Modern JavaScript

Fatih Kadir Akın

İçindekiler

1

Temel Seviye: JavaScript'i Çalıştırmak

Bu Bölümde

JavaScript ilk önce bir scripting dili olarak ortaya çıktı. Yalnızca browser (tarayıcı) üzerinde çalışacak, temel sisteme yardımcı bir dil olarak hizmet edecekti. Web'in gelişimi önüne geçilmez bir hal alınca, JavaScript dünyanın en önemli programlama dillerinden biri haline geldi. HTML5'in yükselişi, mobil uygulamaların HTML5'e olan tam desteği, Google ChromeOS ve JavaScript'in Node.js ile sunucu tarafına taşınması, gerçek zamanlı uygulamaların JavaScript ile kolayca geliştirilebilmesi, HTML5 ve Canvas'ın Adobe Flash'ı çok gerilerde bırakması, Cloud'un vazgeçilemez avantajları ve JavaScript'in de Cloud'da etkin rol alması gibi olağanüstü gelişmeler, JavaScript'i çok değerli bir yere koydu.

Arayüz geliştirme yalnızca tasarımdan çıkıp mühendislik yapmaya elverişli bir hale geldi. Öyle ki bu mühendislik artık sadece arayüz geliştirmede değil, JavaScript ile robot programlamaya kadar ilerledi. Biz de tüm bunları sağlayan, internet kullanan her kullanıcının belki de bilmeden tüm gününü çalıştırmakla geçirdiği JavaScript'i yazmayı öğrenmeye koyulacağız.

NOT Cloud, veya Cloud Computing (Bulut Bilişim); Web'in gelişimi, internet hızlarının artması, sunucuların karşılayabildiği yük miktarının artması ile birlikte doğmuş bir kavramdır. Kavramı konusunda tamamen kesinleşmiş bir ifade olmamasına karşın verilerin bulut diye tabir edilen "online bilgi ağı" üzerinde saklanması, dağıtılması ve biriktirilmesi işlemlerinin tümüdür. Tüm bu işler internet üzerinden yürütülür. Ve bu göz önünde bulundurulursa, browserların en yardımcı dili olan JavaScript'in popüleritesini sağlayan unsurlardan birinin de Cloud olduğu düşünülebilir. Cloud geliştikçe masaüstü uygulama sayısı azalacak ve yerine browser tabanlı çok yetenekli uygulamalar gelecektir. Bu çok yetenekli uygulamalardan biri ve en klişe örneği olarak da Google Docs gösterilebilir. Hatta Google, ChromeOS ile sadece Cloud üzerinde çalışabilen bir işletim sistemi geliştirmiş ve Cloud'un gelmesini istediği yeri de hedeflemiştir.

JavaScript Neye Benzer?

Zihninizde bazı şeylerin canlanması, onu daha iyi öğrenmenizi sağlar. Bu açıdan bir JavaScript kodunun neye benzediğini görelim. Aşağıda yazdığımız kod kısa ve karmaşık bir JavaScript kodudur. Eğer yeni iseniz, bu kodu anlamamanız gayet normal. Fakat ileride kodların size tanıdık gelmesi sizin için bir avantaj.

```
// Degisken tanimlamalari
var
  _nesne = {}, // virgul ile degisken belirleme
  _sinif = function (parametre, baskaparametre) { // fonksiyon veri turu
    if (parametre) { // kosullar
      this.ozellik = "deger"; // deger atama
    } else {
      this.degerler = []; // dizi tanimlama
      (function(self) { // kendiliginden calisan fonksiyonlar
        baskaparametre.forEach(function (deger) { // mapping / dongu isleri
          self.degerler.push(deger); // dizi yonetimi
        });
```

