T.C.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

NODEJS İLE SOHBET UYGULAMASI

Mustafa YILDIRIM

BİLGİSAYAR MÜHENDİSLİĞİ TASARIM ÇALIŞMASI I

DANIŞMANI : Murat ÖZALP

BİLECİK

3 Haziran 2019

T.C.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

NODEJS İLE SOHBET UYGULAMASI

Mustafa YILDIRIM

BİLGİSAYAR MÜHENDİSLİĞİ TASARIM ÇALIŞMASI I

DANIŞMANI : Murat ÖZALP

BİLECİK

3 Haziran 2019

ÖZET

Projenin Amacı

Web üzerinde responsive olarak çalışacak mesajlaşma uygulaması yapmak.

Projenin Kapsamı

Proje kapsamında front-end bağlamlarımı bir arada tutmak için bower compenents kullanarak angular dosyalarımı burada tuttum. Back-end yapılandırmalarımı Node.JS ile gerçekleştirip veri tabanında NoSQL yapısını tercih edip MongoDB kullandım bunlara ek olarak gerçek zamanlı işlerimi gerçekleştirebilmek için Ajax yerine Socket.Io ile işlemlerimi gerçekleştirdim son olarak Redis ile tüm session bilgilerini, mesajları, odaları ve aktif olan kullanıcıları tuttum.

Sonuçlar

Projenin kapsamına uyularak ilerlenmiş ve gerekli tüm yapılandırmalar yapılarak proje digital ocean ile canlı ortama taşınmıştır.

ABSTRACT

Project Objective

Do the messaging app that will run responsibly on the web.

Scope of Project

I kept my angular files using bower compenents to keep my frontend contexts together. I have implemented backend configurations with Node.JS and I have chosen the NoSQL structure in the database. I used MongoDB to perform real-time tasks in addition to Ajax instead of Socket.Io and finally I used all the session information, messages, chambers and active users with Redis.

Results

The project was carried out in accordance with the scope of the project and all the necessary configurations were made and the project was carried to the live environment with the digital ocean.

TEŞEKKÜR

Bu projenin başından sonuna kadar hazırlanmasında emeği bulunan ve beni bu konuya yönlendiren saygıdeğer hocam ve danışmanım Sayın Murat ÖZALP'e tüm katkılarından ve hiç eksiltmediği desteğinden dolayı teşekkür ederim.

Mustafa YILDIRIM

3 Haziran 2019

İÇİNDEKİLER

ŞEKİL LİSTESİ

1 GİRİŞ

1.1 Projenin Genel Hatlarıyla Anlatımı

Bu alanda projeyi özet olarak tekrardan anlatmak istiyorum. İlk olarak projeyi anlatayım. Aslında bakıldığı zaman tamamiyle bir mesajlaşma uygulamasıdır ama neden böyle bir uygulama yazdım. Kullandığım teknolojiler neler bunlar hakkında biraz bilgi vereyim. Yazdığım projede socket.io ile gelen mesajları, oda ekleme işlemi ve aktif olan kullanıcıları anında görmemize imkan sağlamaktadır. Peki nedir bu Socket.Io ve neden Ajax kullanmadım da Socket.Io kullandım. Durumu şöyle anlatabiliceğimi umuyorum. Eğer iletişimimiz tek bir yönde ise örneğin müşteri her zaman sunucuyla iletişimi başlatan kişiyse AJAX'ı kullanmak daha verimli olacaktır. Çünkü daha hızlı uygulayabiliriz ama burada sunucu ile müşteri arasında her iki yönde akıcı bir şekilde iletişim kurmamız gerektiği için Socket.Io seçimi bizim için en doğru seçim olacaktır ki Socket.Io'nun bir başka büyük avantajı, tüm müşterilerle belirli bir anda çevrimiçi olarak iletişim kurabilmemiz. İkisi arasında karşılaştırma yaptığımızda ve yapacağımız işin mesajlaşma olduğunu düşünürsek en mantıklı tercihin Socket.Io olduğu oldukça aşikardır. Tamam Socket.Io'yu kullandık projeyi geliştiriyoruz peki bundan sonrası nasıl olacak burada yapmamız gereken iki olay var verilerin kaydedilmesi ve Socket.Io ile tuttuğumuz verilerin frond-end de gösterilmesi. Önce frond-end de gösterilmesi üzerine duralım. Burada bize client bağlantısı olan ve frond-end için kullanabileceğimiz frameworklerden birini seçersek işlerimiz bir hayli kolay olacak ve işleri o kadar hızlı ilerleteceğiz ben bunun için Angular'ı tercih ettim Vue.Js, React veya diğerleri de seçilebilirdi. Benim Angular'ı seçmemin nedenleri arasında frameworkler arasında en olgun olanı olması, katkıda bulunanlar açısından iyi bir desteğinin olması ve en önemlisi eksiksiz bir paket oluşudur. Şimdi frond-end yapılandırmamızı da hallettiğimize göre uygulamamızın veri tabanı için kullandığım iki veri tabanını anlatmaya başlayım. Bu veri tabanlarından birisi sadece kullanıcılarımın Google Id gibi giriş bilgilerini tutmaktadır. Bu NoSQL olan MongoDB ile yapılmaktadır. Bunun haricinde tüm mesajların tutulduğu, online olan kullanıcıların tutulduğu, kullanıcıların session bilgileirinin tutulduğu ve son olarak eklenen odaların tutulduğu Redis'dir.. Bu yapıların hepsi ve kullandığım paketler ileride ayrıntılı bir biçimde anlatılacaktır. Yani sonuç

olarak burada yaptığımız iş ön tarafta Framework olarak Angular kullanarak, html yerine pug tercihi yaptım sayfaları yazarken bununda birkaç nedeni vardır ileride anlatacağım arka tarafta Node.js'in paketi olan Socket.Io ile işlemleri yürüttüm ve redis ile verilerimi tutarak bir mesajlaşma uygulaması yaptım.

1.2 Neden bu projeyi seçtim

Bu projeyi seçmemin en büyük etkeni proje hocamdır. Bana daha önce yaptığım projeden farklı bir proje olması ve CV'ne yazacağın farklı şeyler olsun diye birkaç kere tavsiye verdiğinden dolayı ve php ile node.js alanındaki işlerin ülkemizde çok yakın dış ülkelerde ise node.js'in önde olmasınından ötürü böyle bir proje yapmayı tercih ettim neticede bu projedeki temel amacım Node.Js öğrenmek ve CV'me yazılabilecek birkaç satır eklemekti.

1.3 Bu proje ile hedeflenen ve faydaları

Bu proje ile Socket.Io ile gelen verilerin anlık olarak işlenmesi ve Redis'e kaydedilmesi hedeflenmiştir. Bunlarda Socket.Io mesajlaşma uygulamaları için biçilmiş kaftandır oldukça hızlı bir şekilde kullanıcıların haberleşmelerini sağlamaktadır. Diğer taraftan Redis en basit haliyle, key-value şeklinde tasarlanmış bir NoSQL veritabanıdır ve çok hızlıdır bunun nedenide Memcached gibi verileri HDD yazmadan RAM üzerinde tuttuğundandır. Bu ikiliyi birleştirdiğimizde ise bize oldukça hızlı bir mesajlaşma platformu hazırlanmış oluyor.

1.4 Kullanılan Bileşenler

Bu alanda kullanılan bileşenler kısa bir şekilde anlatılacak olup daha ayrıntılı bilgi için bir sonraki bölüme bakmanız proje sorumlusu tarafından önerilmektedir. Başladığımızdan beri hep node.js den bahsetmekteyiz peki biz bu node.js'i hangi dille yazıyoruz şimdi herşeyi bir sırayla teker teker ele alalım.

1.4.1 JavaScript

JavaScript, yaygın olarak web tarayıcılarında kullanılmakta olan bir betik dilidir. JavaScript ile yazılan istemci tarafı betikler sayesinde tarayıcının kullanıcıyla etkileşimde bulunması, tarayıcının kontrol edilmesi, asenkron bir şekilde sunucu ile iletişime geçilmesi ve web sayfası içeriğinin değiştirilmesi gibi işlevler sağlanır. JavaScript, Node.js gibi platformlar sayesinde sunucu tarafında da yaygın olarak kullanılmaktadır.

İlk olarak bir Netscape çalışanı olan Brendan Eich tarafından geliştirilen ve 1997 yılından beri bir ECMA standardı olan JavaScript, günümüzde Mozilla Vakfı öncülüğünde özgür yazılım topluluğu tarafından geliştirilmekte ve bakımı yapılmaktadır. Web sayfalarının görünümünün mükemmelleştirilmesi, işlevselliğinin artırılması ve dinamik web sayfaları tasarlanması amacıyla JavaScript, HTML kodu içinde gömülü olarak (embedded) kullanılabilir. JavaScript, bir programlama dili disiplini ve özelliklerine sahiptir.

