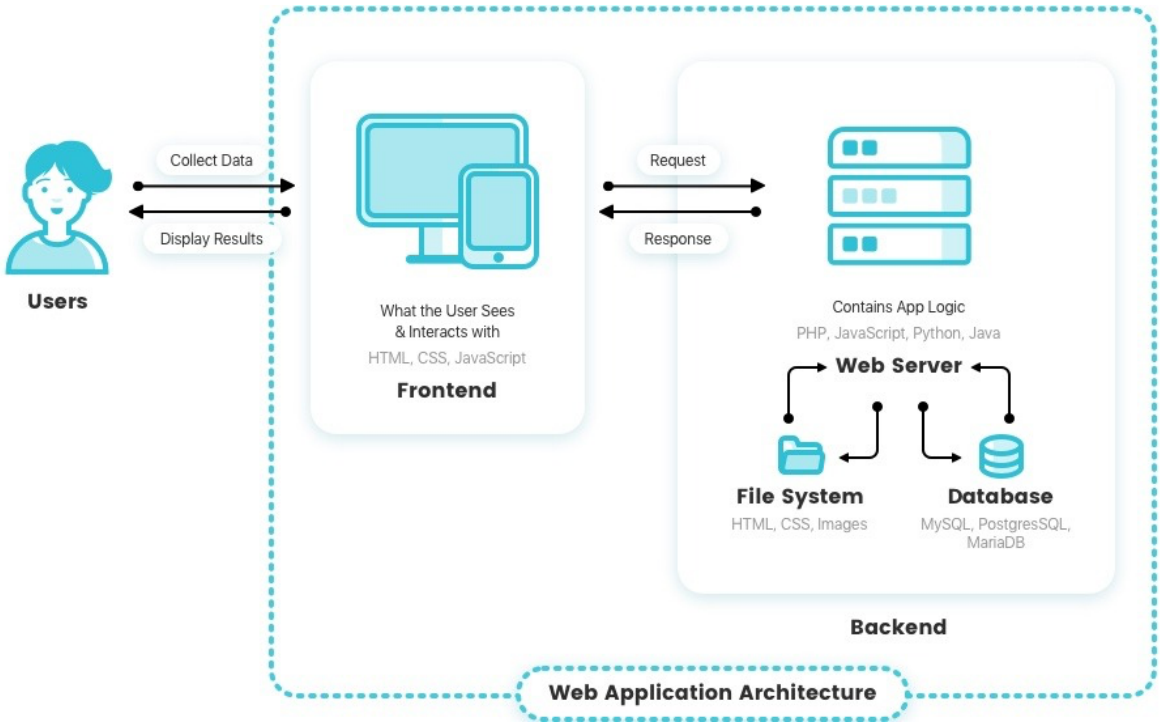
Bölüm-7

Web Uygulama Mimarisi nedir?

Web uygulaması birbiriyle etkileşim halinde olan birçok bileşenden meydana gelmiş bütün bir çözümdür. Tüm alt bileşenler belli görevleri yerine getirerek bir bütünü oluştururlar. Aslında, insan bedenine benzetebiliriz. Her organın bir fonksiyonu vardır. Bu organlar birbiriyle çeşitli yollarla haberleşmektedirler. Aynı şekilde bir web uygulama mimarisi de alt bileşenlerden meydana gelir. Örneğin, amazon.com, netflix.com gibi web uygulamaları arka planında karışık bir etkileşim içindedirler. Web uygulamasının kullanıcı tarafından görülen önyüzü, gelen istekleri karşılayan sunucu yazılımları, mobil uygulamalar ve veri tabanları gibi epey bileşen birbiriyle belli bir mantık içinde belli kurallar çerçevesinde haberleşirler. Bu etkileşim ağını kurulması ve yönetilmesi yazılım mühendislerinin, sistem mühendislerinin ve veri tabanı yöneticilerinin kolektif şekilde çalışmasıyla mümkün olur. Zaten bu yapı da web uygulama mimarisinin kendisini ortaya çıkarır. Bileşenler ve onların ne şekilde etkileşim içinde olacağı mimariyi belirgin hale getirir.

Bunu basit bir şemayla izah edelim.



Yukarıdaki temel ve basit şemada bir web uygulaması temelde önyüz (Frontend) ve arka plan (Backend) olmak üzere iki önemli dünyadan meydana gelmektedir. Bu iki ana parça bir web uygulamasını oluşturur.

Gördüğümüz gibi kullanıcı bilgisayarındaki tarayıcıda gördüğü ara yüzler, ekranlar onun etkileşime geçtiği katmandır. Bu katmanda daha çok Frontend tarafını temsil eder. Bu kısma istemci tarafı da (client-side) denilmektedir. Bu katmanda kullanıcı bilgisayarındaki tarayıcı yazılımı sayesinde web uygulaması ile etkileşimde bulunup Backend tarafına isteklerde

bulunur. Backend'den aldığı cevapları tarayıcı tarafında işletilir. HTML, CSS ve Javascript bu katmanın en önemli teknolojileridir. Frontend geliştirici de ağırlıklı olarak bu teknojiler ile geliştirmeler yaparlar. Frontend tarafında geliştirilen kodlar ve yazılım modülleri kullanıcının bilgisayarındaki tarayıcıda çalıştırılır.