3

HTML/DOM Üzerinde JavaScript

Bu Bölümde

Şimdiye kadar gördüklerimiz, biraz "işe yaramaz" şeylerdi. "İşe yaramazlık" derken kastettiğim şey, bunlarım önemsiz olduğu değil, neyi nerede kullanacağınızı bilememenizdir. Bu çok normal. Gördüğümüz şeylerin tümü, şimdi yapacaklarımızın temellerini oluşturuyordu. JavaScript bilindiği gibi, tarayıcıların en güçlü silahı, en büyük destekçisi. Eskiden yaptığı bir çok angarya işi artık CSS'e bıraktığını düşünürsek, eskiden daha çok şeye malzeme olurken, şimdilerde CSS ile birlikte müthiş uygulamalar yapmak için harika bir yardımcı. HTML ve DOM'un ne olduğunu bilmiyor olabilirsiniz. Bu durumda bu kısmı atlayarak sonraki kısımlara geçebilirsiniz.

HTML NEDİR?

HyperText Markup Language, diyerek uzatabileceğimiz bu kısaltma, internetin en büyük ve en eski teknolojilerinden de biri. **Markup Language**, yani işaretleme dili olarak çevirebiliyoruz. İşaretleme dilleri, basitçe kümelerdir. Ben HTML'i anlamanız için kısaca "küme" örneğini kullanacağım. "İşaretlemek" aslında kümelemektir.

NOT Ben HTML'e çok hızlı değinip geçeceğim. HTML'i öğrenebileceğiniz HTML5 kitapları vardır. Dikeyeksen Yayınlarından HTML5 ve CSS3 kitabını tavsiye ederim. Satın alıp okuyabilir veya internet üzerinde milyonlarca kaynağa gözatabilirsiniz.

Kümeleme günlük hayatta sürekli yaptığımız bir işlemdir; hemen bakalım:

» Araba
- Ön Kapılar
  - ◊ Sağ Kapı
    - * Kapı kolu
    - * Cam
    - * Ayna
  - ◊ Sol Kapı
    - * Kapı kolu
    - * Cam
    - * Ayna
- Arka Kapılar
  - ◊ Sağ Kapı
    - * Kapı kolu
    - * Cam
  - ◊ Sol Kapı
    - * Kapı kolu
    - * Cam
- Bagaj
- Kaput

4

CSS ve JavaScript

Bu Bölümde

Bir arayüz geliştiricinin temel olarak bilmesi gereken dillerden biri olan CSS (Cascading Style Sheets, *Katmanlı Stil Dosyaları* şeklinde Türkçeleştirilebilir) de en az HTML kadar JavaScript ile bağlantılıdır. Öyle ki, işin CSS kısmı işin tasarım ve yaratıcılık kısmını tetiklediğinden kullanıcının en çok gördüğü kısımdır. Yani bir arayüz geliştiricisinin geliştirdiği arayüz kullanıcılar tarafından yazdığı CSS üzerinden not verilir. İstediğiniz kadar iyi JavaScript veya HTML bilin, CSS'iniz iyi değilse pek iyi görünmeyen arayüzler çıkarabilirsiniz. Bu açıdan CSS'in önemini ve JavaScript ile bağlarını önemsemek ve uygulamak gerekir.

CSS NEDİR?

CSS, bir sayfanın stillenmesine yardım eden bir tanımlama listesidir. CSS3 getirdiği yeni özelliklerle yalnızca stillendirme değil, animasyon ve bir çok interaksiyonu da geliştiricilere sunmuştur.

NOT HTML, CSS ve JavaScript arasındaki ilişkiyi bir anatomiye benzetebiliriz.
HTML; İskelet yapı, temel interaksiyonların gerçekleşmesini sağlayan yapı. HTML, CSS ve JavaScript düşünerek yazılmalıdır. Zira eklem olmayan bir yere istediğiniz kadar kas koyun bir anlam ifade etmeyecektir.
CSS; Yüzey yapısı ve estetik. Bu iskelet yapısına biraz estetik kazandırmak çok yerinde olacaktır. Zira bu her ne kadar mühendislik ve matematik de olsa, kullanıcı her zaman estetiğe dikkat eder.
JavaScript ise, bu yapının hareket ve interaksiyon mekanizması, mekaniğidir. JavaScript için sayfaların mekaniği diyebiliriz. Veya canlı bir anatomi için kas sistemi. Bu açıdan bakıldığında bu hiyerarşinin yeterince anlaşıldığını düşünüyorum.