1.4.2 Node.JS

Node.js, JavaScript'dir. Web tarayıcılarına muhtaç kalmadan heryerde çalışabilir. Peki ne işe yarar bir yazılımın hem arkayüz (back-end) hem önyüz (front-end) kodlarını aynı dilde yazmanın tek yoludur. Günümüzde çoğu uygulama, zamanının çoğunu veritabanlarına veya Internet'deki çeşitli servislere istek yapıp gelen sonuçları beklemekle getiriyorlar. Node.js, yapısı gereği asenkrondur. İstekleri paralel olarak yapar. İstekler bittikleri zaman Node'a callback yaparlar. Bu sayede anlık olarak çok sayıda istek hızlı bir şekilde işlenebilir.

1.4.3 Angular

Angular javascript tabanlı açık kaynak kodlu yazılım geliştiricilerin web, mobil ve masaüstü ortamda kolay uygulama geliştirmelerini sağlayan bir platformdur. Ayrıca istemci tarafında çalışan yada daha açık bir ifade ile kullanıcıların görebildiği ve etkileşebildiği ortamlarda (web tarayıcılarında) çalışan bir framework olduğu için bir Front-End Deve-

lopment pratiğidir.

1.4.4 MongoDB

MongoDB bize kendisini, geliştirme ve ölçekleme kolaylığı için tasarlanmış açık kaynak, belge yönelimli(document-oriented) veritabanı olarak tanıtmaktadır. MongoDB'de her kayıt, aslında bir dökümandır. Dökümanlar MongoDB'de JSON benzeri Binary JSON(BSN) formatında saklanır. BSON belgeleri, sakladıkları elemanların sıralı bir listesini içeren nesnelerdir. Herbir eleman, bir alan adı ve belirli tipte bir değerden oluşur.

1.4.5 Redis

En basit haliyle Redis, key-value şeklinde tasarlanmış bir NoSQL veritabanıdır. Memcached gibi verileri HDD yazmadan Ram üzerinde tutmaya yarayan bir platformdur. Memcahedden farklı olarak NoSql mantığıyla çalıştığı için serverin kapansa dahi verilerin kaybolmasına izin vermez.

2 KULLANILAN TEKNOLOJİLER

Bu bölümde projede kullanılan teknolojilerin tamamı ayrıntılı bir biçimde anlatılacaktır.

2.1 Node.Js'e Giriş

NodeJS, server-side (sunucu tarafli) uygulamalar geliştirmek için üretilmiş bir teknolojidir. İlk yayınlandığı 2009 tarihinden bugüne diğer geleneksel sunucu taraflı uygulama teknolojilerinin aksine, birçok alanda farklı yapısıyla türünün öncüsü olmuştur. **Ryan Dahl** henüz 1 sene önce (2008 yılında) Google'ın Chrome tarayıcı için geliştirmiş olduğu **v8 engine**'in üzerine oluşturduğu mimariyle, **JavaScript** ile **Back-end** uygulama geliştirebileceğini göstermiştir. Bu noktada v8'in yapısı gereği sadece Web Browser üzerinde JavaScript çalıştırmak için yapıldığından Back-end kısmında dosya işlemleri gibi birçok özelliği içinde bulun**ma**maktaydı. NodeJS bunun gibi pek çok ekstra yapıyı ve **V8**'i extend ederek bünyesinde barındırmaktadır.

V8 engine C++ diliyle geliştirilmiştir. Runtime'da JavaScript kodlarını makina koduna dönüştürür ve sonrasında native makina kodunu çalıştırır. Hem Node projelerinin hem Chromium projesinin bu kadar performanslı olmasının arkasında yatan sebep budur.

NodeJS, **Single Thread Non-Blocking IO**, Asenkron programlama gibi pek çok özelliği ile öne çıkmaktadır.

NodeJS **Community**'si oldukça büyük olmasının yanında Yahoo, Microsoft, LinkedIn, Uber, Paypal gibi pek çok dev firmanın yazılım sistemleri içerisinde bir pay edinmesi bu başarını gösteren en önemli kanıtlardan biridir.

Asıl amacı Back-end projeler geliştirmek olmakla birlikte NodeJs, Front-end otomasyonu ve package management gibi yazılımların pek çok alanında çok yönlülüğü ile karşımıza çıkmaktadır.

2.2 Ecmascript 6 ve NodeJS

Bu proje hazırlandığı aşamada Ecmascript 7 ve 8 çıkmış bulunmaktadır. Fakat ben projemde başlangıç itibariyle kaynak bulurken sıkıntı çekmemek ve daha fazla örneğe ulaşabilmek adına ES6 kullandım yaptığım bilgilendirme de ona göre olacaktır. Öncelikle **"Ecmascript"** ve **"JavaScript"** farkı nedir, çok fazla tarihçesine girmeden özetliyorum. Ecmascript, JavaScript dilinin standartlaştırılan sürümünün adıdır (Bu standart Ecma-262 olarak da ifade edilmektedir). Diğer bir değişle, JavaScript dilinin standardının ismidir diyebiliriz. Haziran 2015 itibariyle Ecmascript 6 kullanılmaya başlanmıştır. Bu versiyonla birlikte pek çok yeni özellik hayatımıza girmiştir.

NodeJs içerisinde tüm ES6 özellikleri **Shipping, Stage** ve **Progress** olarak 3 grupta incelenmektedir.

Shipping: Artık NodeJs üzerinde **default** olarak kullanıma açılmış ve herhangi bir tanımlama yapmadan kullanabileceğimiz özellikleri içerir.

Staged: Özel bir flag ile *"-harmony"* kullanabileceğimiz ancak %100 olarak tamamlanmamış özellikleri içerir.

In Progress: Henüz implementasyonu tamamlanmamış özelliklerdir. Özel bir flag'le bu özelliklerde kullanılabilir, ancak isminden de belli olduğu gibi özellikle production'a dönük uygulamalarda uzak durmamız gerekiyor.

2.2.1 Arrow Function

Execution Context, belki de JavaScript geliştiricilerin en çok zorlandığı konuların başında geliyor. **Execution context** nasıl oluşur veya **this** keyword'ü neden window objesini gösterir gibi (çünkü nesneden bağımsız çağrılıyor) küçük nüanslar problem yapabiliyor. Bu noktada, arrow function özellikle fonksiyon içerisinde tanımlanan fonksiyonları bulunduğu context'e otomatik olarak bağlamasıyla (bind metodu) protik anlamda kolay bir kullanım sağlıyor.

2.2.2 String Interpolation

Belki ilk bakışta neden buna gerek var ki sorusu aklınıza gelebilir, ancak yeni programlama dillerinde ve eski dillerin yeni versionlarının çoğunda bulunan **String Interpolation** ifadelerine alışmalıyız. Çünkü dillerin çoğunda, String değerler **immutable** olarak tutulurlar ve bu String üzerinde değişiklik yapmak için arkaplanda da olsa pek çok nesne oluşturmak zorunda kalınabilir (performans problemi çıkarabilir). Bunun yanında **interpolation** kullanımının pratik olarak yazımı kolaylaştırması ve Multiline String ifadeler kullanmamıza desteklediğinide belirteyim.

2.2.3 Class Yapısı

JavaScript programlamanın **prototypical inheritance** konusu genellikle birçok geliştiricinin kafasında hep bir soru işareti bırakmıştır. Bu konuyla birlikte Prototype'lardan kurtulduk mu diye düşünüyorsanız yanılıyorsunuz. Çünkü Class kullanımı ile JavaScript'in kalıtım yapısı değişmiyor.

Daha da basit bir ifadeyle, klasik programlama dillerinde olan **Class Based Inheritance** prensibine uygun olarak kod geliştirenlere genellikle ters gelen prototypical inheritance JavaScript'in belki gelişim aşamasını bile etkilemiştir, çünkü anlaşılması hep zor gelmektedir. Bu durumda Ecmascript 6 ile birlikte **proptypical inheritance** JavaScript tarafından görüntüde terkedilmiş gibi olsa ve **class based inheritance** yazılıyor gibi görünse de aslında kaputun altında hala prototypical inheritance kullanılmaya devam edecektir. Sonuç olarak, biz class tanımlamayla ilgili class'a metotlar eklediğimizde, JavaScript bunu prototypical inheritance prensiplerine uygun olarak yorumlamaya devam edecektir.

2.2.4 Let Const İfadeleri

Ecmascript 6 ile gelen yeni tanımlama ifadeleridir. **Var** değişkenininden farklı olarak, block scoped olarak ifade edilebilirler. Yani **var** değişkeninin sahib olduğu ve çoğu yazılım geliştiricisinin kafasını karıştıran ve uygulamamızın biraz da kararlılığını etkileyen

bu duruma çözüm sağlayacaktır.

```
function a(){
if(true){
var b = 5;
}
console.log(b);
}
```

Yukarıdaki gibi bir kullanım, mesela Java'da olsa kodunuz hata verecekti. Çünkü if kendi scope'unu oluşturacaktı ve if bloğunun içerisinde tanımlanan bir değişken block dışında kullanılamaz olacaktı. Ancak **var** ile tanımlanan değişkenler if'i veya for gibi ifadeleri tanımazlar.