Backend tarafı aynı zamanda server-side (sunucu tarafı) diye de bilinir. Bu katmandaki yazılımlar sunucu bilgisayarlarda çalışan ve bilgisayar ağları üzerinden gelen isteklere, özellikle de HTTP protokolünden gelen isteklere cevap verebilen yazılım katmanlarıdır. Bu katmana dış kullanıcıların doğrudan erişimi yoktur. Web uygulamalarında yer alan ekranlardan veya ara yüzlerden sunucu tarafındaki bu yazılım katmanları tetiklenir. Backend yazılımları özellikle HTTP isteklerini alıp işlerler. Bu HTTP istekleri bir REST servisi çağrısı, bir dosyanın, bir HTML sayfanın veya fotoğrafın sunucudan talep edilmesi şeklinde olabilir. Aynı şekilde HTTP istekleri ile Backend yazılımları veri tabanlarında veya kalıcı diskte yeni kaynaklar yeni kayıtlar oluşturabilirler. Bu katmanda yazılım geliştirmek için PHP, Python, Java, C# ve Javascript gibi birçok alternatif teknoloji bulunmaktadır. Backend tarafında yazılan kodlar ve yazılım modülleri sunucu dediğimiz güçlü bilgisayarlarda çalıştırılır. Bu kodlar direkt olarak kullanıcı bilgisayarında çalıştırılamazlar.

Örneğin bir alışveriş sitesinde üye ol ekranına yönlendirildiğinizde tarayıcıda (örn Google Chrome) karşınıza bir web ekranı gelir. Bu ekrandan üyelik bilgilerini doldurup, email adresinizi yazarak bir hesap oluşturursunuz. Ardından, sizi profil sayfanıza yönlendirir.

Bu basit akışta gördüğünüz üyelik ekranı Frontend tarafı temsil etmektedir. Bu ekran üzerinden girdiğiniz üyelikle ilgili veriler Frontend'de toplanır ve Backend'e iletilmek üzere HTTP istekleri şeklinde sunucu bilgisayarlara gönderilir. Sunucu bilgisayara ulaşan bu istekler Backend yazılımları tarafından ele alınarak bir orta katmanda işlenir ve kalıcı olarak veri tabanına kaydedilir. İşte isteklerin sunucuya ulaştıktan sonraki kısım ise Backend dünyasını ilgilendirir.

Web Uygulama Mimarileri

Web üzerine geliştirme yapmaya kalktığımızda yazılım geliştirme açısından aslında en temelde iki tip mimari yaklaşım ortaya çıkar.

- Sunucu Taraflı İşlem (Server-side Rendering)
- İstemci Taraflı İşlem (Client-side Rendering)

Sunucu Taraflı İşlem (Server-side Rendering)

Bu tip mimarilerde bir sayfa üzerinden bir HTTP isteği yaptığınızda, örneğin formun kaydet tuşuna tıkladığınızda, sunucu tarafına bir HTTP isteği gider. Ardından, Backend tarafında çeşitli kodlar icra edilir. Örneğin Java kodları çalışır ve müşterinin geçmiş ödeme listesini veri tabanından çeker. Veri tabanından gelen bilgilerle bir HTML sayfası Backend katmanında oluşturulur. HTML halindeki sayfa bütünüyle istemciye cevap olarak gönderilir. Böylece, istemcinin tarayıcı vasıtasıyla yaptığı HTTP isteğine sayfanın tümünü göndererek yanıt veren

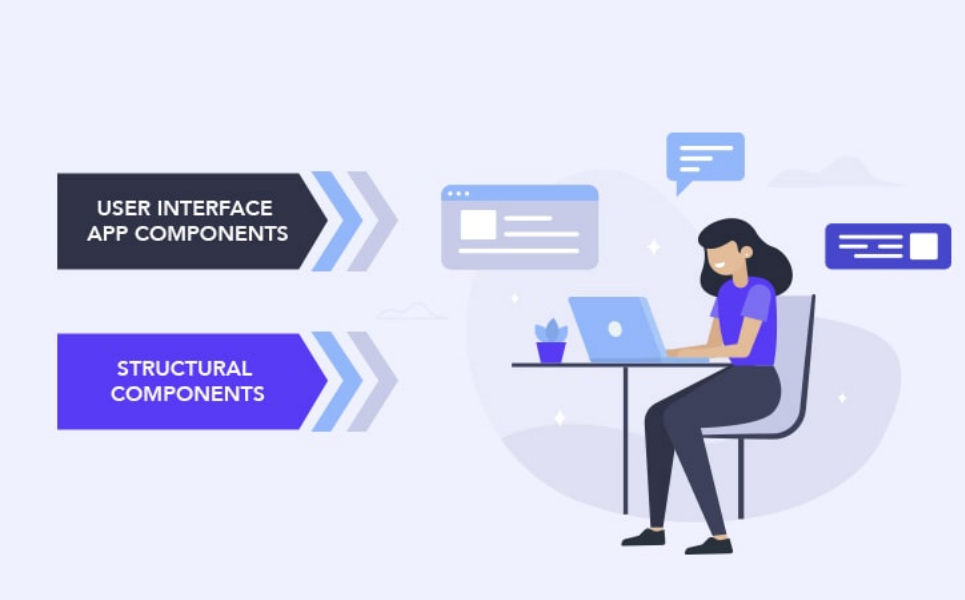
teknolojilere server-side rendering sistemleri diyebiliriz. Örneğin Java dünyasında Java Server Page teknolojisi bu türdendir.

