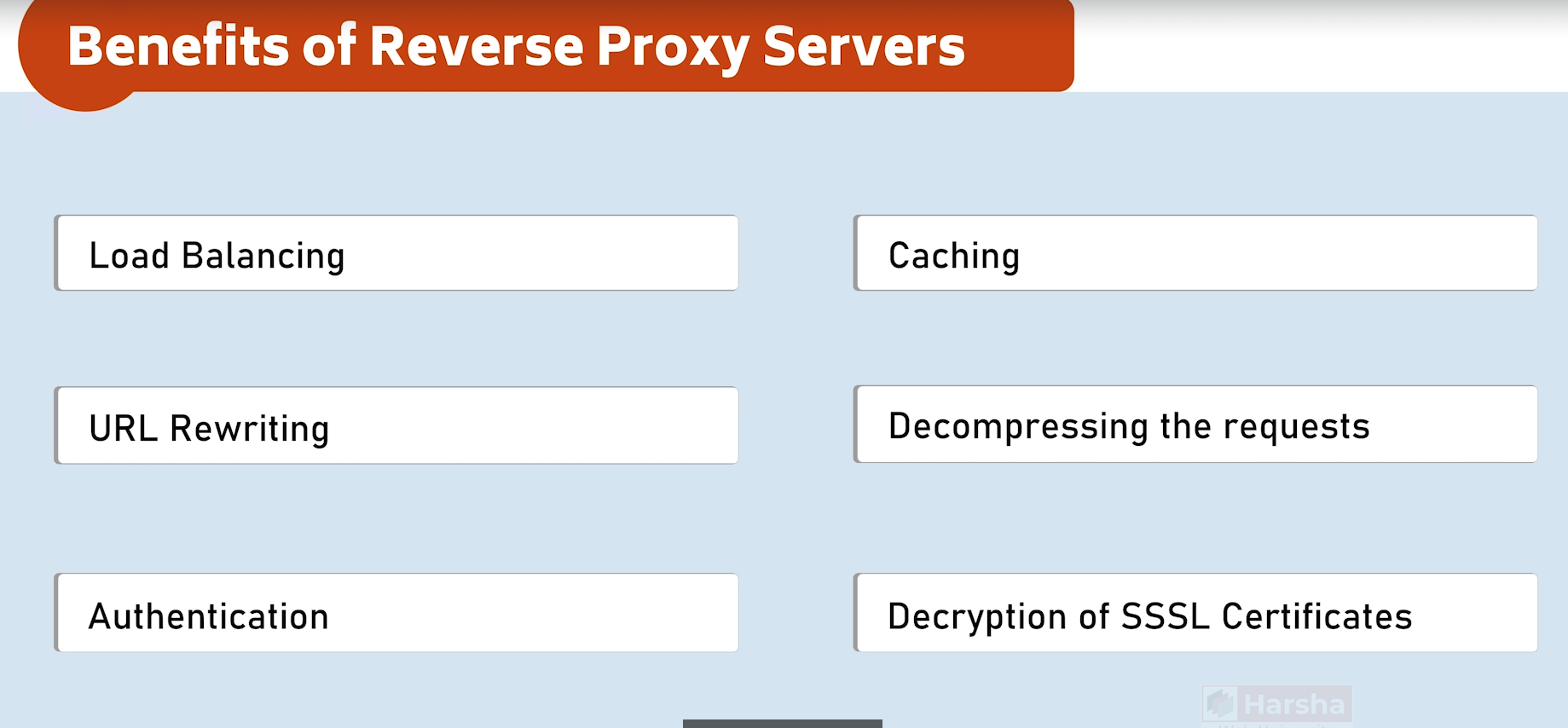
**2.1-Kestrel ve Diğer Servisler**

Kestrel varsayılan olarak bizim Application Server’ümüzdür. Geliştirme işlemlerinde sunucu rolünü oynar. Projemiz tamamlanıp gerçek dünya projesi olarak publish edilmek istendiğinde ise ***Reserve Proxy Server’***ları kullanmamız gerekir. Bunlardan bir popüler birkaçı IIS, Nginx, Apache gibi. Kestrel gerçek dünya projelerindeki ***load balancing****,* ***url rewriting,***  gibi özellikleri desteklemez. Aşağıda request olarak gönderilen HTTP’nin response kısmında Kestrel ve diğer reverse proxy server’ların nasıl çalıştıklarını görebiliz.