Bu durumda yeni gelen **let**'in, **var**'ın block scope tanıyan tipi olduğunu düşünebiliriz.

Aynı şekilde **const** ifadesi de, block scope tanımasının yanında isminden de belli olduğu gibi sabittir.

```
function a(){
if(true){
let b = 5;
}
console.log(b);
}
```

Yukarıdaki gibi kullanıldığında ise console.log(b) kısmı çalışmayacaktır çünkü b tanımlanmadı.

2.3 Node.JS Mimarisi ve NPM

2.3.1 Event Driven Programming

Event Driven Programming, uygulamanın akışını event'lerin karar verdiği uygulama tipidir. Yani uygulama içerisindeki akış, yayımlanan event'lerin tetiklediği kod blokları ile ilerler. Bu event'ler bazen kullanıcının bir input üzerinde yaptığı değişiklikler olduğu gibi bazen de model katmanınızdaki değişiklikler olabilir. Genellikler kullanım alanlarının başında JavaScript ile Web uygulamarı geliştirme gelmektedir. NodeJs ile Event Driven uygulama geliştirmek JavaScript'in yapısı gereği oldukça basittir.

```
<button onclick="clickme()">Click</button>
//Event handler
function clickMe(){
alert("Kullanıcı Tıkladı");
}
```

Yukarıdaki örnekte kullanıcının tıklaması durumunda **clickMe** ismindeki fonksiyon çalışacaktır. Burada aslında Event driven uygulamanın temel bir örneğini görüyoruz. Burada dikkat etmemiz gereken bir başka nokta JavaScript Engine'in bu event'leri nasıl çalıştırdığıdır (çünkü onlarca event handler aynı anda tetiklenebilir).

Şimdi JavaScript Engine'in bir event loop'u nasıl yönettiğini görelim.

1. Browser içerisindeki JavaScript Engine, bir event loop oluşturuyor.
2. Yukarıdaki onClick metoduyla olduğu gibi callback fonksiyonumuzu herhangi bir olayla ilişkilendirerek kaydediyoruz.
3. Burada clickMe ilgili butona tıklanması durumunda çalışacak kod bloğudur.(Event Handler)
4. Yani clickMe aslında click Handler olarak da ifade etmemiz yanlış olmaz.

5. Öncelikle listener'lar vasıtasıyla event'lerimizi kaydediyoruz. Yukarıdaki örnekte butona click durumu için kayıt işlemi gerçekleştir.

6. Kullanıcı butona basması durumunda bir event, emit edilecek (yayınlanacak), sonrasında JavaScript Engine gelen bu emit'li event sırasının içerisine ekleyerek (aynı anda birden çok event tetiklenebilidiği için JavaScript engine bu event'leri sıraya alır) ve ilişkili olduğu kod bloğunu çalıştırır(Event Handler).

Event Driven Programming prensibi sadece Web değil kullanıcının aktif olarak uygulama ile iletişim kurduğu mobil uygulamalar gibi pek çok alanda kullanılmaktadır(RXJava buna örnek gösterilebilir).

2.3.2 Single Thread, Multi Thread ve Non-blocking IO

Non-blocking IO'a gelmeden önce single ve multi thread kavramlarının da üstenden geçelim. **Single Thread** isminden de belli olduğu gibi **t** zamanında sadece bir adet işlem yapmanıza izin veren yapılardır.

```
var a = 1;
var b = 2;
var c = 3;
var c = c+a+b;
var d = c+a+b;
console.log("c:" + c);
console.log("d:" +d);
```

Yukarıdaki 5 satırlık basit toplamaların yapıldığı bir JavaScript kodu vardır.

JavaScript engine kodu yukarıdan aşağıya okuyacak **c** için "6" ve **d** için de "9" çıktısını verecektir. Zaten JavaScript ile front-end uygulamaları da geliştirirken Browser üzerinde çalışan **engine** bu şekilde ilerliyor. Yukarıdaki oldukça basit bir işlem olduğu için mikrosaniye mertebesinde **engine** bizim için ilk satırdan başlayıp son satıra kadar gelecekti.

Peki bundan daha hızlı yapabilme şansımız var mıydı? Yani kodumuzu biraz düzenleyip iki parçaya ayırıp farklı **threadler**'de çalıştırabilir miydik?

```
var a = 1;
var b = 2;
var c = 3;
var c = c+a+b;
console.log("c:" + c );
(THREAD - 1)

var d = c + a + b;
console.log("d:" + d);
(THREAD - 2)
```

Sorunun cevabı JavaScript için hayır olsa da Java gibi farklı programlama dilleri bu konuda destek veriyor. Bir **t** zamanı için kodun farklı bölümleri aynı anda işleniyor. Böylece **multithreading** ismi verilen bu kavramla bir birim zamanda x yerine 2x kadar kod işleyebilirdir.

Ancak açıkça sonucu olan hız artışı dışında multithreading'in de çeşitli dezavantajları var. Tek thread yerine birden fazla thread çalışabildiği için daha çok sistem kaynağı harcar (**RAM, CPU vs**) Yukarıdaki örnekte de birebir gördüğümüz gibi Thread - 2'de kullanılacak olan a, b ve c değişkenleri Thread - 1'de de kullanılıyor. Yani **Shared Resource** (Paylaşılmış kaynaklar) durumu var ki bunun yönetimi biraz zordur. Buna ek olarak uygulamalarımız yukarıdaki gibi 5-6 satır değil binlerce satır kodla çalıştığıında uygulamanın da yönetimi bir o kadar zorlaşırken, fazladan kullanılan thread'ler fazladan sistem kaynağı harcanmasına neden olabilir. Ancak tabi ki düzgün kullanılan mimarisi oturtulmuş **multithread** uygulamalarının açık bir avantajı vardır. Olaya JavaScript tarafından baktığımızda, JavaScipt'in multithread özelliğini desteklemediğini belirttim. Her kodumuzda maalesef yukarıdaki gibi olmayacak. Bazı durumlar sunucuya **request** atacağız veya kullanıcıdan bir **input** girmesini bekleyeceğiz. Bu durumda uygulamamız ilgili satırın çalışması için tüm Thread'i durduracak mı? Tabii ki hayır. **Non-Blocking IO** uygulama

içerisinde **Network Request**'leri, dosya işlemleri, database işlemleri gibi uzun sürecek işlemler sırasında JavaScript içerisinde zaten tek thread kullanıldığı için bu işlemleri event driven olarak tanımlayıp (Mesela HTTP request tamamlandığında bir event fırlat ve bu kod bloğu çalışsın gibi), kod bloğumuzun bloklanmadan çalışması prensibine dayanır. Bu kullanım geleneksel **back-end** sistemlerine göre oldukça büyük performans artışı sağlar. Neden mi?

Şöyle düşünelim; farklı **back-end** uygulama mimarilerinde kuullanıcıdan gelen istek database ile işlem yapmak gerektiğinde kodumuz Database'e istek attıktan sonra cevabı beklemeye koyulur ve bu bekleme esnasında aslında sistem kaynağı boşuna harcanır. Bu bekleme milisaniyeler mertebesinde olsa da hergün binlerce requesting geldiği bir sistemde milyonlarca database işlemi yapılır. Ve bu doğal olarak çok büyük bir sistem kaynağının boş yere harcanması demektir.

Peki bu durumda **NodeJS()** ne yapıyor? Database'e girmesi gereken durumda bir **callback** fonksiyonu oluşturarak database cavabını gelmesi durumunda bir event, emit edilerek callback metodunun işlemesi sağlanıyor. Bu bekleme esnasında JavaScipt Engine event loop'unu çalıştırmaya devam ederek başka işlemler yapmaya devam ediyor. Bu durumda sistem kaynağı bekleme esnasında boşuna harcanmaktan kurtulup diğer işlemlerde kullanılıyor.

2.4 NPM (Node Package Manager)

NodeJS yapısı gereği küçük bir çekirdek sunarak oluşturacağımız uygulamaların özelliklerinin sonradan bizim tarafımızdan yazılacak veya 3. Parti olarak eklenecek paketler halinde geliştirilmesini teşvik etmektedir. NodeJS sadece dosya işlemleri gibi **low level** uygulamanın bulunduğu ortamla ilişki kurmasını sağlayacak işlerde **API** saplamakta, diğer tüm işlemler için ekstra olarak modüller eklememiz gerekiyor. Diğer yandan kullanacağımız bu modüllerden birçok ekstra modüle bağlılığı olacaktır. Bu durumda baktığımızda bir çok paketin kullanıldığı bir uygulamada Node Package Manager'e ihtiyacımız vardır.