CSS DOSYALARI

CSS dosyaları, *.css* ile biten dosyalardır ve bir CSS dosyası şuna benzer:

```
html,
button,
input,
select,
textarea {
    color: #222;
}

body {
    font-size: 1em;
    line-height: 1.4;
}

::-moz-selection {
    background: #b3d4fc;
    text-shadow: none;
}

::selection {
```

```
    background: #b3d4fc;
    text-shadow: none;
}

img {
    vertical-align: middle;
}

fieldset {
    border: 0;
    margin: 0;
    padding: 0;
}

.button {
    border: 0;
}
```

CSS içerisindeki tanımlamlar şu şekilde özetlenebilir.

```
<seçici> {
    <özellik>: <değeri>
}
```

CSS dosyaları sayfaya `link` tagı ile bağlanabilir veya sayfa içerisinde `style` tag'i olarak kullanılabilir. Ayrıca elementlere `style` ve `class` `attribute`'larıyla da eklenilebilir.

```
<!doctype html>
<html>
  <head>
   <title>JavaScript!</title>
   <link rel="stylesheet" href="main.css">
   <script src="main.js"></script>
 </head>
 <body>
    <h1>Merhaba!</h1>
   <p>
     Bu benim ilk HTML Sayfam ve CSS var!
   </p>
 </body>
```

6

Test Güdümlü JavaScript Geliştirme

Bu Bölümde

Test Driven Development, kısaca TDD, Türkçe'ye "Test Güdümlü Geliştirme" olarak çevrilmiş bir programlama anlayışıdır.

NİÇİN TEST YAZMAMIZ GEREKİYOR?

Geleneksel kodlamada işler şu şekildedir;

» Sınıfı (sınıf bile olmayabilir) yaz,
» Kodu deploy et,
» Hata bildirilirse hatayı bul ve kodu kendin düzelt.

Biraz daha geliştirilmiş geleneksel kodlamada ise işler şu şekildedir;

» Sınıfı yaz,
» Olası case'ler içine loglar koy,
» Kodu deploy et
» Hata bildirilirse loglara bak ve kodu düzelt.

Fakat bu geleneksel yöntemler nesne yönelimli programlama sırasında biraz canınızı sıkabilir, çünkü bu kez **log**lamaları koymanız gereken yer sizin sınıflarınız olacak ve her ne kadar kod tekrarını engellemiş olsanız da, tekrar kullanılabilirlik o derece aşağılara düşecektir. Yani aslında yaptığınız şey OOP değil OOP görünümlü bir yapı olacaktır. Hem stabilizasyonu artırmak hem de daha hızlı ve düzgün kod yazmak için kullanılması gereken yöntem ise TDD'dir.

İnternette TDD ile ilgili bulduğunuz dökümanları aradıktan sonra kendinize şu soruyu soruyorsunuz genellikle; "Niçin test yazayım ki, ne gereği var?" ve ardından; "Test yazacağıma kod yazarım, test yazmakla vakit harcamaktansa.." düşünceleri kafanızda dönüyor. Fakat işler tam düşündüğünüz gibi değil.

ÖRNEK

Şimdi biraz örnekle inceleyelim; diyelim ki bir matematik sınıfımız var. Klasik dört işlemi yapabilsin, yani toplama, çıkarma, çarpma, bölme işlemleri için birer metod olsun. Normalde nasıl bir sınıf yazardık?

```
Maths = {

  add: function (a, b) {
    return a+b;
  },

  subst: function (a, b) {
    return a-b;
  },
```

```
  div: function (a, b) {
    return a/b;
  },

  multiple: function (a, b) {
    return a*b;
  }
};
```

Veya daha zekice işlemler, daha farklı implementasyonlar olabilir. TDD size kod yazmayı sevdirirken yazdığınız kod kadar da iyi kod yazmaya yönlendiriyor.