Şekil -Reverse Proxy vs Only Kestrel.

Reverse Proxy Sunucularının Sağladığı Birkaç İmkan:



Şekil -Reverse Proxy'nin Faydaları

**2.2-IIS Express**

ISS Express ISS proxy sunucusunu simüle eder. Requesti alıp Kestrel’a iletir. Sadece Windows ortamında çalışır.

**2.3-launchSettings.json**

Properties’in altında yer alan bi JSON (kısaca key-value çiftlerinin tutulduğu format) uzantılı launchSettings dosyası bulunur. IIS ve profiles için konfigürasyonu barındırır. Application URL ksımında localhost için 1024-65536 arasında port manuel olarak verilebilir. Bu 1024’ün altındaki portlar Windows işletim sistemi için reserved(ayrılmıştır) edilmiştir.

**2.4-Enviroment variables**

Kodun herhangi bir yerinden ulaşılabilecek global değişkenleri belirlediğimiz kısımdır. Varsayılan olarak Kestrel kullanırken IIS(Internet Information System) Express de launchSettings üzerinden değiştirilebilir. IIS’in Kestrel’dan avantajlı olduğu şeylerden birkaçı: http access logs, port sharing, windows authentication, process activation, configuration api gibi..

**Bölüm3-HTTP (Hyper Text Transfer Protocol)**

Tarayıcıya request ve Server’a response atılmasını sağlayan kurallar topluluğudur. İnternetteki tüm web siteleri ssl(Secure Sockets Layer) ile birlikte(kısaca https) veya tek başına http kullanır. Tüm tarayıcıların varsayılan protokolü http olduğu için araç URL kısmında yazılmasına gerek kalmaz çoğu zaman. Developer Tools’u tarayıcı da açmamız gerekebilir. Chorome için shorcut (ctrl+shift+i). Bu sekmede request ve response ile alakalı çoğu ayrıntıyı inceleyebiliriz.

**3.1-HTTP Status Codes**

HTTP Status Code’lar kestrel tarafından oluşturup browser(tarayıcıya) iletilir.

Birkaç örnek verecek olursak:

***1XX | İnformational***

Switching Protocols | 101 🡪 örneğin http’den https’e geçişi gösteririr.

***2XX | Success***

OK | 200 🡪 İşlem başarıyla tamamlanırsa OK olarak response döner.

***3XX | Redirection***

Found | 302 🡪 Found bir URL’den başka URL’e geçişi gösterir. Örneğin view-courses şeklinde bir URL girilirse /course/view URL’ine yönlendirme yapılılırsa 302 Status Code çalışır.

Not Modified | 304 🡪 Tarayıcının cache’inde zaten erişilebilir olan (image, html dosyası ya da css dosyaları gibi) dosyaları tekrar çağırmak yerine cache’deki dosyaları kullanır.