NPM paketlerimizin yüklenmesi için iki farklı yol önermektedir: Global ve local (ye-

rel) yükleme seçenekleri. **Global Yükleme**: İsminden de belli olduğu gibi ilgili paketin tüm sistem üzerinde erişebileceğimiz şekilde yükleneceğini belirtmektedir. Yani yükleyeceğimiz paketlerin **"her zaman işimize yarayacak proje bağımsız"** olmasına dikkat etmemiz gerekmektedir.

```
npm install -g grunt-cli
```

Yukarıdaki install komutu ile global olarak kurulmasını sağlayan komut "-g" flag'idir. **Local Yükleme**: Proje bazlı kullanılmak üzere kullanacağımız modülleri yüklemek için "-g" flag'ini kullanmadan yükleme yapmamız yeterlidir.

```
npm install express
```

Yukarıdaki örnekte Express Framework modülü local olarak proje içerisinde kurulacaktır.

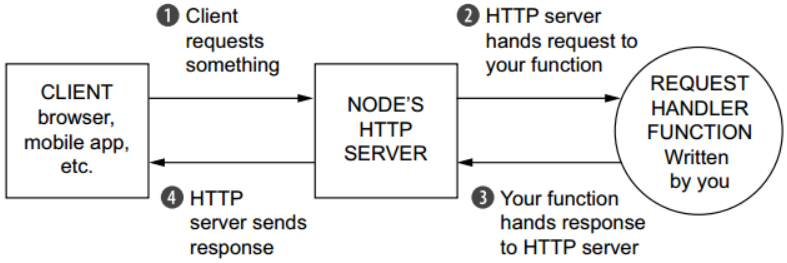
2.5 ExpressJS

ExpressJS kullanıcının Request'i ile Response arasındaki süreyi en kısa tutacak şekilde yapılandırılmıştır. Bu durumda hem performans olarak çok hızlı yanıt verecek hem de sistem kaynağı tüketimi minumum olacaktır. ExpressJS basittir, aslında basit olması beraberinde hızı da getirmiştir. Ayrıca mimarisi sizi uygulamlanın hangi alanı ile ilgilenirseniz ilgilenin hiçbir şeyi kullanmaya mecbur bırakmadan kendi modüllerinizi ve 3. parti modülleri kullanabileceğiniz şekilde tasarlanmıştır. Bu esneklikle size kafanızdaki uygulama mimarinizi oturtmaya imkan sağlar.

2.6 Middleware Kavramı

Middleware kavramının tam bir tanımı olmamakla birlikte genel olarak; geliştirici ile sistem arasında bulunan katmandır. Sistem derken ya işletim sistemi ya da Node altyapısı olan sistemi kasteder. Daha spesifik olarak middleware katmanı kendisini uygulama ile

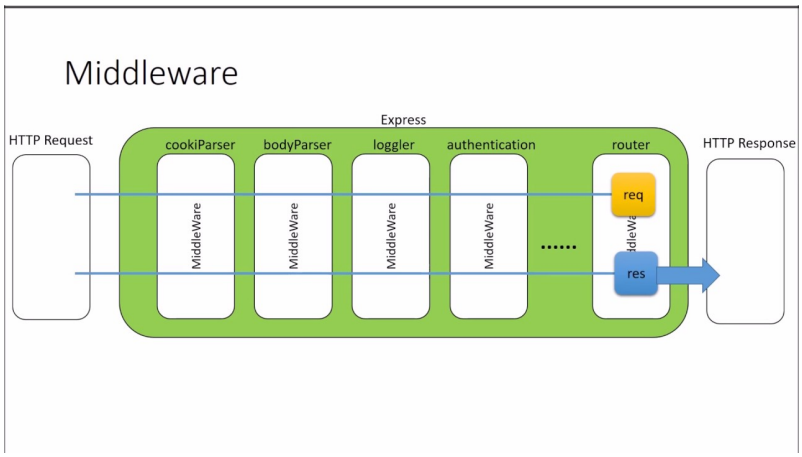
sistem arasında iletişim katmanı olarak atamaktadır. Middleware katmanı olmadan uygulama geliştirirsek tüm request işlemi tek ve uzun bir request handler ile yönetilecektir. Middleware yapısı sayesinde request handler parçalara bölünüp, her parçada farklı bir iş yapmamızı sağlar. Middleware kullanarak yapabileceklerimiz;authorization, proxies, routers, cookie ve session yönetimi. Örneğin ilk middleware yapısı log işlemi olsun ve tüm istekler console üzerinden görüntülensin. Bir sonraki middleware olarak authentication olsun ve yetkisi olmayan kullanıcılar daha sonraki middleware katmanına geçemesin.



Şekil 1: *NodeJs Middleware Yapısı*

Normal node yapısında middleware yapısı Şekil 1'deki gibi işlemektedir. Client istek yapar. Node http server bu isteği yazdığımız handler function yapısına iletir. Handler içerisinde istek işlenir ve http server'a tekrar yollanır.

Express kullanırken middleware fonksiyonlarını kendimiz yazabiliriz veya hazırda bulunan geliştiriciler tarafından yazılımış açık kaynaklı module yapılarını kullanabiliriz. Aşağıdaki yapı express middleware yapısını göstermektedir.



Şekil 2: *ExpressJS Middleware Yapısı*

ExpressJs Middleware yapısında ise Şekil 2'de gösterildiği gibi node http server üzerinden giden istekler stack yapısında toplanır. Stack yapısı içerisinde istek middleware fonksiyonlarını teker teker işleyip sonunda response olarak http server'a kullanıcı isteğinin cevabını döndürür. Stack aşamasında middleware yapıları işlenirken herhangi bir hata oluştuğu durumda işlem sırası kesilip hata middleware yapısına geçilmektedir. Kısaca özetlersek express frameworkte gelen istek stack yapısı üzerinden yukarıdan aşağıya doğru tüm middleware işlemlerinden geçer.

Middleware fonksiyonları 3 parametre almaktadır. Request ve response sürekli bulunmalıdır. Bunlar: function logger(request,response, next). Hata middleware yapısı ise 4 parametre alır. err, request, response, next

Örnek verecek olursak; logger yaparken, console.log ile basit bir logger yapısı yaparız. Eğer logger işlemini gerçekleştirmek için hazırlanmış "morgan" adı verilen module yapısını kullanırsak işlemimizi kısaltırız. Sonuçta tekerleği yeniden icat etmemize gerek yok hazır geliştirilmiş modülleri kullanıp yapacağımız işe odaklanmak daha mantıklıdır.

Express.js, middleware yapısını kendi sitesinde basitçe anlatmış.Şekil 3'i kısaca özetleyecek olursak. Önceden yüklenmiş(npm install express -save) express modulünü ekliyor ve app değişkenine express uygulamasını tanımlıyor.

```
var express = require('express')
var app = express()

var myLogger = function (req, res, next) {
  console.log('LOGGED')
  next()
}

app.use(myLogger)

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000)
```

Şekil 3: *Express.js Kullanımı*

Mylogger ile basitçe log yapısı oluşturup next() metodu ile bir sonraki yapıyı çağırıyor. app.use(myLogger) ile stack yapısına ekleme işlemi yapılıyor. Ardından routing işlemi

yapılarak "/" yani açılışta gösterilecek ilk ekrana "hello world" yazdırıyor.

2.7 Uygulamada Kullanılan Node Modülleri

2.7.1 Morgan

Uygulama içerisindeki HTTP request (isteklerini) loglamak için kullanılan bir modüldür. Bu modülle uygulamamıza gelen istekleri izleyebiliriz. Çalışma mekanızması middleware şeklindedir (diğer pek çok modülde olduğu gibi). Gelen request morgan üzerinden geçerken loglanır.

2.7.2 Body-Parser

Bu modül isminden de anlaşılacağı üzere uygulamıza gelen request'lerin body'lerinin kullanılmak üzere parse edilmesini (ayrıştırılmasını) sağlamaktadır. JSON, Text, Raw ve URL-enceded-form body'lerini parse edebilmekte, multipart request'leri ise parse ede**me**mektedir.

2.7.3 Nodemon

Front-end dünyasında zaten çok sık kullanılan bu teknolojinin, Back-end tarafındaki güzel bir temsilcisidir. NodeJS ile uygulamalar geliştirirken aynı diğer backend teknolojilerinde olduğu gibi kodların üzerinde değişiklik yaptığımızda sunumuzu tekrardan çalıştırmamız gerekiyor. **Nodemon** çalışma ortamımızda dosyaları sürekli izleyerek dosyalardaki değişiklik durumunda Node sunucusunu yeniden başlatmakta böylede yaptığımız değişikliği hemen test edebilmemizi sağlamaktadır.

2.7.4 Lodash

Lodash'ı pek çoğumuz biliyoruz aslında. Front-end uygulamalarının bir çoğunda JavaScrpit kodlarımızın pek çok işlemi bizim yerimize performanslı olarak yapan kütüphanedir.