İLK OLARAK TEST YAZILMALI

Nasıl yani? Olmayan bir sınıfı mı test edeceksiniz? Evet, aynen öyle. Çünkü TDD testten geçmeniz için var olan eksiklerinizi söyleyecek, sınıf yazılması da dahil. Hani okulda sınavdan geçemediğimiz zaman öğretmenlerden hangi soruda hata yaptığımızı öğrenmek isterdik ya, diğer sınavı geçebilmek adına, mantık tamamen aynı işliyor.

TEST SINIFI

Önce bomboş bir *Maths.js* oluşturun, daha sonra ise aşağıdaki örnekteki gibi bir *MathsTest.js*

```
<html>
  <head>
    <script src="Maths.js"></script>
    <script src="MathsTest.js"></script>
  </head>
</html>
```

MathsTest.js:

```
console.group('Ekleme fonksiyonu');
console.assert(Math.add(1, 1) == 2, "1 + 1 = 2 olmalı");
console.assert(Math.add(1, -1) == 0, "1 + (-1) = 0 olmalı");
console.assert(Math.add(-1, -1) == 0, "(-1) + (-1) = -2 olmalı");
console.groupEnd();

console.group('Çıkarma fonksiyonu');
console.assert(Math.subst(1, 1) == 0, "1 - 1 = 0 olmalı");
console.assert(Math.subst(1, -1) == 2, "1 - (-1) = 2 olmalı");
console.assert(Math.subst(-1, -1) == 0, "(-1) - (-1) = 0 olmalı");
```

7

JavaScript Front-End Frameworkler

Bu Bölümde

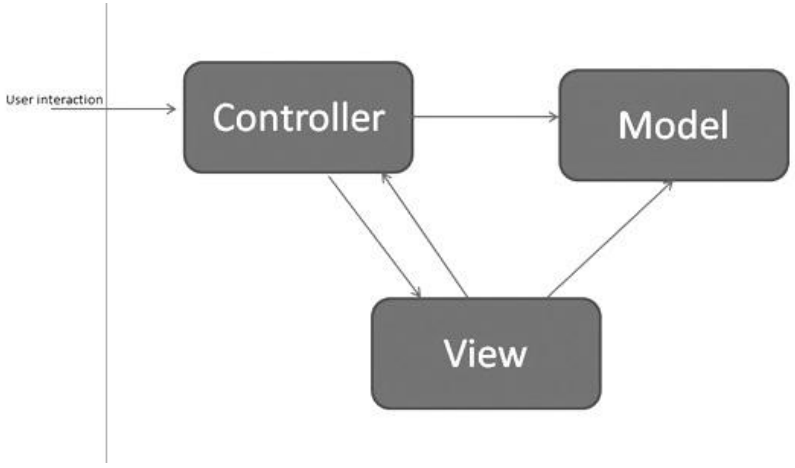
JavaScript, geliştiği dönem süresince insanlar tarafından anlaşılması ve kullanılması zor bir dil olarak değerlendirildi ve bir çok geliştirici hala aynı şekilde düşünüyor. Bu zorluk algısı insanları bazı kolaylaştırıcı etkenlere yöneltti ve bunun sonucunda da günümüzde yükselişe geçen frameworkler oluştu.

MVC, MVP, MVCP, MVVM NEDİR?

Öncelikle bu işin temelini oluşturan **MVC** (*Model View Controller*), **MVP** (*Model View Presenter*), **MVVM** (*Model View View Model*) tasarım şablonlarının tanımlarından bahsedelim. Sonrasında ise hangi framework'ü neye göre seçeceğimizi konuşacağız.

MVC

MVC tasarım şablonu, en yaygın kullanılan, özellikle **back-end** yapılarda en uygun tasarım şablonudur. Aşağıda görülen bürokrasi ile işler yürür.



Görüldüğü gibi, kullanıcılar interaktiviteyi **Controller**'a **View** üzerinden erişerek sağlarlar. **Model** ise bir nevi veri birimlerini yöneten bir işlev görür ve kimi zaman, Controller ile View arasına girerek belli yapılardaki veriyi taşıma işlevi görürler. **Zend Framework**, **Symfony**, **Django**, **CodeIgniter** vb. framework'ler bu işlevi yerine getiren yapılardandır.

NOT Ayrıca daha önceden PHP ile geliştirmiş olduğum MVC framework'ü bu işlevi yerine getiren unsurlardan birisidir. Oldukça basit yazılmış olan bu sistem, MVC'yi doğru anlamlarıyla kullanır ve Object Oriented'ın önemli kurallarına uyar. Nasıl çalıştığını merak ediyorsanız, kodları incelemek için GitHub hesabımdaki ***http://github.com/fkadeveloper/mvc*** projesine gözatabilirsiniz.

Backbone.js, bu işi client-side yapan framework'lerden birisidir. Fakat bilindiği gibi, client-side'da işler biraz farklı yürüyor. Veri barındırma, veri saklama gibi

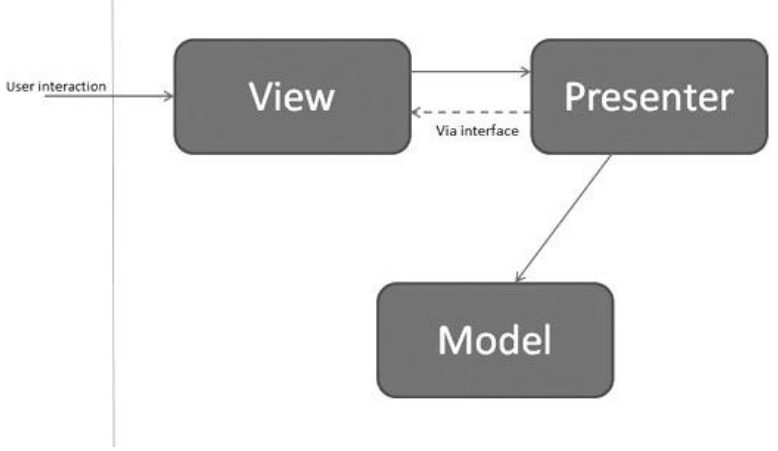
işlemler daha çok server tarafında yapıldığından bu, yeni başlayanlarda soru işaretleri yaratabiliyor. Özellikle öncesinde bir MVC geçmişi olanlar bu işin client-side'da nasıl yürüdüğüne dair bir takım sıkıntılar yaşıyorlar. Ben de bir Zend Framework kullanıcısı olarak Backbone'u öğrenirken bu tarz sıkıntılara aşina oldum, fakat Jack'i geliştirdiğim zaman her şey daha berraklaşmıştı.

O halde kafamızda soru işaretleri kalmaması adına, biraz daha derinlere inelim ve MVP tasarım şablonu nedir, onu öğrenelim.

MVP

Model-View-Presenter olarak açılan bu tasarım şablonunda görülen Presenter öğesi, Controller yapısının biraz daha view'e yaklaşması olarak nitelendirilebilir. Presenter ismi ise, bence, tam oturmuş bir isimdir. Çünkü aslında hem bir şeyler göstermekte, hem de yönetmektedir. Yani tam olarak sunmaktadır. Yani ne görünüm (*view*) kadar sabit ve işlevsiz, ne de kontrolcü (*controller)* kadar sadece işleve yarayandır. View'e biraz daha fazla karışır, fakat Model ile de ilişkilidir. MVVM'e çok benzer (onu da anlatacağım), fakat daha az karmaşıktır. İş yükünün büyük kısmı Presenter'a düşer.

Kısaca bir şablon ile özetlemek gerekirse;



Backbone, yapısı itibariyle MVP'ye benzer. Backbone'un view'ları **Presenter** gibi hareket ederler. Ayrıca **Router** yapısını içermesi, klasik MVC gibi de kullanılabilmesi sebeplerinden ötürü, MVC'ye de benzer. Bu yüzden Backbone için MV* gibi bir tabir koyanlar da olmuştur.