***4XX | Client Error***

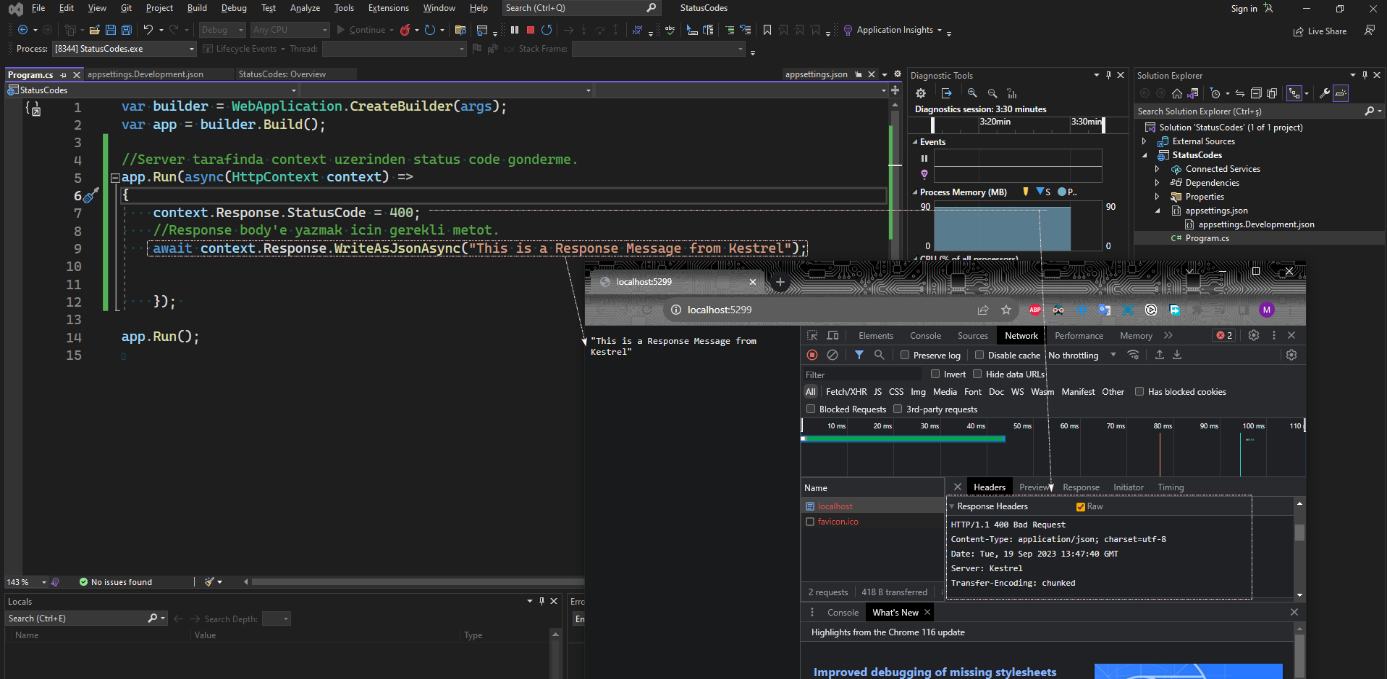
Bad Request | 400 🡪 Requestin parçası olan verilerde yanlış varsa ya da requestte verilmesi gereken parça verilmemişse bu Status Code döner. Örneğin id’si zorunlu olarak verilmesi gereken bir URL parçasında id verilmezse bad request hata kodunu alırız.

Unauthorized | 401 🡪 Yetkilendirme hatasıdır. Kullanıcı kimliği (e-mail,şifre veya otp) yanlış girilmişse bu hatayı alırız.

Not Found | 404 🡪 Uygulamamızın kaynak kodunda bulunmayan bir URL için yönlendirme yapmaya çalışırsak bu hata kodunu alırız.

***5XX | Server Error***

Internal Server Error | 501 🡪 Server taraflı çalışma zamanlı istisnalar gibi hatalarda bu hata kodunu alırız.

Belirtilen Status Code’lar en önemli ve en yaygın alınabilecek hata kodlarıdır. Kodumuzda bu hata kodlarını göstermek istiyorsan Program.cs içerisindeki app.Run() metodunu konfigüre ederecek çağırabiliriz. Parametre olarak requesti kabul ettikten sonra otomatik olarak oluşturulan **HttpContext** tipindeki **context**’i barındırır. Bu context üzerinden response ve request’e ait metodlar çağrılabilir. Run, Map, ve Use metodu arasındaki farklar ***Middleware*** kısmında detaylıca incelenecek. Await anahtar kelimesi ile belirtilen kısımdaki tüm çalışmaların tamamlandıktan sonra alt satırlarda işleme devam edeceğini belirtir.

Şekil -Status Code Gönderme

Şekil -Kestrel Üzerinden Response Gönderme

Not: Eğer kod içerisinde asenkron olarak çağrılan bir metot, değişken ya da obje varsa kullanıldığı çağrıldığı yerde ***async*** komutuyla belirtilmeli ve çalıştırılırken ***await*** anahtar kelimesi kullanılmaldır.

***3.2-HTTP Response Headers***