Yapımı aşamasında esinlenildiği Undurscore kütüphanesine ek, pek çok fonksiyon ve performans üstünlüğü eklemiştir. Özellikle diziler (array) ve nesneler (object) üstüne eğilse de stringler gibi diğer konularda da gayet iyi çözümler sunuyor.

2.7.5 Cookie-Parser

Request ile gelen Cookie'leri okumak ve deneceğimiz response'a cookie set etmek için kullanılan modüldür. Uygulama seviyesinde middleware olarak eklenir.

2.7.6 Passport

NodeJS için en popüler **Authentication** modülüdür. Kullanıcıların kullanıcı adı şifreleri veya **OAuth** (Open Protocol) ile 3. parti (Facebook, Twitter, Google) hesapları ile authentication yapmasını sağlamaktadır. Authentication işlemleri uygulamanın akış şemasını ve davranışlarını değiştirebilecek bir yapıdır. Dolayısıyla Passport'u, Log gibi yerel olarak sadece uygulamanın belli bir yerine eklenebilecek bir middleware yerine uygulama genelinde kullanabileceğimiz bir yapı olarak düşünebiliriz.

2.7.7 dotenv

Dotenv, ortam değişkenlerini bir .env dosyasından process.env dosyasına yükleyen sıfır bağımlılık bir modüldür. Yapılandırmayı koddan ayrı bir ortamda saklamak için The Twelve-Factor App metodolojisine dayanır.

2.7.8 Mongoose

MongoDB ile Node uygulamalarını yapabilmemiz için kullanacağımız modülün ismi Mongoose'dur. NodeJS dünyasında oldukça ünlü olan bu modül yaklaşık 8.500 yıldıza sahiptir. Birçok back-end teknolojisinde Relational veritabanları için **ORM** olarak tanımlanan Object Relational Mapping'in Mongo tarafındaki karşılığı olarak düşünülebilir.

2.7.9 PM2 Modülü

Geliştirmenizi tamamladınız ama üretim ortamında uygulamanızı çalıştırmanız ve izlemeniz gerekiyor. İşte PM2 burada devreye giriyor ve size hataları log'lama, uygulamanız öldüğünde tekrar çalıştırma, kullandığı kaynakları izleme gibi imkanlar sunuyor. PM2 uygulama process'lerinin sürekliliğini sağlamak için kullanılan process manager'dir.

2.8 Express Generator

Express Generator projesiyle uygulamanın ilk ayağa kalkması esnasında ihtiyacınız olacak yapı ve minik kod blokları eklenmiş şekilde size bir iskelet sunuyor. Böylece bu iskeleti oluşturmak için harcayacağınız zaman da size kalıyor.

Öncelikle **Express generator** isimli paketimizi kuruyoruz.

```
npm install express-generator -g
```

Sonrasında geliştireceğmiz uygulamanın ismiyle

```
express mustafaApp
```

yazmamız yeterli, Express-generator bizim için uygulamayı oluşturacaktır.

2.9 MongoDB

MongoDB, ilk yayınlandığı 2009'dan bu yana sadece NodeJS değil pek çok back-end teknolojisi ile bir arada kullanılan veritabanı çözümüdür. MongoDB'ye geçmeden önce biraz geleneksel ilişkisel veritabanı mimarilerini düşünelim. Birçok tablonun olduğu ve bu tabloların içerisine gelecek verilerin **absolute** (kesin) bir şekilde belirlendiği, sonrasında veritabanının verileri alırken de bu farklı tablolar arasında ilişkiler kurularak verilerin

alındığını görüyoruz. Örnek olarak bir E-Ticaret sitesi düşünelim ve bu web uygulamasının back-end tarafından geleneksel veritabanı sistemleri ile yapılmış olsun. Bir satın alma olayından sonra kullanının adını ve ürünlerini ekrana yazdıracağız. Bunun için minimum iki tabloya ihtiyacımız olacak. Şöyle ki;

1. Kullanıcının tüm bilgilerini içinde barındıran kullanıcı tablosu (isim, yaş, telefon, müşteri numarası vs.
2. Ve sipariş detaylarını içinde barındıran Order tablosu (Sipariş numarası, sipariş saati, müşteri numarası vs

Yukarıdaki örnekte **User** tablosundan müşteri numarası üzerinden ilişki kurduğumuz **Order** tablosundan verileri çekebiliyoruz. Bu durumda iki tablo arasındaki ilişkiyi **customer_id** isimli yani müşteri numaranızı tutan **foreign key** ile sağladığımızı varsayalım. Bu basit işlemde iki tablodan verileri aldık. Peki listeleyeceğimiz verilerde satın alınan ürünlerde olsaydı? Bu durumda yine database'in yapısına göre en az 1 tabloya daha ihtiyacımız olacaktı. Geleneksel mimaride işler bazen karmaşıklık durumuna göre birçok tablo girdiği için zorlaşabiliyor. Bu durumda yıllar içerisinde **NoSQL** olarak tanımlanan bu geleneksel yaklaşımdan ayrılmış çözümler ortaya çıktı. **MongoDb**'de belkide bunların en popüleridir. **NoSQL** sistemler ile verilerimizi anahtar değer, **document oriented** gibi farklı şekillerden SQL'in katı yapısına bağlı kalmadan saklayabiliriz.

MongoDB'nin Özellikleri şu şekildedir

1. Ölçeklenebilirdir (Scalable). Veri boyutu arttığı durumlarda veya performans sıkıntısı yaşadığımız durumlarda makine ekleyebiliriz
2. Veriler document (belge) biçiminde saklanır. Burada JSON verilerini kullanabiliriz
3. Veriler JSON şeklinde saklandığı için gelen veri yapısı değişse bile kaydetme işleminde sıkıntı yaşanmaz.
4. Verilerin birden fazla kopyası saklanabilir ve veri kaybı yaşanmaz (Replication)
5. Veriler üzerinde index oluşturarak verilere hızlı bir biçimde ulaşabiliriz

2.10 Redis

Redis, data structure'larımızı saklamak için kullandığımız bir teknolojidir. Redis in memory olarak çalışır, yani verileri bellekte tutmakta ve bu durum onu performans anlamında çok öne çıkarmaktadır. Kullanım alanlarını incelediğimizde çok fazla data'nın aktığı ve performans beklentisi gerektiren işler olduğunu da görüyoruz. Session, Log vs...
Temel yapısı <Key,Value> şeklinde olan Redis verileri String, Hash, Set, Sorted Set ve Sıralı List şeklinde tutar.

Avantajları

1. CPU kullanımını azaltır.
2. Performans artışı sağlar.
3. IO işlemini azaltır.
4. Veriye ulaşımı en basite indirir.
5. Açık kaynak kodlu olması büyük bir avantaj.
6. Birçok popüler yazılım dilini desteklemektedir.
7. Komutları kolay ve dökümante edilmiştir.
8. Birçok veri türünü desteklemektedir.
9. Senkron çalışmaktadır.
10. Cluster Sharing, Sentinel, Replication gibi birçok enterprise özelliklere sahiptir.

Dezavantajları

1. Veri boyutu ile doğru orantılı olarak RAM ihtiyacınız artar.
2. İlişkisel veritabanlarında olduğu gibi karmaşık sorguları desteklemez.
3. Join Mantığı yoktur.

4. Transaction desteği yoktur.

5. Veri güvenliği için bir kontrol mekanizması yoktur.

2.11 Socket.IO

Gerçek zamanlı bir uygulama dendiğinde akla sunucudaki bir değişikliğin anında istemci tarafında, istemci tarafındaki bir değişikliğin anında sunucu tarafında değişikliğe sahip olması olarak özetleyebiliriz. Son zamanlarda Web uygulamalarında popülerliği ve kullanım alanı giderek artsa da (Sosyal medya uygulamalarındaki chat implementasyonları gibi) aslında neredeyse internetin hayatımıza girdiği ilk zamanlardan beri teknolojik yapısını ne olursa olsun kullanmaktayız(IRC, ICQ gibi). Ancak zaman içerisinde kullanıcı tarafında pratikte kullanımı değişmese de altyapı olarak farklılaştığını söyleyebiliriz. Öncelikle Polling yöntemiyle başlayalım. Döviz kularının canlı olarak gösterildiği bir web sayfaası yaptığımızı düşünelim. Bu web sayfası içerisindeki rakamların sürekli olarak sunu tarafında güncellendiğini varsayalım. clilent tarafında sunucuya belli aralıklarla AJAX istek gönderiyoruz. Eğer değişiklik varsa View'imizi yeni datalarla güncelliyoruz. Ancak **Polling** yönteminin çeşitli dezavantajaları var. Tek yönlü çalışıyor. sunucudaki değişikliği sunucuya sormadan öğrenemiyoruz. Sürekli olarak kontrol amaçlı sunucuya request gittiği için fazladan sistem ve veriyolu kaynağı harcanıyor. Peki şimdi altyapıyı nasıl sağlıyoruz? Günümüzde tüm modern browser'larda W3C tarafından standart hale getirilmiş bir teknoloji olan Websocket protokolü kullanılmaktadır. Bu teknoloji Two way commutication sağlar. Yani sunucudaki bir değişiklik istemci tarafına, istemci tarafındaki bir değişiklik sunucu tarafına **PUSH** yapılarak iletilir. Veirlerin Push ile akması, yani bir tarafın diğer tarafa güncllleme var mı diye sormadan eğer güncelleme varsa direkt olarak datalarla bildirmesi sistem kaynağı ve veriyolu anlmaında avantaj sağlamaktadır. Doğal olarak Polling metoduna göre performans olarak hızlıdır (full dublex communication).