İstemci Taraflı İşlem (Client-side Rendering)

İstemci taraflı uygulamalarda aslında Backend yazılımlarla haberleşir. Bu kavramdan mimariden Backend katmanlarını tamamen çıkardığımızdan bahsedemeyiz. Yine istemci ve sunucu arasında bir etkileşim vardır. Fakat, bu mimaride örneğin alışveriş sepetini görmek için bir butona tıkladığınızda Frontend tarafından sunucuya bir HTTP isteği gönderilir. Fakat, bu istek sadece alışveriş sepetindeki ürünler listesini verecektir. Sayfanın tamamı Backend katmanında hazırlanıp bir HTML sayfası olarak dönmeyecektir. İstemci tarafı yani tarayıcımız sunucudan gelen bu minik veri parçasını alıp sayfanın sadece ilgili bölümünü güncelleyecektir. Sayfada bulunan diğer bölümler ve alanlarda bir değişim olmayacaktır. Gelen bu veri parçası Javascript dili kullanılarak Frontend tarafında bir iş mantığında ekrana yansıtılacaktır.

Web Uygulama Mimarisi Bileşenleri

Kullanıcı tarafından kullanılan ara yüz bileşenleri ve yapısal bileşenler olmak üzere ikiye ayrılır.



Kullanıcının nasıl bir ekranı göreceğini, tasarım, renk ve deneyim olarak tasarladığımız bileşenlerdir. Örneğin ürün arama sayfasına gelen bir kullanıcı arama çubuğuna aramak istediği ürünü yazar. Yazdıkça ürünler liste halinde altta sıralanır. İndirimde olan ürünlere dair sağ üstte minik bir bilgilendirme kutusu çıkar. Bu senaryoda ön yüzde olması gerekenleri tasarlıyoruz.

Yapısal bileşenler ise daha çok ekran üzerinde kullanılabilecek fonksiyonlarla ile ilgilidir. Kullanıcının siparişi oluştur tuşuna bastığında isteğin sunucuya ulaşması ve belli bir iş

mantığından geçtikten sonra sipariş ile ilgili bilgilerin veri tabanına kaydedilmesi gibi bir sürece ait bileşenler yapısal bileşenlerdir.

Yapısal bileşenler kabaca 3 başlıkta toplanabilir.

- Web tarayıcısı veya bir istemci örneğin bir mobil uygulama
- Web uygulama sunucusu, geliştirilen web uygulamalarının çalıştırılmasından sorumlu yazılım
- Veri tabanı sunucuları

Tarayıcılar kullanıcının etkileşim halinde olduğu alanlardı. Tarayıcılarda web uygulamamıza ait web sayfalarını gösteririz. HTML, CSS ve Javascript gibi teknolojiler bu bileşende adreslenebilir.

Web uygulama sunucuları ise hazırlanan web projelerini çalıştıran sunucu yazılımlarıdır. Genellikle bu katmanda çalışacak olan web uygulaması birden fazla katmana bölünür. Örneğin, iş mantığını Servis Katmanı'nda, veriye erişim ise DAO Katmanı'nda kurgulanabilir. Java, C#, PHP, Python gibi programlama dilleri ve ona bağlı teknolojiler ile bir web uygulaması geliştirilir. Bu uygulama dediğimiz gibi bir uygulama sunucusunda çalıştırılır. Örneğin bahsettiğimiz uygulamayı Java'da geliştirirsek Tomcat Web Uygulama Sunucusu'nu bu projeyi çalıştırması için kullanabiliriz. Tomcat, Java'da hazırlanmış web uygulamalarını çalıştırıp, istemciden gelecek istekleri web uygulamasına yönlendirir.

Veri tabanı katmanında ilişkisel veri tabanları, NoSQL veri tabanları gibi bir çok sistem bulunmaktadır. Veriyi sakladığımız bileşendir.

Web Uygulaması

Web Uygulaması = Frontend + Backend

Biliyoruz ki server-side rendering içeren teknolojilerde sayfada örneğin kaydet tuşuna tıklandığında sayfa tümüyle sunucu tarafında işlenip yeniden bütün bir HTML sayfa halinde istemciye yani web tarayıcısına gönderilirdi. Bu işleme aslında Postback diyebiliriz. Postback ne kadar çok artarsa aslında sunucu tarafında da yoğunluk o miktarda artar. Bu sunucunun performansının düşmesine neden olabilir. Bu nedenle de web uygulamasında yer bazı fonksiyonlar veya iş mantığı Frontend ile birlikte kullanıcının tarayıcısında çalışacak şekilde kodlanabilir. Böylece sunucu üzerindeki yükü azaltırız. Müşterinin geçmiş siparişlerini sayfada görebilmek için sayfanın tümünü render etmek yerine sadece geçmiş sipariş kayıtlarını liste olarak sunucudan döndürmek daha az maliyetli bir operasyon olacaktır.

Ajax (Asynchronous Javascript and XML)

Bir web sayfasının sunucu tarafındaki bir veriye ihtiyacı olduğunda dediğimiz gibi sayfanın tümünü sunucudan almak yerine sadece ilgilendiğimiz veri parçasını almak daha uygun bir çözümdür. İşte istemci tarafının yani kullanıcının tarayıcısından sunucudaki bir veriyi HTTP

protokolü vasıtasıyla alabilmesini ve haberleşmesini sağlayan alt yapıya Ajax denilmektedir. Ajax ile Javascript dilini kullanarak sunucuda ihtiyaç duyduğumuz veriyi istemci tarafına transfer edebiliriz. Böylece, aldığımız veri ile birlikte sayfanın sadece ilgili bölümünü güncellemiş oluruz.

Ajax ile sunucuda çalışan web servislerinden ilgilendiğiniz veriyi talep edebilirsiniz. Javascript kullanarak Ajax ile yapılan HTTP isteğine sunucudaki web servis veri tabanına erişip ilgili kayıtları bir araya getirerek bir cevap halinde istemci tarafına dönecektir.

Web Sitesi kavramı ve HTTP Protokolü

Web sitesi aslında birçok web sayfasının, statik kaynakların (fotoğraf, dosya, video vb.) ve sunucu taraflı yazılımların bir araya gelmesinden oluşan bir uygulamadır. Bir web sitesinin erişim adresi olmalıdır. Bu adres www.google.com gibi bir isimlendirme ile temsil edilir. “google.com” web sitesi için domain’i ifade eder. Domain’ler akılda kalıcı ve herkesin daha rahat kullanabildiği adreslerdir. Web sitesinin erişimi için bir adrestir. Domain arka planda bir IP adresi ile ilişkilendirilir.

Statik Web Siteleri

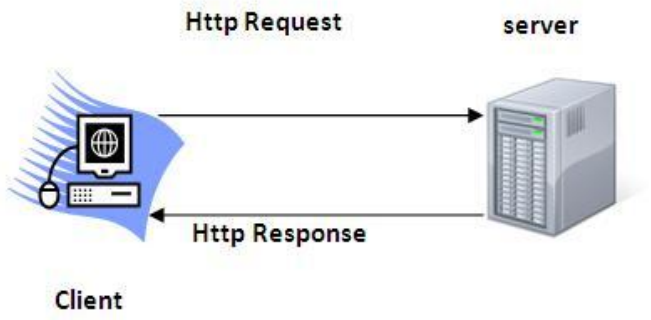
Sabit web sayfalarından oluşurlar. Herhangi bir veri tabanı etkileşimi içermezler. Sayfalar tasarlanır ve kullanıcı etkileşimi olamayacak şekilde servis edilir.

Dinamik Web Siteleri

Günümüzde hemen hemen her yerde karşımıza çıkan web sitesi tipleridir. Veri tabanı etkileşimi ile birlikte sunucu taraflı birçok iş süreci işletilir. Web sitesi üzerinden kullanıcı etkileşimli bir şekilde web sayfalarını kullanabilir. Örneğin sepetine ürünleri ekleyip sipariş oluşturabilir.

HTTP Protokolü

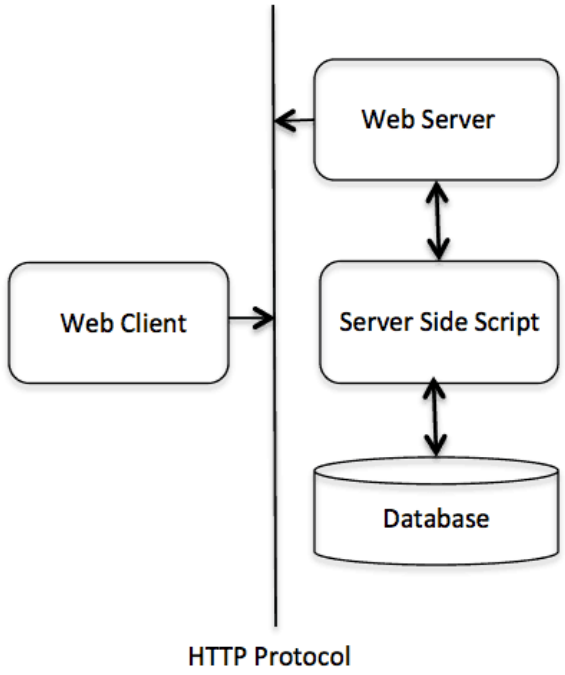
HTTP protokolü uygulama katmanında yer alan bilgisayar ağıdır. HTTP üzerinden istemci ve sunucular haberleşebilirler. Bilgisayarlar birbiriyle bu protokol üzerinden haberleşebilirler. İnternet ortamında yazı, ses, video, fotoğraf gibi kaynakların paylaşılmasını ve bilgisayarlar arası transferini sağlamak için ortaya çıkmıştır. Alt katmanında TCP protokolü yer almaktadır. HTTP protokolden gelen istekler TCP protokolü ile iletilir. HTTP protokolünün portu varsayılan olarak 80’dir.



HTTP Protokolü Özellikleri

- Web sunucuları ile kullanıcıların bilgisayarlarında yüklü olan tarayıcılar arasında veri alışverişini sağlar.
- İstek (Request) ve Cevaba (Response) dayalı bir protokoldür.
- Alt katmanında TCP kullanır. TCP katmanında varsayılan portu 80'dir.
- Protokol yapılan isteğe dair herhangi bir durum bilgisi saklamaz. Her yapılan istek sunucuya iletilir ve cevabı beklenir. Ardından bu kurulan iletişimle ilgili kalıcı olarak herhangi bir bilgi saklamaz. İletişim bittikten sonra sunucu ve istemci birbirini unutur.
- Herhangi bir kaynak HTTP üzerinden paylaşılabilir. Örneğin video, ses, fotoğraf, yazı metni, web sayfaları gibi kaynaklar sunucudan istemciye HTTP üzerinden aktarılır.
- HTTP protokolü üzerinden sunucuya istek aktarıldıktan sonra sunucu ile bağlantıyı sonlandırır ve sunucudan gelecek cevabı beklemeye başlar.

HTTP Protokol Şema

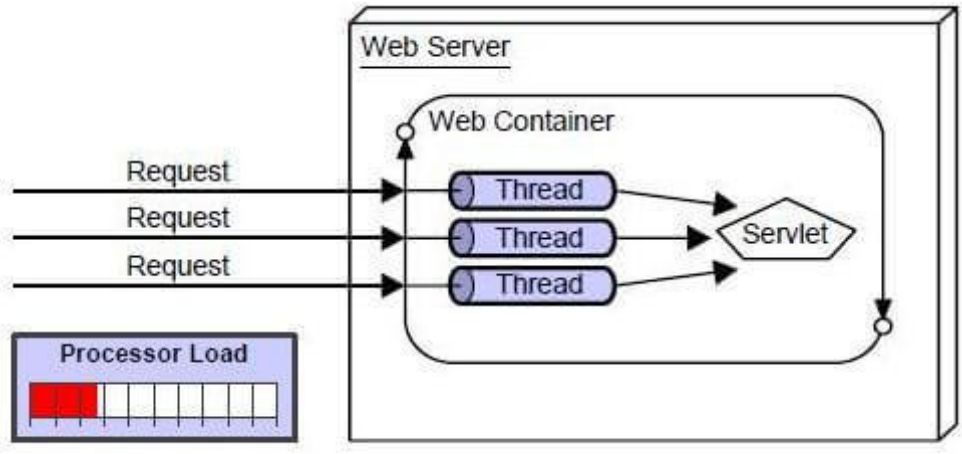


Java Web Uygulamalarının Kalbi => Servlet

Java ile web uygulamaları geliştirmek isteyenler mutlaka Servlet kavramını ve onun yaşam döngüsünü iyi bilmelidirler. Java Web altyapılarının hemen hemen hepsi Servlet'lere dayanır. Bazen biz detaylarına vakıf olmasakta Java Web teknolojileri arka planda Servlet teknolojisini kullanırlar. Örneğin Java Server Page ve Java Server Face gibi teknolojiler Servlet'e dayalıdır.

Servlet, Java ile dinamik web uygulamaları yapabilmenize olanak tanıyan bir alt yapıdır. Web uygulama sunucusu (örn: Tomcat) tarafından çalıştırılan Java kodlarının HTTP üzerinden gelen isteklere cevap verebilme yeteneği kazanmasını sağlar.

Servlet tanımladığında bir yaşam döngüsü oluşur. Oluşturulma, işletilme ve sonlandırılma olmak üzere 3 ana süreci vardır. İleri notlarda bunu biraz daha açacağız.



Yukarıda Servlet'e gelen HTTP istekleri şematize edilmiştir. Görüldüğü gibi web sunucusu gelen HTTP isteklerini Thread'lere ayırarak her isteği Servlet'e iletmektedir. Buradaki web sunucusu Tomcat olarak düşünebiliriz.

Java Servlet Tanımlamak

Java'da Servlet tanımlamak için öncelikle bir Maven projesi açıyoruz. Maven projesini "war" olarak seçiyoruz.

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
</dependency>

<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.0</version>
        <scope>provided</scope>
</dependency>
```

Yukarıdaki bağımlılıkları projeye ekliyoruz. Böylece, Java projesinde Servlet sınıfları oluşturabiliyoruz.

Bir Java sınıfının HTTP Servlet özelliği kazanabilmesi için “HttpServlet” sınıfından kalıtım alması gerekmektedir. Aşağıda tanımladığımız Servlet sınıfı HTTP üzerinden gelebilecek GET, POST, DELETE ve PUT tipindeki istekleri karşılayabilecek fonksiyonları açıyoruz. Böylece, hazırladığımız Servlet’i HTTP POST yöntemiyle çağrılırsa ilgili fonksiyona düşecektir. O fonksiyon içindeki kodları işlemeye başlayacaktır. Eğer, HTTP DELETE tipinde bir istek olsaydı, bu sefer de “doDelete” fonksiyonuna düşecekti.

```
public class EmptyServlet extends HttpServlet {

	private static final long serialVersionUID = 4253740468216504555L;


	@Override
	protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
		super.doGet(req, resp);
	}

	@Override
	protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
		super.doPost(req, resp);
	}

	@Override
	protected void doPut(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
		super.doPut(req, resp);
	}

	@Override
	protected void doDelete(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
		super.doDelete(req, resp);
	}
}
```

Tanımladığımız Servlet’i “WEB-INF” klasörü altındaki “web.xml” içinde tanımlamak gerekecektir. Böylece, web uygulaması Tomcat üzerinde çalıştırılmaya başlandığında, hazırladığımız web uygulaması “web.xml” içindeki tanımları okuyup bunlardan Servlet nesneleri yaratacaktır.

```
<servlet>
	<servlet-name>EmptyServlet</servlet-name>
	<servlet-class>com.servlets.basics.EmptyServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
        <servlet-name>EmptyServlet</servlet-name>
        <url-pattern>/empty</url-pattern>
</servlet-mapping>
```

Böylece artık web uygulamamızı Tomcat üzerinde çalıştırıp yayına alabiliriz. Aşağıdaki şekilde çağırabiliriz.

http://localhost:8090/chapter7-servlet-basics/empty

Protokol ismi + Domain Adı + Port Numarası + Uygulama İsmi + Servlet Url Pattern (web.xml içinde tanımlamıştık.)

@WebServlet Etiketi ile Servlet Tanımlamak

Etiket tabanlı yöntemi kullanarak Servlet'leri tanımlayabiliriz. Böylece, Servlet ile ilgili tanımlamaları "web.xml" dosyasına eklememize gerek kalmaz. Şimdi de etiket tabanlı yöntemle nasıl bu işi yapabiliriz onu inceleyelim.

```
@WebServlet(name = "accountManagerServlet", urlPatterns = "/user/account")
public class AccountManagerServlet extends HttpServlet {

        private static final long serialVersionUID = 6149205809868611564L;


        // Java codes
}
```

@WebServlet etiketini sınıf üst kısmında tanımlıyoruz. Ardından "name" ile Servlet'e tekil bir isim veriyoruz. "urlPatterns" ile tanımladığımız Servlet'e hangi adres üzerinden erişilebileceğini belirtiyoruz.

Servlet Yaşam Döngüsü

Servlet oluşturulurken ve kullanımdayken bir yaşam döngüsü vardır. Yaşam döngüsü için "init()", "service()" ve "destroy()" olmak üzere 3 önemli fonksiyonu vardır. Örneğin Tomcat üzerinde web projeniz çalışmaya başladığında web projesi içinde yer alan Servlet'ler oluşturulmaya başladığında "init" metodu sadece bir kez işletilir.

```
@Override
public void init() throws ServletException {

        userIdCounter = 0;
        users = Collections.synchronizedList(new ArrayList<User>()); }
```

Yukarıdaki adımda Servlet oluşturulduktan sonra Servlet artık HTTP isteklerini karşılayabilecek durumdadır. “service” fonksiyonunu veya “doGet”, “doPost”, “doPut”, “doDelete” fonksiyonlarını kullanarak hizmet verebilir.

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

}

@Override
protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

        System.out.println(req.getMethod() + " is trigged!");
}
```

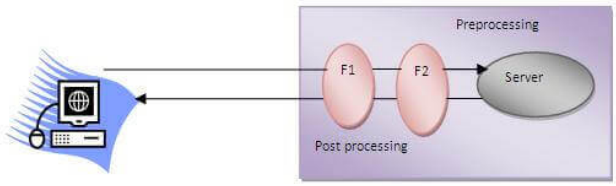
Oluşturduğumuz Servlet nesnesi artık kullanılmayacak olduğunda Tomcat Servlet’in “destroy” fonksiyonunu çağıracaktır. Bu fonksiyon içinde Servlet’in kullandığı veri tabanı bağlantıları, açık dosyalar veya bilgisayar ağları ile ilgili açık bağlantılar varsa bunlar kapatılabilir. Kullanılmayacak nesneler destroy edilebilir.

```
@Override
public void destroy() {

        users.clear();
}
```

Servlet Filter

Servlet’lere istekler iletilmeden önce veya Servlet işlemini bitirdikten sonra araya Filter sınıfları koyarak gelen HTTP istekleri üzerinde çeşitli kontroller yapabiliriz. Böylece, gelen ve giden HTTP isteklerini kontrol edip üzerinde çeşitli işlemler yapabiliriz.



Yapacağımız örnekte önceden tanımladığımız Servlet'lere erişim yetkisi olmayanları kısıtlamak istediğimizi düşünelim. Böyle bir durumda Servlet'lere HTTP istekleri varmadan araya girip gelen istekleri validasyona sokabiliriz. Böylece, erişim için yetkisi olup olmadığını anlayabiliriz. Eğer erişim yetkisi yoksa HTTP 401 statusunda bilgilendirme ile uyarı veririz. Eğer geçerli bir istek ise Servlet'lere erişmesine olanak tanırız.

```
@WebFilter(
                filterName = "authenticationFilter",
                urlPatterns = { "/user/account", "/empty", "/greeting" } )
public class AuthenticationFilter implements Filter {

        private List<String> tokens;

        @Override
        public void init(FilterConfig filterConfig) throws ServletException
{
                tokens = loadValidTokens();
        }

        @Override
        public void doFilter(ServletRequest request, ServletResponse
response, FilterChain chain)
                        throws IOException, ServletException {

                HttpServletRequest httpRequest =
(HttpServletRequest)request;
                HttpServletResponse httpResponse =
(HttpServletResponse)response;


                String httpMethod = httpRequest.getMethod();

                if("GET".contentEquals(httpMethod) &&
"/user/account".contentEquals(httpRequest.getServletPath())) {
                        System.out.println("Authorization is ok!");
                        chain.doFilter(request, response);
                }
                else {
                        String apiKey = httpRequest.getHeader("x-api-key");

                        if(apiKey == null) {
                                apiKey = httpRequest.getParameter("x-api-
key");
                        }

                        if(apiKey == null) {
                                unAuthorizedHttpRequest(httpResponse);
                                return;
                        }
```

```
                        else if(!tokens.contains(apiKey)) {
                                unAuthorizedHttpRequest(httpResponse);
                                return;
                        }

                        System.out.println("Authorization is ok!");
                        chain.doFilter(request, response);
                }

        }

        @Override
        public void destroy() {
                tokens.clear();
        }

        private void unAuthorizedHttpRequest(HttpServletResponse
httpResponse) throws IOException {

                httpResponse.setStatus(401);
                httpResponse.setContentType("text/html;charset=UTF-8");
                httpResponse.getWriter().println("<html>");
                httpResponse.getWriter().println("<head>");
                httpResponse.getWriter().println("Kaynağa erişim için
geçerli bir yetkiniz yoktur. Lütfen geçerli bir x-api-key giriniz!");
                httpResponse.getWriter().println("</head>");
                httpResponse.getWriter().println("</html>");
        }

        private List<String> loadValidTokens() {

                List<String> tokens = new ArrayList<String>();
                tokens.add("t1");
                tokens.add("t2");
                tokens.add("t3");
                tokens.add("t4");
                tokens.add("t5");

                return tokens;
        }

}
```

Java sınıfı “Filter” interface’den kalıtım alarak gerekli fonksiyonları override eder. Ardından, Filter sınıfını dilersek web.xml içinde de tanımlayabiliriz. “web.xml” dosyasına tanım yapmak istemeyenler @WebFilter etiketini kullanabilirler. Örneğimizde etiket tabanlı bir tanım yapıldı. “filterName” ile filtrenin ismini belirtiyoruz. “urlPatterns” ile filtre belirtilen adreslere HTTP isteği geldiğinde otomatik olarak devreye girecektir.

“”AuthenticationFilter” isimli filtremizde HTTP istekleri Servlet’lere gitmeden önce bir “authorization” kontrolünden geçiyorlar. Eğerki, gelen HTTP isteklerinde geçerli bir “x-api-key” yok ise kullanıcıya uygun bir mesaj verilecektir. Fakat, geçerli bir token’a sahipse Servlet’lere erişebilecektir.

Eğer filtreyi “web.xml” içinde tanımlamak isteseydik. Aşağıdaki gibi bir ekleme yapmamız gerekecekti.

```
<filter>
        <filter-name>authenticationFilter</filter-name>
        <filter-
class>com.servlets.basics.filter.AuthenticationFilter</filter-class>
</filter>

<filter-mapping>
        <filter-name>authenticationFilter</filter-name>
        <url-pattern>/user/account</url-pattern>
        <url-pattern>/greeting</url-pattern>
        <url-pattern>/empty</url-pattern>
</filter-mapping>
```

Filter Yaşam Döngüsü

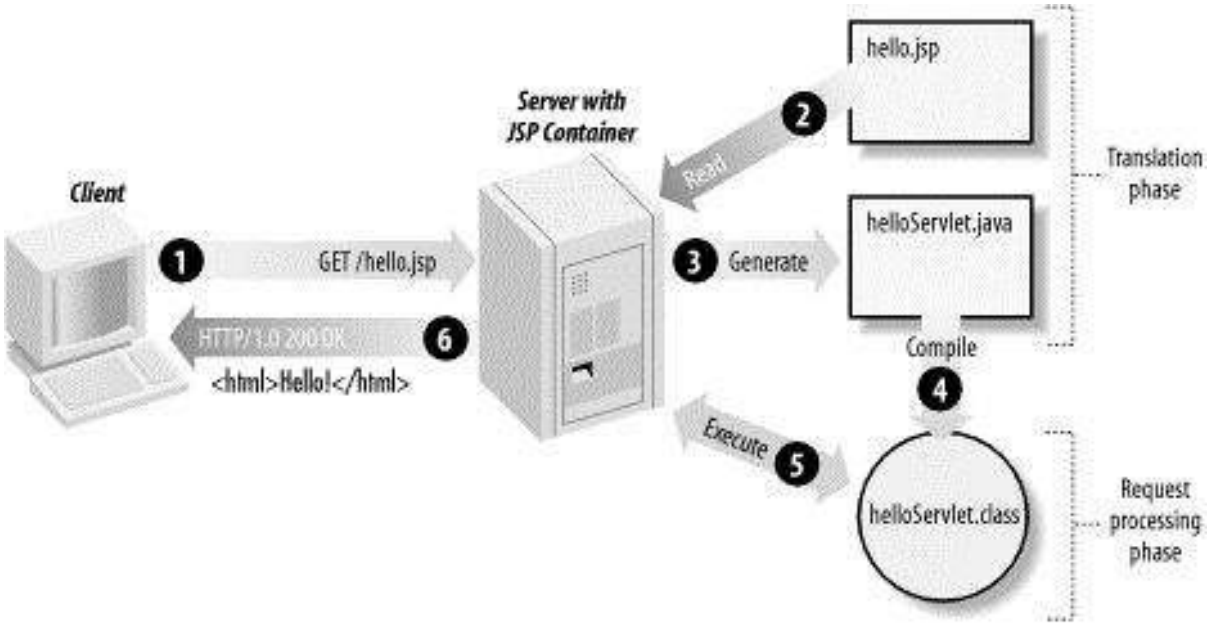
Filter sınıflarda da “init”, “doFilter” ve “destroy” olmak 3 önemli yaşam döngüsü fonksiyonu vardır. Bunların yaşam döngüsü açısından çalışma mantığında yukarıda değindiğimiz Servlet ile aynıdır.

Java Server Pages

JSP, Java ile dinamik web sayfaları oluşturmamızı sağlayan web uygulama geliştirme alt yapısıdır. Temelinde Servlet teknolojisine dayanır. Servlet ile web programlama yapabilmek nispeten daha maliyetli bir yoldur. Bu maliyeti düşürmek ve hızlı geliştirme yapabilmek adına Java Server Pages teknolojisi ortaya çıkmıştır. Halen günümüzde kurumsal projelerde belli noktalarda JSP tercih edilebilmektedir.

JSP, server-side rendering teknolojisine uygundur. Bu kavramı önceki konularda aktarmıştık. HTML halindeki web sayfası sunucu tarafında tamamen oluşturulup istemciye cevap olrak gönderilir. Tarayıcı da bu gelen HTML sayfasını kullanıcı ekranında oluşturur.

JSP ile hazırladığınız sayfalarda HTML blokları içerisine Java kodu yazabilirsiniz. <% Java code %> şeklinde yüzde işaretleri arasında Java kodları yazılabilir. Tabi bu yöntemin MVC tasarım desenine çok uygun olmadığını söylemek gerekir.



Yukarıda JSP ile geliştirilmiş bir projeye istek geldiğinde yaşananlar tasvir edilmiştir. İstemci hello.jsp isminde web sayfasını sunucudan talep eder. Sunucu ilgili JSP sayfasını bulur. Sonrasında JSP Container bu JSP sayfası için otomatik olarak arka planda bir Servlet oluşturur. “helloServlet.java” isminde bir Java kod bloğu oluşur. Bu Java dosyası derlenerek Byte koda çevrilir. Biliyoruz ki Java’da derlenen kaynak kod ara bir dile dönüştürülür. Ardından, oluşturulan Servlet içinde yazılan kodlar icra edilir, örneğin veri tabanından kayıt çekmesi gerekiyorsa onlara erişir. Servlet’in çalıştırılmasından hemen sonra elde edilenler ile bir HTML sayfa oluşturulur. Malum ki istemci tarafı yani tarayıcılar sadece HTML’den anlamaktadırlar.