Websocket ile Socket.io arasında nasıl bir ilişki var? Websocket'in bu kadar gelişmiş bir yapısı olmasına rağmen direkt olarak kullanmanın bazı sakıncaları var. Bunlardan biri HTML5 standardı olduğu için implementasyonunun sadece modern browser'lara yapılıyor olması. Geliştirdiğimiz uygulamaların tüm kullanıcılar tarafından kullanılmasını

istiyorsak, Websocket kullandığımızda eski tarayıcıya sahip kullanıclar kullanamayacaktır. Socket.io, aslında Websocket üzerinde çalışan bir teknolojidir. Geliştirici ile Websocket arasında Abstraction sağlayarak eğer kullandığı tarayıcı Websocket desteklemiyorsa alternatif yöntem üzerinden yine Real time haberleşmeyi sağlayabilmektedir. Socket.io, client tarafının socket bağlantısını açmak istediği ilk request'te Websocket kullanıp kullanamayacağını kontrol ederek server ile bağlantısını configure etmektedir. Socket.io ile hem yapısal hem de sonradan oluşturacağımız custom event'leri gönderip alabiliyoruz.

2.12 AngularJS

AngularJS, dinamik web uygulamaları için yapısal bir frameworktür. HTML'i şablon dili olarak kullanmanızı sağlar ve uygulamanın bileşenlerini açık bir şekilde ifade etmek için HTML sözdizimini genişletmenize izin verir. Angular'ın veri bağlama ve bağımlılık enjeksiyonu, aksi takdirde yazmak zorunda kalacağınız kodun çoğunu ortadan kaldırır. Hepsi tarayıcıda olur ve herhangi bir sunucu teknolojisi ile ideal bir ortaklık yapar.

AngularJS Özellikleri

1. AngularJS, Rich İnternet Uygulaması (RIA) oluşturmak için güçlü bir JavaScript tabanlı geliştirme çerçevesidir (framework).

2. AngularJS, temiz bir MVC (Model View Controller) yöntemiyle istemci tarafı uygulaması (JavaScript kullanarak) yazmak için geliştiriciler sağlar.

3. AngularJS ile yazılmış uygulama çapraz (Cross) tarayıcı uyumludur.

4. AngularJS otomatik olarak her tarayıcı için uygun JavaScript kodunu işler.

5. AngularJS açık kaynak kodlu, tamamen ücretsiz ve dünyadaki binlerce geliştirici tarafından kullanılmaktadır. Apache Lisansı sürüm 2.0 kapsamında lisanslanmıştır.

6. Genel olarak, AngularJS, bakımı kolay bir şekilde tutarak büyük ölçekli ve yüksek performanslı bir web uygulaması oluşturmak için bir çerçevedir.

AngularJS Temel Özellikleri

Veri bağlama - Model ve görünüm bileşenleri arasında otomatik olarak veri senkronizasyonu.

Kapsam - Bunlar modele referans nesneleri. Denetleyici ve görüntü arasında tutkal gibi davranırlar.

Denetleyici - Bunlar belirli bir kapsama bağlı olan JavaScript işlevleridir.

Hizmetler - AngularJS, XMLHttpRequests oluşturmak için $ https: gibi çeşitli yerleşik hizmetler ile birlikte gelir. Bunlar, yalnızca bir kez uygulamada örneklendirilen tek nesnelerdir.

Filtreler - Bunlar, bir dizideki öğelerin alt kümesini seçer ve yeni bir dizi döndürür.

Yönergeler - Yönergeler, DOM öğelerindeki işaretleyicidir (öğeler, özellikler, css ve daha fazlası gibi). Bunlar, yeni ve özel widget'lar olarak görev yapan özel HTML etiketleri oluşturmak için kullanılabilir. AngularJS yerleşik yönergelere sahiptir (ngBind, ngModel ...)

Şablonlar - Bunlar, denetleyici ve modelden gelen bilgi içeren işlenmiş görünümdür. Bunlar, tek bir dosya (index.html gibi) veya "partials" kullanarak bir sayfada birden fazla görünüm olabilir.

Yönlendirme - Görüş değiştirme kavramı.

Model Görünümü - MVC, bir uygulamayı farklı kısımlara (Model, Görünüm ve Denetleyici olarak adlandırılır) bölmek için bir desen kalıbıdır ve bunların her biri farklı sorumluluklara sahiptir. AngularJS MVC'yi geleneksel anlamda uygulamıyor, aksine MVVM'ye (Model-View-ViewModel) daha yakındır. Açısal JS takımı, mizahi bir şekilde Model Görünümü olarak bahsetmektedir.

Derin Bağlantı - Derin bağlantı, URL'de yer alan uygulama durumunu kodlayarak yer imi eklemenizi sağlar. Uygulama daha sonra URL'den aynı duruma geri yüklenebilir.

Bağımlılık Enjeksiyonu - AngularJS, uygulamanın geliştirilmesini, anlaşılmasını ve test edilmesini kolaylaştırarak geliştiriciye yardımcı olan yerleşik bağımlılık enjeksiyonu alt sistemine sahiptir.

AngularJS Avantajları

1. AngularJS, Tek Sayfa Uygulaması'nı çok temiz ve bakımlı bir şekilde yaratma olanağı sağlar.
2. AngularJS, HTML'ye veri bağlama yeteneği sağlar ve böylece kullanıcıya zengin ve duyarlı bir deneyim kazandırır
3. AngularJS kodu birim test edilebilir.
4. AngularJS ile geliştirici daha az kod yazar ve daha fazla işlevsellik elde eder.
5. AngularJS'de, görünümler saf html sayfalardır ve JavaScript ile yazılmış kontrolörler işlemlerini yapar.

3 Projeyi Canlıya Alma Etme

Bu bölümde yapılan projeyi nasıl canlıya alırız, canlıya alma işlemleri yapılırken neler yapılmalıdır, hangi plaform bizim projemiz için daha doğru bir yaklaşım olacaktır ve Git, github nedir projeler için neden önemlidir bu konular üzerinde durulacaktır.

3.1 Git nedir?

Şüphesiz son yıllardaki en popüler versiyon kontrol sistemi Git'dir. Projeler geliştirirken ister tek ister ekip olarak çalışalım, kodlarımızın sağlıklı bir şekilde tutulması, sürümlerin yönetilmesi için en önemli görev Git'e düşmektedir.

Genel olarak diğer **VCS** (Versiyon Kontrol Sistemi)'e göre avantajları aşağıdaki gibi sıralayabiliriz.

1. Dağıtık sistemli olmalarından ötürü geçmişte **SVN**(Subversion) ile yaşadığımız kod kaybı gibi pek çok problemin önüne geçmektedir. Sunucunun çökmesü artık eskisi kadar dert değil. Çünkü herkeste tüm history'i içeren local bir kopyası var.

2. Artık yaşam alanlarımızın her noktasında internet olsa da projelerimizde git kullanıyorsak projemizin localde tutulduğu için internet bağlantısına ihtiyaç yoktur.

3. Hızlıdır. Yüzlerce MB'lık projeleri bile oldukça rahat bu sisteme dahil edebilirsiniz.

4. Birçok geliştiricinin bulunduğu projelerde merge işlemi sancılıdır. Gir diğer VCS'lere nazaran merge işleminde daha başarılı.

5. Basit işlemleri öğrenmesi kolay ama sonraki aşamada ileri seviyeye çıkmak o kadar kolay değil sırf Git'in akışı için birçok pattern geliştirilmiş durumda.

6. Braching mekanizması oldukça gelişmiştir.

7. Github gibi açık kaynak için tekel olan bir uygulamada kullanılıyor.

3.2 Sunucu Seçimi

Geliştirdiğimiz projeleri son kullanıcıya iletmek için sunucu üzerinden yayınlamamız gerekiyor. İnternetin ilk zamanlarında bu sunucu üzerinde bir shared hosting kiralayıp kullanıcılara sunma yönündeydi. Shared hosting'de onlarca farklı uygulama aynı anda isminden de belli olduğu gibi 1 shared environment'de çalışmaktadır. Bu da beraberinde her ne kadar oldukça maliyetsiz bir çözüm yöntemi olsa da performans ve güvenlik sorunlarını doğurmaktadır. Godaddy gibi birçok firma bu konudaki ürünlerini sunmaya devam etmektedir. Ancak bu nokada artık internet kullanan kişi sayısının yıllar içerisinde artması ve buna bağlı olarak cloud mimarilerinin gelişmesi production'daki ürünlerin daha sağlam ve scale edilebilir sistemler üzerinde yayınlanmasını beraberinde getirdi. Şu anda en popüler iki mimari olan **Platform as a Service** (PaaS) ve **Infrastructure as a Service** (IaaS) kavramlarını inceleyelim.

Paas: Sunduğu yapıyla gelişticileri sistem bilgisine sahip olmasalar bile kendi yapıları üzerinde çalıştırmayı varsaymaktadır. İçerisinde barındırdığı pek çok servisle geliştirme süreçlerine katkı sağlamaktadırlar. Şunu kabul etmemiz gerekir ki sistem süreçleri, sistemin kaynaklarının yönetimi en az yazılım kadar geliştiriciyi zorlamaktadır. Nispeten daha kolay kullanım açısından Paas çözümler Iaas sistemlerden öne çıkarken, güvenlik ve esneklik ile shared hosting'lerden çok hada iyi konumdadır. Paas hizmeti veren en önemli firmalar aşağıdaki gibidir.

Heroku, Digital Ocean, EngineYard, OpenShift.

IaaS: Buluut bilişimiz en **low level** hizmeti olarak ifade edebiliriz. Hizmeti sunan firma tarafından kişiye açılacak sanal sunucunun yönetimini (ekstra hizmetlerde sunabilmekte) geliştirici tarafına bırakmaktadır. IaaS tipi yapıları kullanabilmek için sistem tarafında bilginiz olması gerekiyor. Bu hizmeti aldığımız firma sunucu üzerindeki hemen hemen herşeyin bizim tarafımızdan yapılabileceğini belirtiyor. Bunun çıktısı olarak da oldukça esnek, scale edebilen sistemler geliştirebiliyoruz. IaaS hizmeti veren en önemli firmalar AWS, Azure ve Rackspace'dir. Iaas ve Paas arasıdnaki genel farkları inceledikten sonra deploy işlemini ilk denediğim ama redis yapılandırmasını yapamadığım için yarıda kalan

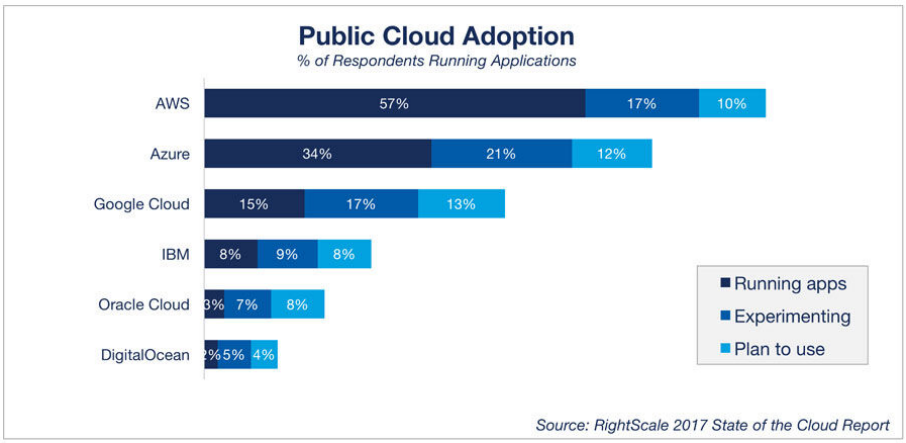
Heroku'dan size bahsetmek istiyorum.

3.2.1 Heroku

Heroku destek verdiği birçok dil ve kullanım kolaylığı ile öne çıkan **Platform as a Service** tipinde bir firmadır. Genelde küçük ve orta ölçekli projeler ya da prototip uygulamalar için tercih edilmektedir. En büyük özlliğinin kolay kullanımı ve hızlı deploy yapabilme imkanı tanıması olduğunu tekrar hatırlatayım. AWS gibi sistemler maalesef her geliştiriciye hitap etmeyebiliyor. Çünkü bu yapıları kullanabilmek için en azından orta ölçekli bir sistem bilgisine ihtiyaç bulunmaktadır. Bunun yanında Digital Ocean gibi ara çözümlerde bulunmaktadır. Bunun yanında **Heroku** eğer kısa ve orta vadede hızlı scale etmeniz gereken bir projenize varsa maliyet olarak biraz fazla yük getireceğini belirteyim. Şimdi sizlere kullanmış olduğum ve redis yapılandırmam dahil herhangi bir problem yaşamadığım Digital Ocean'dan bahsetmek istiyorum.

3.2.2 DigitalOcean nedir? Ne yapar?

DigitalOcean, bulut tabanlı altyapı sağlayıcısı olarak kendini konumlandıran, geliştirme, sürüm kontrolü ve test ortamları gibi bir çok ihtiyaca cevap veren dünyanın en büyük bulut sunucu sağlayıcılarından biri. Tabi, detaylara bakıldığında Şekil 4'deki markaların farklı sorunlara odaklandıkları da aşikar.



Şekil 4: *Genel Bulut Kullanımı*

Örneğin Amazon ile kesişen noktalar (Lightsail) dışında resmin tamamına baktığımızda Amazon rekabetinin çoğunlukla Google AppEngine ve Microsoft Azure tarafında yoğunluk kazandığını görebiliriz.

DigitalOcean geliştiricilere neler sunuyor?

1. Kullanım Kolaylığı

   Droplet adı altında ifade edilen bulut sunuculara Image yada app tercihi, kapasite ve bölge seçiminizin ardından saniyeler içerisinde 1 dakikadan az bir sürede sahip olabilir, oluşturduğunuz dropletleri pratik bir şekilde kontrol edebilirsiniz. Ayrıca, API üzerinden de droplet kontrolleri gerçekleştirebilmektesiniz.

2. **SSD Disk** Verileriniz performansı yüksek ve standart olarak sunulan SSD disklerde tutulmakta.

3. **Ücretlendirme Avantajı** Aylık minimumda $5 (saatlik $0.007)'dan başlayan fiyatlarla kullanıma başlayabilirsiniz. Droplet pasif olduğu durumlarda da veri barındırdığı için saatlik ücret işlemeye devam ediyor. Image alıp droplet'i kaldırarak test kullanımlarını çok daha efektif bir fiyatlandırmayla sürdürebilirsiniz.

4. **Distro Seçimleri** Oluşturacağınız droplet için Ubuntu, CentOS, Debian, Fedora, CoreOS gibi bir linux dağıtımlarının yanı sıra FreeBSD de seçebilirsiniz.

5. **Tek Tıkla App Kurulumu** LAMP, LEMP, MEAN, Django, Ghost, WordPress ve Docker gibi tek tıklama ile popüler bir çok uygulama kurulumunu hızlıca gerçekleştirebilirsiniz.

6. **Teknik Destek** Karşılaşabileceğiniz bir çok soruna yönelik olarak hazırlanmış oldukça kullanışlı bir içerik yığınına sahipler. Ek olarak, içerik dahilinde ulaşamadığınız çözümlere komünite üzerinden hızlı bir şekilde cevap alabilirsiniz. Hala çözümsüz kalmışsanız hızlı dönüş alabileceğiniz bir ticket oluşturabilirsiniz.

3.2.3 Droplet

Kullanımı kolay ve yeniden boyutlandırılabilir bulut sunucusudur (VPS).

1. Bir Linux dağıtımını, uygulamayı seçerek ya da önceden oluşturduğumuz snapshor ile Droplet oluşturabiliriz.

2. İhtiyacımız olan kaynaklara dayalı bir Droplet boyutunu seçip. Kontrol panelinden istediğimiz zaman yeniden **dikey** (vertical) boyutlandırabiliriz. Dikey boyutlandırma mevcut Droplet'in (sunucu) **CPU, RAM** veya **Diskinin ölçeklendirmesi** (scalability) anlamındadır. **Yatay** (Horizontal) ölçeklendirme ise; mevcut havuza daha fazla makine (VPS-Droplet) ekleyerek dinamik olarak ölçeklendirmektir.

3. Droplet'i dünya üzerinde bulunan farklı veri merkezlerinden (datacenter) birini seçerek oluşturabilirsiniz. Burda dikkat edilmesi gereken, bazı datacenter bölgelerinin (region) neleri destekleyip desteklemediğidir.

Özellikler:

- **Cluster Deployment:** Oluşturduğumuz her Droplet bir cluster (küme)üyesidir.
- **Resize:** İhtiyacınıza bağlı olarak Droplet'lerimizin (Droplets) kaynaklarını dikey olarak ölçeklendirin.
- **Yedekleme ve Görüntü Alma:** droplet oluştruma sırasında otomatik yedeklemeleri (backup) etkinleştirim veya istediğiniz zaman anlık görüntü (snapshot) alın.
- **İzleme:** Droplet'lerimizin bant genişliği, disk ve CpU seviyelerini yakından takip edin.
- **User Data:** İlk kurulum sırasında paketlerin yüklenmesini otomatikleştimek için özel komut dosyaları ekleyin.
- **User Data:** İlk kurulum sırasında paketlerin yüklenmesini otomatikleştirmek için özel komut dosyalrı ekleyin.
- **40GbE:** 40 Gigabit Ethernet (40GbE), Ethernet çerçevelerinin (frame) saniyede 40 gigabit'e (GbpS) kadar aktarımını sağlayan bir standarttır. 40GbE standardı yerel sunucu bağlantısı için tasarlanmıştır; Daha sağlam standart, 100 Gigabit Etherne

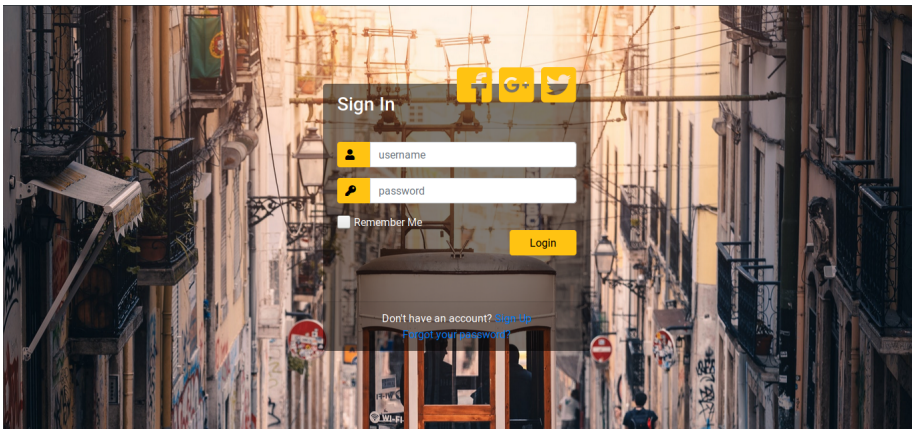
(100GbE), İnternet omurgalarına yöneliktir. **KVM:** Gelişmiş ağ performansı ve güvenliği için kurumsal düzeyde KVM. **Droplet arası haberleşme:** Dropletlerimiz birbiriyle private network olanağı ile haberleşebilir.

4 Web Sitesinin Görüntüleri

Bu bölümde yazılan projenin front-end görünümü paylaşılacak olup, giriş sayfası mesajlaşma sayfası ve projeye doğrudan girmenizi saylayacak kare kodu paylaşılacaktır muhtemelen artık o siteye erişimizin olamayacağını da belirtmek isterim.

4.1 Giriş Sayfası

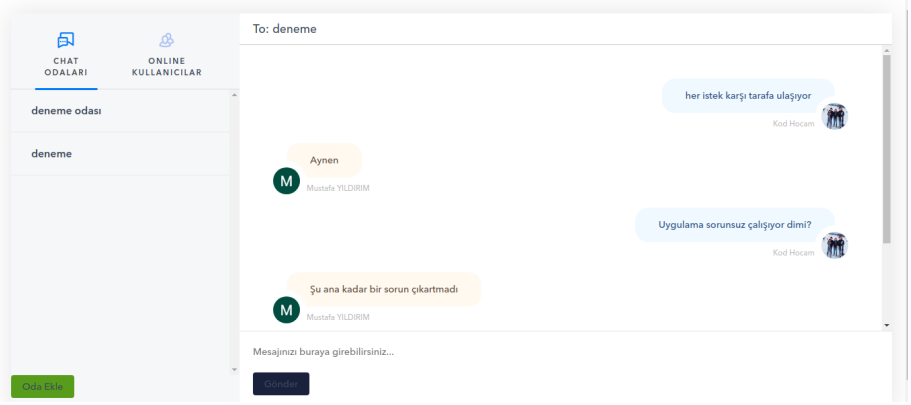
Projemize ait giriş sayfası Şekil 5'de göründüğü gibidir.



Şekil 5: *Kullanıcı Giriş Sayfası*

4.2 Mesajlaşma Sayfası

Projeye ait mesajlaşma sayfası Şekil 6'deki gibidir.



Şekil 6: *Mesajlaşma Sayfası*

4.3 Sitemi denemek ister misiniz?

Eğer sitemi denemek isterseniz Şekil 7'deki kare kodu taratarak ilgili sayfaya gidebilirsiniz.

Şekil 7: *Sayfayı açacak olan karekodu*

5 SONUÇLAR VE ÖNERİLER

Yapmış olduğum uygulamada gözlemlediklerime göre şu anda tüm sistemler sadece node.js ve NoSql yapıları kullanılarak oluşturulmuyor olsalarda önümüzdeki 4 - 5 sene içerisinde geleneksel format yavaş bir şekilde bırakılarak yerine yenilikçi metodlar gelecek gibi görünüyor. Uygulamayı Node.Js'le yazarken gözlemlerime göre php ile veya diğer geleneksel teknolojilerler uygulama yazarken ölçeklenebilirlik sorunu ile karşı karşıya kalıyoruz. Bu problemin üstesinden klasik (geleneksel) yöntemlerle de gelebiliriz ama bu bizim için daha maliyetli bir yaklaşım olur. Bunun yerine hızlı ölçeklenebilir bir yapı olan Node.Js kullanmak daha mantıklı ve maliyetsiz bir yaklaşım olacaktır. Bunlara ek olarak benim kullandığım yapının daha güzel bir şekilde kullanılmış hali bulunmakta buna **MEAN** diyorlar. Mongo, Express, Angular ve Node teknolojilerinin birleşmesiyle dinamik web uygulamarı yapmak için üretilen stack'leri anlatmak için kullanılan bir terimdir. Mean.io adresinden geliştirilmesi devam eden MEAN stack haricinde farklı MEAN Stack implemantasyon bulunmaktadır. MEAN stack kendisini bu ünlü 4 teknolojiyi buluşturan bir Framework olarak tanımlamaktadır. Bu yapının ortaya çıkmasındaki neden de yapılan yapılan back-end uygulamaları daha modern bir şekilde kullanıcıların önüne getirme ihtiyacıdır. Web uygulamarında view katmanı JavaScript ve UX kavramlarının gelişimi ile giderek farklılaşmaktadır. İşte bu noktada MEAN yapısı bize sağladığı back-end yapılandırmasının yanında Angular desteği ile kolaylık sağlamaktadır. Kısaca bu uygulama için kullandığım teknolojiyi yaptığım araştırmalardan yola çıkarak en doğru ve kesin sonuç olarak görüyorum. Server-client arasında sürekli ve çift yönlü etkileşim olan uygulamalarda kullanılmasını tavsiye ediyorum.

KAYNAKLAR

[1] CTAN,https://www.codeinwp.com/blog/angular-vs-vue-vs-react/ [Ziyaret Tarihi: 25 Mart 2019]

[2] CTAN,https://gelecegiyazanlar.turkcell.com.tr/konu/web-programlama/egitim/301-javascript/javascript-nedir [Ziyaret Tarihi: 29 Mart 2019]

[3] CTAN,https://www.mobilhanem.com/angular-dersleri-angular-nedir/ [Ziyaret Tarihi: 2 Nisan 2019]

[4] CTAN,https://kodcu.com/2013/04/nosql-kavrami-ve-mongodb/ [Ziyaret Tarihi: 12 Mayıs 2019]

[5] CTAN,https://nodejs.org/en/ [Ziyaret Tarihi: 14 Mart 2019]

[6] CTAN,https://angular.io/ [Ziyaret Tarihi: 15 Mart 2019]

[7] CTAN,https://expressjs.com/ [Ziyaret Tarihi: 12 Mayıs 2019]

[8] CTAN,https://redis.io/ [Ziyaret Tarihi: 12 Mayıs 2019]

[9] CTAN,https://www.mongodb.com/ [Ziyaret Tarihi: 12 Mayıs 2019]

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı :Mustafa YILDIRIM

Uyruğu : Türk

Doğum Yeri ve Tarihi: Üsküdar 12.05.1997

Adres :

Telefon : +905466678093

e-mail : musttafayildirim@gmail.com

EĞİTİM DURUMU

Lisans Öğrenimi : BŞEÜ Bilgisayar Mühendisliği Bölümü

Bitirme Yılı :

Lise : A.Ö.L

İŞ DENEYİMLERİ

Yıl :

Kurum :

Stajlar :

İLGİ ALANLARI:

YABANCI DİLLER:

BELİRTMEK İSTEDİĞİNİZ DİĞER ÖZELLİKLER: