

1.	Kriptolojiye Giriş	3
1.1.	Kullanılan Temel Kavramlar	3
1.2.	Kriptolojinin Gelişimi	5
1.3.	Kriptografik Algoritmalar	6
1.3.1.	Simetrik Şifreleme Algoritmaları	6
1.3.2.	Asimetrik Şifreleme Algoritmaları	6
2.	Yazılım ile İlgili Bazı Kavramlar	7
2.1.	Frontend-Backend Kavramı	7
2.2.	Javascript	8
2.3.	NodeJS	8
2.4.	NPM (Node Package Manager)	9
2.5.	MySQL	9
3.	Rest API ve JSON	10
3.1.	REST'in Özellikleri	11
3.1.1.	Stateless	11
3.1.2.	Uniform Interface	11
3.1.3.	Cacheable	12
3.1.4.	Client-Server	12
3.1.5.	Layered System	12
3.1.6.	Code On Demand	12
3.2.	URI	13
3.2.1.	URI Hiyerarşisi	13
3.3.	REST'te HTTP GET/POST Metotlarının Kullanımı	14
3.3.1.	GET :	14
3.3.2.	POST :	14
4.	NodeJS ve Socket.IO	15
4.1.	package.json	15
4.2.	server.js	17
4.3.	MySQL Veritabanı	18
4.4.	db.js	19
4.5.	helper.js	20

4.6.socket.js.....	23
4.7.routes.js.....	26
4.8.app.js.....	30
4.9.app.service.js.....	31
4.10.auth.controller.js.....	32
5. Ekran Görüntüleri.....	34
5.1.Kullanıcı Girişi.....	34
5.2.Kullanıcı Kayıt.....	35
5.3.Anlık Mesajlaşma Ekranı ve Online Kullanıcı Listesi	36
5.4.Android Platform Ekran Görüntüsü	37

BÖLÜM 1

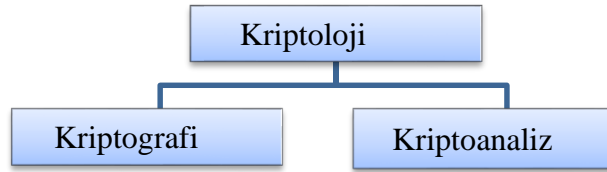
1. Kriptolojiye Giriş

Bilgisayarın keşfi ve internet kullanımının yaygınlaşması sonucunda geleneksel iletişim yerini elektronik iletişime bırakmıştır. Bunun sonucunda elektronik ortamlarda yapılan işlemler için güvenlik kavramı çok fazla önem kazanmaktadır.

Günümüzde çok sık kullandığımız Internet üzerinden yaptığımız haberleşmeler ve işlemlerin güvenliğini sağlamak için disiplinlerarası çalışmalarla geliştirilen birçok yöntemin birlikte kullanıldığı söylenebilir. Bu nedenle çok çeşitli şifreleme yöntemleri geliştirilmiştir. Bilgisayarların gücünün ve kapasitesinin artması ile bilgiler hızlı bir şekilde şifrelenerek iletilebilmektedir.

1.1.Kullanılan Temel Kavramlar

Kullanılan şifreleme algoritmalarının güvenilirliğinin test edilmesi de önemli bir konudur. Şekil 1’de görüldüğü gibi şifre bilimi yani Kriptoloji, Kriptografi ve Kriptoanaliz olmak üzere iki alt bilim dalına ayrılmaktadır.



Şekil 1. Kriptoloji bilimi alt bilim dalları

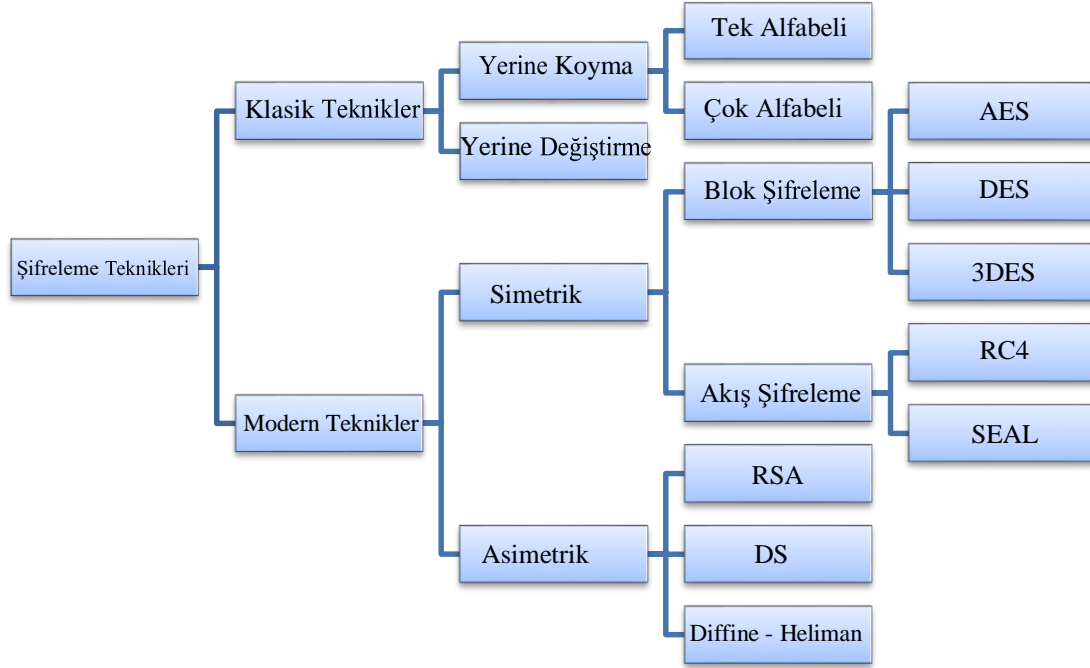
Burada kullanılan yöntemlerin birçoğunun dayandığı bir matematiksel tekniklerin bütününe kriptografi denir.

Kriptografi, bir bilginin istenmeyen taraflarca anlaşılmayacak bir hale dönüştürülmesinde kullanılan tekniklerin bütünü olarak açıklanabilir. Kriptografi gizlilik, bütünlük, kimlik denetimi, inkâr edememe gibi bilgi güvenliği kavramlarını sağlamak için çalışan matematiksel yöntemleri içermektedir

- Gizlilik : Bilgi istenmeyen kişiler tarafından anlaşılmamalıdır.
- Bütünlük : Bilginin iletimi sırasında hiç değiştirilmediği doğrulanmalıdır.
- Kimlik Denetimi : Gönderici ve alıcı birbirlerinin kimliklerini doğrulamalıdır.
- İnkâr Edememe : Gönderici bilgiyi gönderdiğini ve alıcı bilgiyi aldığını inkâr edememelidir.

Kriptoanaliz, şifrelenmiş yani anlamsız bir metinden doğru metni bulma yöntemidir. Kriptoloji ise kriptografi ve kriptanalizin birlikteliği için kullanılmaktadır. Başka bir deyişle Kriptoloji, haberleşmede veri güvenliğini sağlayan kripto cihazlarını, bu cihazlarda kullanılan algoritmaların güvenilirliğini araştıran, matematik bazlı elektrik ve elektronik mühendisliği, bilgisayar bilimleri, bilgisayar mühendisliği, istatistik ve fizik bölümlerini ilgilendiren disiplinlerarası bir alandır.

Gelişen teknoloji ile şifreleme için uygulanan yöntemler de değişkenlik göstermiştir. Kriptografi algoritmaları Şekil 2’de görüldüğü gibi klasik ve modern olmak üzere iki ana kategoriye ayrılmıştır.



Şekil 2. Şifreleme algoritmalarının sınıflandırılması

Geçmişte sadece askeri ve bazı ileri akademik alanlarda kullanılan klasik şifreleme yöntemleri algoritması gizli olan şifreleme yöntemlerini kapsamaktadır ve genellikle basit işlemlerle hesaplanabilecek algoritmalarından oluşmaktadır [4].

İlk klasik yöntemlerden biri olarak ENIGMA İkinci Dünya Savaşı döneminde kullanılmıştır. Sezar, Vigenère, Vernam, Playfair, Hill sistemleri klasik yöntemlerden bazılarıdır.

Metin şifrelenirken kullanılan dildeki harf sayısına göre şifreleme işlemi yapılır. Görüntü şifrelemede ise büyük/küçük harfler, nümerik rakamlar ve bazı operatörlerden oluşan 64 karakterlik bir alfabe (Base64) kullanılmaktadır (A-Z, a-z) (0-9)(+/-).

Görüntüler Base64 vasıtasıyla bir karakter dizisine dönüştürülür ve algoritmaya düz metin girdisi olarak verilir. Daha sonra şifreleme anahtarı ile girdi üzerinde algoritma çalıştırılır ve şifrelenmiş görüntüye karşılık gelen çıktı elde edilir.

1.2.Kriptolojinin Gelişimi

1970'lere kadar sadece askeri ve resmi kurumların kullandığı kriptografik yöntemler, 1976 yılında Diffie ve Hellman'ın önerdiği "Açık Anahtarlı Sistemler" kavramıyla bir devrim geçirmiştir. 1976 yılına kadar var olan şifre sistemlerinin güvenilirlikleri anahtarın gizliliğine dayanmaktaydı.

Gizli Anahtarlı Sistemler olarak adlandıracağımız bu sistemlerde, şifreleme ve şifre çözme işlemi için önceden belirlenen anahtarlar kullanılmakta ve şifre sistemlerinin de hep bu tür olabileceği düşünülmekteydi. Ancak, Açık Anahtarlı Sistemlerin keşfiyle aynı anahtarın hem alıcı hem de gönderici tarafından bilinmeden de güvenli haberleşmenin sağlanabileceği ortaya çıkmıştır. Ayrıca, Açık Anahtarlı Sistemler gizliliğin yanı sıra veri bütünlüğü, kimlik kanıtlama ve inkâr edememe konularına da çözüm getirerek birçok yeni uygulamaları da beraberinde getirmiştir.

Gizli Anahtarlı Sistemlerin işleyişinde en çok yer değiştirme ve karıştırma işlemleri kullanılmaktadır. Örneğin, en çok bilinen basit şifrelerden birisi olan Sezar şifresinin çalışma mantığı, şifrelenmek istenen harfin, kendisinden sonra gelen 3. harf ile yer değiştirilmesidir. Sezar şifresi ile ABC, ÇDE olarak şifrelenebilir.

Günümüzdeki hesaplama gücünün yüksek olması ve iletişimin bilgisayar ortamında yapılmasından ötürü, bu tip yer değiştirme ve karıştırma işlemleri şekil değiştirmiştir. Bu tip yer değiştirmeler için çeşitli matematiksel uygulamalar bulunmaktadır.

Örneğin, tüm dünyada kullanılan ve güvenilir olduğu bilinen AES (Advanced Encryption Standard) algoritması eski tip yer değiştirme işlemlerini kullanmasının yanı sıra, matematiksel yöntemleri de kullanmaktadır.

Gizli Anahtarlı Sistemler yer değiştirme ve karıştırma işlemlerini temel aldığından ötürü çok kısa sürede çok büyük boyutlardaki verileri şifreleyebilirler. Aynı hızda şifre çözme işlemi de gerçekleştirebilirler.

Sistemlerde, alıcı ve gönderici aynı anahtarı kullandığından, bu gizli anahtarın paylaşılması bir problemdir. Gizli anahtar öyle bir paylaşılmalıdır ki, sadece alıcı ve gönderici gizli anahtarın ne olduğunu bilsin. Gizli Anahtarlı Sistemlerdeki anahtar paylaşım problemine Açık Anahtarlı Sistemler ile çözüm gelmiştir.

Açık Anahtarlı Sistemlerde, açık (herkes tarafından bilinen) ve gizli (kişiye özel) anahtar olmak üzere iki çeşit anahtar kullanılmakta ve bu sayede, Açık Anahtarlı Sistemler ile herkesin birbirlerini tanımadan bile gizli bir şekilde haberleşmesi sağlanmaktadır.

İki farklı anahtarın kullanıldığı Açık Anahtarlı Sistemlerin çalışabilmesi için matematiksel problemlere ihtiyaç duyulmaktadır. Ayrıca, bu sistemlerin güvenilir olarak adlandırabilmesi için bu problemlerin çözümünün de zor olduğunun gösterilmesi gerek bunlara bağlı olarak günümüzde en çok kullanılan açık anahtarlı sistemlerden RSA çarpanlara ayırmanın zorluğuna, DSA (Digital Signature Algorithm) ve ECDSA (Elliptic Curve Digital Signature Algorithm) ise sonlu cisimler ve eliptik eğri üzerindeki ayrık logaritma probleminin zorluğuna dayanmaktadır.

1.3.Kriptografik Algoritmalar

Tüm modern algoritmalar şifreleme ve şifre çözme işlemlerini kontrol etmek için bir anahtar kullanırlar ve bir mesaj sadece kullanılan anahtar şifreleme anahtarıyla uyduğunda çözülebilir. Şifreleme süresince anahtarlı ya da anahtarsız olmak üzere iki farklı yöntem kullanılabilir. Özet (Hash) fonksiyonları, sıkıştırma fonksiyonları anahtarsız yöntemlere örnek olarak gösterilebilir. Anahtarlı kriptosistemler iki ana başlık altında incelenebilir.

Simetrik anahtarlı şifreleme (veya gizli-anahtarlı şifreleme)

Asimetrik şifreleme (veya açık-anahtarlı şifreleme)

1.3.1. Simetrik Şifreleme Algoritmaları

Simetrik şifreleme algoritmalarında mesajın şifrlenmesinde ve çözülmesinde tek bir gizli anahtar kullanılır. Şifreleme işlemlerini gerçekleştirdikten sonra şifreli metni alıcıya gönderirken şifreli metinle birlikte gizli anahtarın da alıcıya güvenli bir şekilde gönderilmesi gerekmektedir. Simetrik şifreleme algoritmaları çok hızlı şifreleme ve şifre çözme işlemleri gerçekleştirebildiğinden dolayı günümüzde çok yaygın olarak kullanılmaktadır.

Simetrik anahtarlı kriptosistemler, blok ve akan şifreleme olmak üzere ikiye ayrılır. Simetrik şifreleme algoritmaları şunlardır:

- i. Blok şifreleme algoritmaları (AES, DES, IDEA, Skipjack, RC5 ...)
- ii. Akan Şifreleme algoritmaları (RC2, RC4...)

1.3.2. Asimetrik Şifreleme Algoritmaları

Açık anahtarlı kriptosistemlerde ya da başka bir deyişle asitmerik şifrelemede her iki tarafın sahip olduğu açık (e) ve gizli (d) olarak adlandırılan bir anahtar çifti kullanılır. Şifreleme anahtarı olarak kullanılan e' nin gizli olması gerekmez. Açık anahtarlı şifrelemenin altında yatan temel düşünce açık anahtar, e, verildiği halde şifre çözme anahtarı olarak kullanılan d 'nin bulunmasının zor olmasıdır. Açık anahtarlı sistemler sayısal imza ve anahtar değişim protokolleri gibi uygulamalarda kullanılır.

Asimetrik şifreleme algoritmaları şunlardır:

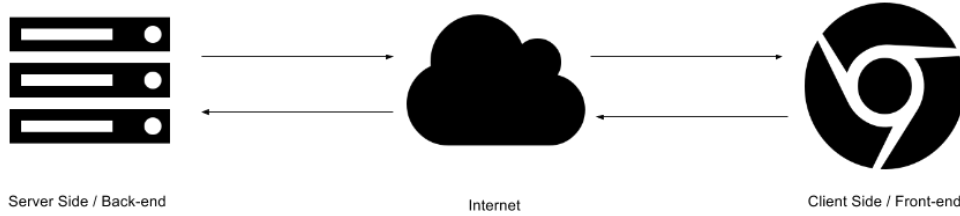
- i. RSA
- ii. El Gamal
- iii. Eliptik Eğri Sistemleri
- iv. Diffie-Hellman anahtar belirlemesi
- v. Kod-tabanlı Kriptosistemler

BÖLÜM 2

2. Yazılım ile İlgili Bazı Kavramlar

2.1. Frontend-Backend Kavramı

Çoklu kullanıcıli uygulamalar genelde Sunucu(Server)/İstemci(Client) mantığı ile yazılır. Sunucu yani çevrim içi çalışan, tüm verilerin tutulduğu merkezi bir sistemdir. İstemci yani bu merkezi sisteme bağlı çalışan merkezi sisteme veri iletimi yapan, kullanıcıyla etkileşimi sağlayan yan programdır.



Günümüz modern uygulamalar da bu istemci genelde tarayıcıdır(browser). Tarayıcı üzerinden kullanılan uygulamalar genelde Web Uygulaması olarak adlandırılır. Kullanıcı uygulamayı kullanmak istediğinde tarayıcıdan sunucuya bir istek gönderir, sunucu isteğe göre uygulama içerisinde kendi anlayacağı bölümleri işleyerek kullanıcıya uygun çıktıyı üretir ve tarayıcıya gönderir. Tarayıcı aldığı çıktıyı kendi içerisinde yorumlar ve kullanıcıya uygun çıktıyı üretir. Sonra kullanıcı gerektiği yerde tekrar sunucuya istek gönderir, tekrar cevap gelir. Genel olarak tarayıcı bazlı Sunucu/İstemci mimarisi basit anlatımla bu şekilde çalışır.

Burada önemli bir kısım sunucunun kendi anladığı kısımları yani sunucu tabanlı diller(C#, Java, Php, Python, Ruby...) ile yazılmış kısımları çalıştırması, tarayıcının da kendi anladığı yani tarayıcı dilleri(HTML,Css,Javascript, Dart...) ile yazılmış kısımları çalıştırmasıdır. Sunucu tarafında çalışan koda Backend, tarayıcı üzerinde çalışan koda Frontend denir. Web uygulamaları yazarken Backend ve Frontend kodları aynı proje içerisinde kullanılır. Hangi iş mantığı hangi kısımda çalışacaksa ona göre ya Backend'de yada Frontend'de yazılır.

2.2.Javascript



Frontend dillerinden en önemlilerinden bir tanesi Javascript'tir. Web tarayıcılarının etkin bir şekilde kullanılması için geliştirilmiştir. Kullanıcıyla etkileşimin artırılması, iş yükünün sunuculardan istemcilere taşınması, daha görsel animasyonlu web siteleri/web uygulamaları geliştirilmesi gibi kullanım alanları vardır. Javascript standartları Ecma International firması tarafından ECMAScript standartlarıyla belirlenir. Günümüzde kullanılan ECMAScript standardı 5.1 sürümüdür. Ve bu standart Javascript 1.8.1 ile desteklenmektedir. Ancak şu aralar ECMAScript 6 standardı üzerinde çalışmalar devam etmektedir.

Javascript prototip bazlı nesne yönelimli programlama, fonksiyonel programlama, imparator programlama paradigmasını destekler. Genellikle tarayıcılarda kullanılır. Google'ın Chrome tarayıcısını ve bu tarayıcı içerisindeki V8 Javascript motorunu geliştirmesiyle Javascript performansında ve gelişimde büyük yükselişler oldu. Bu Javascript motorunun gücü sayesinde daha önce yapılamayan birçok uygulama yapıldı. Ve ilk defa Javascript frontend'den çıkarak Backend tarafına geçiş yaptı ve Nodejs doğdu.

2.3.NodeJS



NodeJs 2009 yılında Joyent firmasında çalışan Ryan Dahl tarafından geliştirilen Javascript Çalışma Ortamıdır.(Javascript Runtime Environment). Joyent firmasının desteği ve V8 motorunun gücü ile Dahl Javascript'i Backend kısmına taşıdı.

“Nodejs backend tarafında çalışan, javascript tabanlı scriptleri yorumlanabilen ve çalıştırabilen, hızlı, ölçeklenebilir network uygulamaları geliştirmeyi sağlayan bir çalışma zamanı ortamıdır(runtime environment).”

Javascript'in backend'e(server side) taşınması ile developerlar tek dil bilerek hem frontend hem backend kod yazabilir hale geldiler. Frontend de kullanılan javascript kütüphanelerinden bazıları backend tarafında kullanılabilir hale geldi.

Javascript'in doğal Non Block mimarisi sayesinde backend tarafında ölçeklenebilir uygulamalar daha kolay bir şekilde yazılabilir hale geldi. Nodejs'in ölçeklenebilir uygulamalardaki başarısı sayesinde kullanıcı sayısı milyonlarla ifade edilen büyük siteler nodejs kullanmaya başladı. Üstelik bunu single thread olarak yapabiliyor.

“Ölçeklenebilirlik, bir uygulamanın aynı anda birden çok kullanıcıya takılmadan cevap verebilir halde olmasıdır.”

Peki bunu nasıl başarıyor? Nodejs olay bazlı(event-driven) Non-Blocking yapısını Event loop denilen bir mimariyle sağlıyor. Bu mimariyle tek bir thread(single thread) ile asenkron çalışabilmektedir.

2.4.NPM (Node Package Manager)

NPM (Node Package Manager / Node Packaged Modules), Isaac Z. Schlueter tarafından, tamamen JavaScript dili kullanılarak geliştirilen, temel olarak bir harici olarak sunulan yazılımların / paketlerin / modüllerin yönetimini sağlayan bir paket yöneticisidir.

Paket ve bağımlılık yöneticiler çerçevesinde temel özellikler,

- Otomatik ya da el yordamı ile projelere paketleri dahil etme / yükleme
- Edinilmiş paketleri silme
- Kullanılan paketleri kayıt altında tutma (package.json vb.) ve listeleme
- Kullanılan paketleri güncelleme

2.5.MySQL



MySQL bir ilişkisel veritabanı yönetim sistemidir.

Veritabanı yönetim sistemi veritabanlarını tanımlamak, yaratmak, kullanmak, değiştirmek ve veri tabanı sistemleri ile ilgili her türlü işletimsel gereksinimleri karşılamak için tasarlanmış sistem ve yazılımdır.

İlişkisel veritabanı ilişkisel veri tabanını çeşitli tablolar arasında organize edilmiş verilerden oluşan veri tabanı olarak açıklayabiliriz. Bu farklı tablolar arasındaki veriler, çeşitli anahtarlar vasıtası ile birbirlerine bağlanırlar. İlgili tablolarda, sütunlar arasında bir anahtar sütun yer alır. Bu anahtar sütun aracılığı ile birden çok tablo verileri birbiriyle bağlantı sağlayabilir ve herhangi bir sorgulamada birlikte görüntülenebilir.

MySQL çifte lisanslı bir yazılımdır. Yani hem Genel Kamu Lisansı'na (GPL) sahip özgür bir yazılım, hem de GPL'in kısıtladığı alanlarda kullanmak isteyenler için ayrı bir lisansa sahiptir.

BÖLÜM 3

3. Rest API ve JSON

Api (Application Programming Interface) bir uygulamaya ait işlevlerin başka bir uygulamada da kullanılabilmesi için oluşturulmuş bir arayüzdür. İki yazılımın veya veritabanının birbiri ile sorunsuz çalışabilmesini ve en sağlıklı bir şekilde birbiri ile iletişime geçmesini sağlar.

REST(Representational State Transfer) istemci-sunucu arasında hızlı ve kolay şekilde iletişim kurulmasını sağlayan bir servis yapısıdır. REST, servis yönelimli mimari üzerine oluşturulan yazılımlarda kullanılan bir veri transfer yöntemidir. HTTP üzerinde çalışır ve diğer alternatiflere göre daha basittir, minimum içerikle veri alıp gönderdiği için de daha hızlıdır. İstemci ve sunucu arasında XML veya JSON verilerini taşıyarak uygulamaların haberleşmesini sağlar. REST standartlarına uygun yazılan web servislerine RESTful servisler denir.

Bir örnekle açıklamaya çalışırsak, bir alışveriş sitesinde “Dell XPS” anahtar kelimeleriyle bir arama yaptınız ve karşınıza aradığınız kelimelere ilişkin ürünler ve her ürüne ait kapak fotoğrafı, başlık, fiyat gibi bilgiler çıktı. Bu bilgiler client’ın, yani sizin, server’a yaptığı istek sonucu edindiği bilgilerdir. Örneğimizdeki client-server iletişiminde Dell XPS dizüstü bilgisayarlar resource oluyor. Yani kullanıcıların göreceği, güncelleyeceği, sileceği kadar önemli olduğunu düşündüğünüz ve bu sebeple isim verdiğiniz kaynaklar (products). Ürünlerdeki fiyat bilgisi dönem dönem değişebilen bir değerdir. Yani ürün resource’unun state’i sabit değil. Fakat bize bu resource’un en güncel hali gelir. State yalnızca bununla sınırlı değildir. Örneğin Dell XPS 13 9380 resource’unun stok bilgisi de olabilir fakat bir kullanıcı olarak bizim bunu görmeye yetkimiz yoktur. Yalnızca görmeye yetkili olduğumuz bilgileri içeren transfer de bu resource’un bir state’idir.

Dell XPS 13 9380 başlıklı ürünün sayfasına gittiğimizde bizi o ürüne ait daha ayrıntılı bir sayfa karşılar. Teknik özellikler, fotoğraflar, taksit seçenekleri gibi birçok bilgiyi görürüz. Peki bu bilgiler bize hangi formatta geliyor? Ürünün ilgili sayfasına gittiğimizde bir REST isteği ile resource’un state’ini istiyoruz ve bize cevap olarak bir JSON dosyası geliyor.

Aşağıdaki JSON dosyası bir örnek olabilir:

```
{
  "id": 1,
  "title": "Dell XPS 13 9380 Intel Core i7 8565U 16GB 512GB SSD Windows 10 Pro 13.3 HD Taşınabilir Bilgisayar UT56WP165N",
  "brand": "Dell",
  "model": "XPS 13 9380",
  "images": [
    "http://placekitten.com/g/200/300",
    "http://placekitten.com/g/200/300",
    "http://placekitten.com/g/200/300"
  ]
}
```

Bu JSON dosyası da resource'un bir representation'ı. JSON olması zorunlu değil. XML veya başka bir şey de olabilir.

URI

<https://api.spotify.com/v1/artists/14r9dR01KeBLFFyIVSKCZQ>

RESOURCE

Damien Rice'in Spotify'daki Sanatçı Sayfası

REPRESENTATION

```
{
  "external_urls": {
    "spotify": "https://open.spotify.com/artist/14r9dR01KeBLFFyIVSKCZQ"
  },
  "followers": {
    "href": null,
    "total": 756452
  },
  "genres": [
    "acoustic pop",
    "folk-pop",
    "indie folk",
    ...
  ]
}
```

Basit bir REST iletişiminin üç temel aktörü

3.1.REST'in Özellikleri

3.1.1. Stateless

REST'in stateless olması server'ın client hakkında session gibi bilgileri tutmaması demektir. Bu gibi bilgileri yalnızca client tutar. Dolayısıyla server, istek yapan client'ın daha önce kaç istek yaptığı veya hangi istekleri yaptığı gibi bilgileri tutmaz. Client ise yaptığı istekte server'ın ihtiyaç duyduğu tüm bilgileri verir. REST stateless olduğu için monitoring aracı kullanıyorsanız ihtiyaç duyduğunuz tüm bilgiler ilgili isteğin içinde olacaktır. Geçmişe yönelik bir tarama yapmanız gerekmez (visibility). Her request arasında bir kayıt tutmak zorunluluğu olmadığı için kaynak tüketimi azdır ve mimarinin uygulanması daha kolaydır (scalability). Fakat aynı zamanda server, client'a ilişkin veri tutmadığı için client'ın her istekte bazı bilgileri göndermesi maliyeti artırır. Bu da stateless oluşunun dezavantajı olarak sayılabilir.

3.1.2. Uniform Interface

Client ve server arasındaki iletişim için belirlenmiş dört prensiple sağlanır: Her istekte resource'ları tanımlayan bir resource identifier kullanılır. Bunu sağlamak için URI standartları kullanılır.

Bildiğimiz üzere client, yaptığı istek sonucunda server'dan bir representation alır. Resource dediğimiz şey veritabanında bir row olabilir. Representation ise bunun temsidir.

Uniform interface kısıtına göre client sahip olduğu response ile o resource'u silecek, değiştirecek kadar bilgiye sahip olmalıdır.

İstekler kendilerini tanımlayıcı bilgileri barındırmalıdır. Örneğin gelen representation'ın hangi formatta geldiğini söyleyip client'ı hangi parser'ı kullanacağı yönünde bilgilendirir. (bkz. mime types)
Client'lar istek yaparken dört farklı yolla bilgi aktarır: İstek body'si, query-string parametreleri, header'lar ve URI. Server ise üç farklı yolla bunu yapar: response içeriği, response code ve response header.

3.1.3. Cacheable

Server, gönderdiği response'larda resource'un cacheable veya uncacheable olduğunu da söyler. Böylece client gönderilen bilgilere göre bir cache mekanizması oluşturabilir.

3.1.4. Client-Server

Yazılım tasarımında aşına olduğumuz bir kavram olan seperation of concerns prensibi REST tasarımında da uygulanır. Bu kısıta göre client, server'ın sorumluluğunda olan depolama işlemleri gibi şeylerle ilgilenmez. Aynı şekilde server da client'ın sorumluluğu olan user state gibi konularla ilgilenmez. Böylece server ve client iki farklı aktör olarak çalışır. Aralarındaki iletişim de client tarafından başlatılır. Server yalnızca client'tan gelen istekleri bekler. Bunun sonucunda client ve server birbirinden bağımsız olarak geliştirilir, server tarafında geliştirme basit ve ölçeklenebilirlik yüksek olur ve client tarafında ise kodun taşınabilirliği yüksek olur.

3.1.5. Layered System

Client-Server mimarisinden bahsederken kastımız her zaman bir client'ın doğrudan bir server'a istek göndermesi ve ondan doğrudan cevap alması şeklinde değildir. Aralarda güvenlik katmanı, cache katmanı gibi katmanlar olabilir. Böyle bir sistemde aralardaki katmanlar request ve response'a etki etmemeli. Her katman yalnızca iletişime geçtiği katmanları bilmeli.

3.1.6. Code On Demand

Zorunlu olmayan tek kısıt code on demand'dir. Code on demand kısıtı server'ın client'a belli durumlarda executable script'ler ve applet'ler gönderebilmesini kapsar.

3.2.URI

REST’te representation’ını sunduğumuz verilerin temel kaynağının resource olduğundan bahsetmiştik. REST API tasarımında resource’ları adreslemek için URI’ları kullanırız.

URI (Uniform Resource Identifier), client’in ulaşmak istediği representation’ı bir dizi kurala uyarak uniform bir linkle gösterir. Şimdi REST API’larda URI tasarımının nasıl olması gerektiğini inceleyelim.

3.2.1. URI Hiyerarşisi

API tasarımıımızda client, tüm resource’lara bir hiyerarşi ile ulaşmalı. Bu hiyerarşiyi sağlamak için forward slash(/) kullanıyoruz. Aşağıda bazı hiyerarşi örnekleri var:

- <https://ornek-api.com/api/v1/users/123456/media/audio>
- <https://api.ornek.com/posts/3456/comments/1>
- <https://api.deneme.com/users/123/friends/456>

URI’ı 123 ID’li user’ın 456 ID’li arkadaşını işaret eder. Bu URI tasarımında aşağıdaki URI’ların hepsi bir resource’u göstermeli.

- <https://api.deneme.com/users/123/friends/456>
- <https://api.deneme.com/users/123/friends>
- <https://api.deneme.com/users/123>
- <https://api.deneme.com/users>

`https` :// `wubbalubba.com` : `80` /`users/1/posts` ? `orderBy=asc` # `search-fragment`
scheme host port path query fragment

3.3.REST’te HTTP GET/POST Metotlarının Kullanımı

HTTP, web browser ile web server arasında iletişim kurmamızı sağlayan bir protokoldür.

HTTP 1.1 versiyonu (RFC 2616) ile tanımlanan ve diğer eklentilerle gelen başlıca HTTP metodları şunlardır:

3.3.1. GET :

Bu metod sunucudan veri almak için kullanılır. GET ve POST metodları en sık kullanılan metodlar olup sunucudaki kaynaklara erişmek için kullanılırlar.

GET metodu ile sorgu metinleri URL içinde gönderilebilir. Bunun en önemli faydası kullanıcıların bookmark edebilmeleri ve aynı sorguyu içeren istekleri daha sonra gönderebilmelerini sağlaması ve tarayıcıda önceki sorguların “geri” tuşu ile veya tarayıcı geçmişinden çağrılarak aynı sayfalara ulaşabilmeleridir.

Güvenlik açısından URL’lerin ekranda görüntüleniyor olması ve URL’in hedefine ulaşmaya kadar ve hedef sunucu üzerinde iz kayıtlarında görülebilmesi gönderilen parametrelerin gizlilik ihtiyacı varsa sıkıntı yaratabilir. Bu nedenlerle hassas isteklerin GET ile gönderilmemelidir.

3.3.2. POST :

Bu metod ile sunucuya veri yazdırabilirsiniz. Bu metodla istek parametreleri hem URL içinde hem de mesaj gövdesinde gönderilebilir. Sadece mesaj gövdesinin kullanımı yukarıda sayılan riskleri engelleyecektir. Tarayıcılar geri butonuna basıldığında POST isteğinin mesaj gövdesinde yer alan parametreleri tekrar göndermek isteyip istemedimizi sorarlar. Bunun temel nedeni bir işlemi yanlışlıkla birden fazla yapmayı engellemektir. Bu özellik ve de güvenlik gerekçeleriyle bir işlem gerçekleştirileceğinde POST metodunun kullanılması önerilir.

BÖLÜM 4

4. NodeJS ve Socket.IO

4.1.package.json

Bir Node.js projesi yazarken projemizin içerisinde mutlaka bir package.json dosyamız olur. Bu dosyanın içerisinde proje ile ilgili zorunlu-zorunlu olmayan bazı bilgileri içeren bir dosya bulunmalıdır.

```
{
  "name": "chatapp_using_nodejs_mysql",
  "version": "1.0.0",
  "description": "This is a private chat app in nodejs using Mysql as database",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Ecem Okan",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "crypto": "^1.0.1",
    "express": "^4.17.1",
    "mysql": "^2.17.1",
    "socket.io": "^2.3.0"
  }
}
```

name : Proje için bir isim giriyoruz. Küçük harflerle olmalı. Zorunlu alandır.

version : Semantic Versioning denen bir versiyon tanımlama sistemiyle düzenlenir.

Zorunlu alandır. 3 kısımdan oluşur.

Major, Farklı bir API değişikliği yaptığında.

Minor, Geriye dönük bir yolda fonksiyonellik eklediğinizde.

Patch, Geriye dönük bug düzeltmesi yaptığınızda.

Description : Projenin tanımıdır.

Author : Projenin yazarı/yazarının tanımlandığı yerdir.

Dependencies : Projemizde kullandığımız paketleri kullanmamız/indirmemiz için ihtiyacımız olan objedir. Eğer kullanacağımız projenin ihtiyaç duyduğu paketler sistemde global olarak yüklü ise ayrıyeten projeye indirmemize gerek yok. Fakat çok küçük olmayan bir proje yapıyorsanız fazla pakete ihtiyacınız olacağından hepsini global olarak indirmek istemezsiniz. Dolayısıyla buna ihtiyacınız olacak.

Şuana kadar projemizde package.json ve node_modules klasörü oluşturuldu. Uygulamayı çalıştırmak için gerekli olan proje dizinleri ve dosyaları Resim 1.1'deki gibidir.

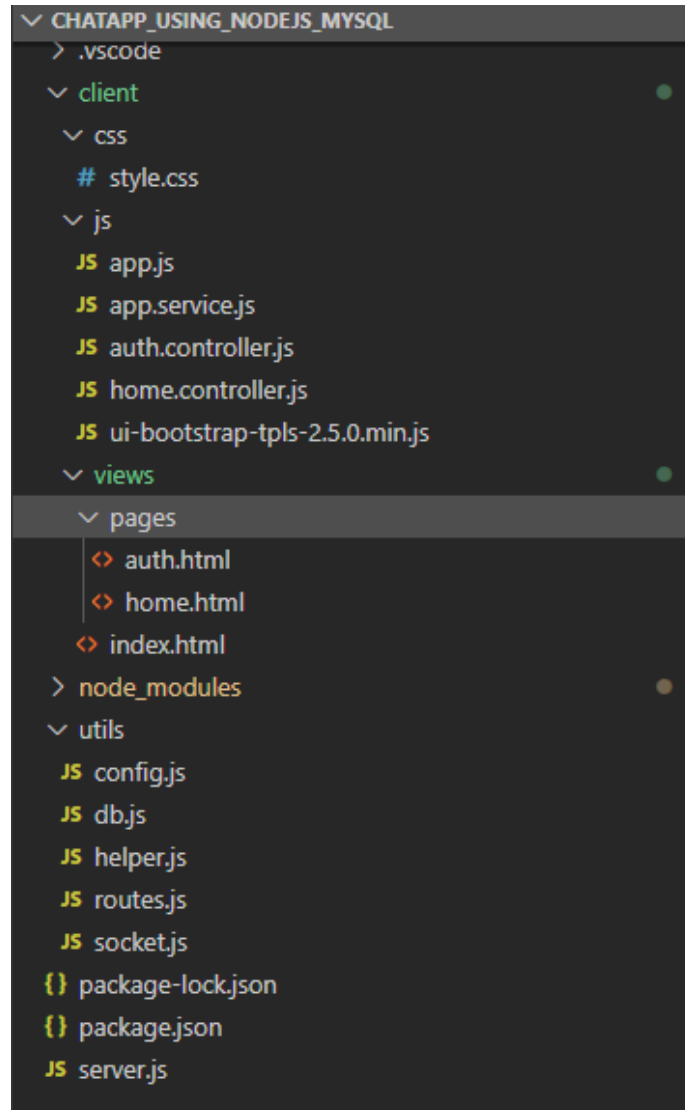
Projenin kök dizininde /client , /utils ve server.js dosya ve klasörleri bulunmaktadır.

Server.js dosyasının içerisinde NodeJS sunucusunu kurmak için gerekli kodlar bulunmaktadır.

/client klasörünün içerisinde ise 3 adet alt dizin bulunmaktadır. Bu dizinler /css , /js ve /views olmak üzere 3 adettir. Web sitesi için yazılmış olan stil dosyaları, javascript dosyaları ve gerekli bootstrap paketleri bu dizinde bulunmaktadır.

/utils klasöründe ise projemizde gerekli olan veritabanı bağlantısının, veritabanı işlemlerinin, anlık veri etkileşimi için gerekli olan socket bağlantısının ve yönlendirme için gerekli olan js dosyaları bulunmaktadır.

Resim 1. Geliştirilen Uygulamanın Kök Dizini



Aşağıdaki kodda uygulama yollarını ve soket olaylarını ekledik ve uygulama yapılandırmasının kurulumunu yaptık.

4.2.server.js

```
const express = require("express");
const http = require('http');
const socketio = require('socket.io');
const bodyParser = require('body-parser');

const socketEvents = require('./utils/socket');
const routes = require('./utils/routes');
const config = require('./utils/config');

class Server{

  constructor(){
    this.port = process.env.PORT || 3000;
    this.host = `localhost`;

    this.app = express();
    this.http = http.Server(this.app);
    this.socket = socketio(this.http);
  }

  appConfig(){
    this.app.use(
      bodyParser.json()
    );
    new config(this.app);
  }

  includeRoutes(){
    new routes(this.app).routesConfig();
    new socketEvents(this.socket).socketConfig();
  }

  appExecute(){
    this.appConfig();
    this.includeRoutes();

    this.http.listen(this.port, this.host, () => {
      console.log(`Listening on http://${this.host}:${this.port}`);
    });
  }
}

const app = new Server();
app.appExecute();
```

4.3.MySQL Veritabanı

chatApp user	
ID	: int(11)
email	: varchar(10)
password	: varchar(20)
name	: varchar(25)
surname	: varchar(50)
isOnline	: enum('N','Y')
socketid	: varchar(20)

chatApp token	
ID	: int(11)
token	: varchar(10)
startTime	: datetime
endTime	: datetime
userID	: int(11)

chatApp message	
ID	: int(11)
from_user_id	: varchar(45)
to_user_id	: varchar(45)
message	: text
time	: datetime
readTime	: datetime

```
-- phpMyAdmin SQL Dump
-- version 4.9.2
-- https://www.phpmyadmin.net/
-- Host: localhost
-- Generation Time: Jan 03, 2020 at 09:05 PM
-- Server version: 10.4.10-MariaDB
-- PHP Version: 7.1.33
```

```
SET SQL_MODE =
"NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";
```

```
-- Database: `chatApp`
```

```
CREATE TABLE `message` (
  `ID` int(11) NOT NULL,
  `from_user_id` varchar(45) DEFAULT NULL,
  `to_user_id` varchar(45) DEFAULT NULL,
  `message` text DEFAULT NULL,
  `time` datetime DEFAULT NULL,
  `readTime` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `token` (
  `ID` int(11) NOT NULL,
  `token` varchar(10) NOT NULL,
  `startTime` datetime DEFAULT NULL,
  `endTime` datetime DEFAULT NULL,
  `userID` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `user` (
  `ID` int(11) NOT NULL,
  `email` varchar(10) NOT NULL,
  `password` varchar(20) NOT NULL,
  `name` varchar(25) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `isOnline` enum('N','Y') NOT NULL,
  `socketid` varchar(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
ALTER TABLE `message`
  ADD PRIMARY KEY (`ID`);
```

```
ALTER TABLE `token`
  ADD PRIMARY KEY (`ID`),
  ADD KEY `userID` (`userID`);
```

```
ALTER TABLE `user`
  ADD PRIMARY KEY (`ID`);
```

```
ALTER TABLE `message`
  MODIFY `ID` int(11) NOT NULL
  AUTO_INCREMENT;
```

```
ALTER TABLE `token`
  MODIFY `ID` int(11) NOT NULL
  AUTO_INCREMENT;
```

```
ALTER TABLE `user`
  MODIFY `ID` int(11) NOT NULL
  AUTO_INCREMENT;
```

```
ALTER TABLE `token`
  ADD CONSTRAINT `token_ibfk_1` FOREIGN KEY
  (`userID`) REFERENCES `user` (`ID`);
COMMIT;
```

Projemizde veritabanı olarak MySQL veritabanı kullanıyoruz. NodeJS sunucusunu MySQL sunucusuna bağlamak için node-mysql paketi kullanacağız.

Db.js dosyamızın içerisinde gerekli olan bağlantılar mevcuttur.

4.4.db.js

```
const mysql = require('mysql');

class Db {
  constructor(config) {
    this.connection = mysql.createPool({
      connectionLimit: 100,
      host: '127.0.0.1',
      user: 'root',
      password: '',
      database: 'chat',
      debug: false
    });
  }

  query(sql, args) {
    return new Promise((resolve, reject) => {
      this.connection.query(sql, args, (err, rows) => {
        if (err)
          return reject(err);
        resolve(rows);
      });
    });
  }

  close() {
    return new Promise((resolve, reject) => {
      this.connection.end(err => {
        if (err)
          return reject(err);
        resolve();
      });
    });
  }
}

module.exports = new Db();
```

Projemizde MySQL veritabanında gerçekleştirilecek olan CRUD işlemleri için gerekli olan helper.js sınıfı yazılmıştır.

4.5.helper.js

```
const DB = require('./db');

class Helper{

  constructor(app){
    this.db = DB;
  }

  async userNameCheck (username){
    return await this.db.query(`SELECT count(username) as count FROM user
WHERE LOWER(username) = ?`, `${username}`);
  }

  async registerUser(params){
    try {
      return await this.db.query("INSERT INTO user (`username`,`password`
`,`online`) VALUES (?,?,"), [params['username'],params['password'],'Y']);
    } catch (error) {
      console.error(error);
      return null;
    }
  }

  async loginUser(params){
    try {
      return await this.db.query(`SELECT * FROM user WHERE username = ?
AND password = ?`, [params.username,params.password]);
    } catch (error) {
      console.error(error);
      return null;
    }
  }

  async userSessionCheck(userId){
    try {
      const result = await this.db.query(`SELECT online,username FROM us
er WHERE id = ? AND online = ?`, [userId,'Y']);
      if(result !== null){
        return result[0]['username'];
      }else{
        return null;
      }
    }
  }
}
```

```

    } catch (error) {
        return null;
    }
}

async addSocketId(userId, userSocketId){
    try {
        return await this.db.query(`UPDATE user SET socketid = ?, online=
? WHERE id = ?`, [userSocketId,'Y',userId]);
    } catch (error) {
        console.log(error);
        return null;
    }
}

async isUserLoggedOut(userSocketId){
    try {
        return await this.db.query(`SELECT online FROM user WHERE socketid
= ?`, [userSocketId]);
    } catch (error) {
        return null;
    }
}

async logoutUser(userSocketId){
    return await this.db.query(`UPDATE user SET socketid = ?, online= ? WH
ERE socketid = ?`, ['', 'N', userSocketId]);
}

getChatList(userId, userSocketId){
    try {
        return Promise.all([
            this.db.query(`SELECT id,username,online,socketid FROM user WH
ERE id = ?`, [userId]),
            this.db.query(`SELECT id,username,online,socketid FROM user WH
ERE online = ? and socketid != ?`, ['Y',userSocketId])
        ]).then( (response) => {
            return {
                userinfo : response[0].length > 0 ? response[0][0] : respo
nse[0],
                chatlist : response[1]
            };
        }).catch( (error) => {
            console.warn(error);
            return (null);
        });
    } catch (error) {
        console.warn(error);
        return null;
    }
}

```

```

    }
  }

  async insertMessages(params){
    try {
      return await this.db.query(
        "INSERT INTO message (`from_user_id`,`to_user_id`,`message`) v
alues (?,?/?)",
        [params.fromUserId, params.toUserId, params.message]
      );
    } catch (error) {
      console.warn(error);
      return null;
    }
  }

  async getMessages(userId, toUserId){
    try {
      return await this.db.query(
        `SELECT id,from_user_id as fromUserId,to_user_id as toUserId,m
essage FROM message WHERE
        (from_user_id = ? AND to_user_id = ? )
        OR
        (from_user_id = ? AND to_user_id = ? ) ORDER BY id ASC
        `,
        [userId, toUserId, toUserId, userId]
      );
    } catch (error) {
      console.warn(error);
      return null;
    }
  }
}

module.exports = new Helper();

```

Bu sınıfın içerisinde kullanıcı isminin doğrulanması, kullanıcı kaydının yapılması, kullanıcı girişi yapılan kullanıcının socket id'sinin atanması ve gönderilen mesajların veritabanına kaydedilmesi için gerekli olan yardımcı methodlar yazılmıştır.

4.6.socket.js

Kullanıcı bir sokete bağlanacağı zaman, execute socket.js dosyası ile kullanıcının socket kimliğini güncelleyeceğiz.

Bu ara katman yazılımı herhangi bir socket olayı yürütülmeden önce yürütülür ve socket sunucusu her başlatıldığında yeni socket kimliğini sokar.

Aşağıdaki kodda, socketConfig () methodu MySql sorgusunu bu ara katman yazılımına yazacaktır. Bunu yapmak için,

```
const path = require('path');
const helper = require('./helper');

class Socket{

  constructor(socket){
    this.io = socket;
  }

  socketEvents(){

    this.io.on('connection', (socket) => {

      socket.on('chat-list', async (userId) => {

        let chatListResponse = {};

        if (userId === '' && (typeof userId !== 'string' || typeof use
rId !== 'number')) {

          chatListResponse.error = true;
          chatListResponse.message = `User does not exists.`;

          this.io.emit('chat-list-response', chatListResponse);
        }else{
          const result = await helper.getChatList(userId, socket.id)
;

          this.io.to(socket.id).emit('chat-list-response', {
            error: result !== null ? false : true,
            singleUser: false,
            chatList: result.chatlist
          });
        }
      });
    });
  }
}
```

```
        socket.broadcast.emit('chat-list-response', {
            error: result !== null ? false : true,
            singleUser: true,
            chatList: result.userinfo
        });
    }
});

socket.on('add-message', async (data) => {

    if (data.message === '') {

        this.io.to(socket.id).emit(`add-message-
response`, `Message cant be empty`);

    }else if(data.fromUserId === ''){

        this.io.to(socket.id).emit(`add-message-
response`, `Unexpected error, Login again.`);

    }else if(data.toUserId === ''){

        this.io.to(socket.id).emit(`add-message-
response`, `Select a user to chat.`);

    }else{
        let toSocketId = data.toSocketId;
        const sqlResult = await helper.insertMessages({
            fromUserId: data.fromUserId,
            toUserId: data.toUserId,
            message: data.message
        });
        this.io.to(toSocketId).emit(`add-message-
response`, data);
    }
});

socket.on('logout', async () => {
    const isLoggedIn = await helper.logoutUser(socket.id);
    this.io.to(socket.id).emit('logout-response', {
        error : false
    });
    socket.disconnect();
});
```



```

socket.on('disconnect', async () => {
  const isLoggedIn = await helper.logoutUser(socket.id);
  setTimeout(async () => {
    const isLoggedIn = await helper.isUserLoggedIn(socket.id);

    if (isLoggedIn && isLoggedIn !== null) {
      socket.broadcast.emit('chat-list-response', {
        error: false,
        userDisconnected: true,
        socketId: socket.id
      });
    }
  }, 1000);
});

});

}

socketConfig(){

  this.io.use( async (socket, next) => {
    let userId = socket.request._query['userId'];
    let userSocketId = socket.id;
    const response = await helper.addSocketId( userId, userSocketId);
    if(response && response !== null){
      next();
    }else{
      console.error(`Socket connection failed, for user Id ${userId}`);
    }
  });

  this.socketEvents();
}

module.exports = Socket;

```

4.7.routes.js

Routes.js dosyamızın içerisinde projemize ait rest servisler bulunmaktadır.

```
this.app.post('/usernameCheck', async (request, response) =>{
  const username = request.body.username;
  if (username === "" || username === undefined || username === null
) {
    response.status(412).json({
      error : true,
      message : `username cant be empty.`
    });
  } else {
    const data = await helper.userNameCheck(username.toLowerCase())
  );
    if (data[0]['count'] > 0) {
      response.status(401).json({
        error:true,
        message: 'This username is alreday taken.'
      });
    } else {
      response.status(200).json({
        error:false,
        message: 'This username is available.'
      });
    }
  }
});
```

Sınıfımızın /usernameCheck ismiyle post ettiğimiz methodunda kullanıcı adı ile ilgili null check kontrolleri ve diğer kontrollerler yapılmaktadır.

```
this.app.post('/registerUser/', async (request, response) => {
  const registrationResponse = {}
  const data = {
    username : request.body.username,
    password : request.body.password
  };

  if(data.username === '') {
    registrationResponse.error = true;
    registrationResponse.message = `username cant be empty.`;
    response.status(412).json(registrationResponse);
  }
  else if(data.password === ''){
    registrationResponse.error = true;
    registrationResponse.message = `password cant be empty.`;
    response.status(412).json(registrationResponse);
  }
  else{
    const result = await helper.registerUser( data );

    if (result === null) {
      registrationResponse.error = true;
      registrationResponse.message = `User registration unsuccessful, try after some time.`;
      response.status(417).json(registrationResponse);
    }
    else {
      registrationResponse.error = false;
      registrationResponse.userId = result.insertId;
      registrationResponse.message = `User registration successful.`;
      response.status(200).json(registrationResponse);
    }
  }
});
```

Sınıfımızın /registerUser ismiyle post ettiğimiz methodunda yardımcı sınıflarımız kullanılarak kullanıcı kaydı ile ilgili işlemler gerçekleştirilmiştir.

```
this.app.post('/login/', async (request, response) =>{
  const loginResponse = {}
  const data = {
    username : request.body.username,
    password : request.body.password
  };
  if(data.username == '' || data.username == null) {
    loginResponse.error = true;
    loginResponse.message = `username cant be empty.`;
    console.log(json.parse(data));
    response.status(412).json(loginResponse);
  }
  else if(data.password == '' || data.password == null){

    loginResponse.error = true;
    loginResponse.message = `password cant be empty.`;
    response.status(412).json(loginResponse);
  }
  else{
    const result = await helper.loginUser(data);
    if (result && result.length ) {
      loginResponse.error = false;
      loginResponse.userId = result[0].id;
      loginResponse.user= result[0];
      loginResponse.message = `User logged in.`;
      response.status(200).json(loginResponse);
    }
    else {
      loginResponse.error = true;
      loginResponse.message = `Invalid username and password combination.`;
      response.status(401).json(loginResponse);
    }
  }
});
```

Sınıfımızın /login ismiyle post ettiğimiz methodunda yardımcı sınıflarımız kullanılarak kullanıcı girişi ile ilgili işlemler gerçekleştirilmiştir.

Sınıfımızın /userSessionCheck ismiyle ve /getMessages ismi ile post ettiğimiz methodunda yardımcı sınıflarımız kullanılarak kullanıcı girişinin kontrolü ve mesaj gönderimi ile ilgili işlemler gerçekleştirilmiştir.

```
this.app.post('/userSessionCheck', async (request, response) => {
  const userId = request.body.userId;
  const sessionCheckResponse = {}
  if (userId == '') {
    sessionCheckResponse.error = true;
    sessionCheckResponse.message = `User Id cant be empty.`;
    response.status(412).json(sessionCheckResponse);
  } else {
    const username = await helper.userSessionCheck(userId);
    if (username === null || username === '') {
      sessionCheckResponse.error = true;
      sessionCheckResponse.message = `User is not logged in.`;
      response.status(401).json(sessionCheckResponse);
    } else {
      sessionCheckResponse.error = false;
      sessionCheckResponse.username = username;
      sessionCheckResponse.message = `User logged in.`;
      response.status(200).json(sessionCheckResponse);
    }
  }
});
```

```
this.app.post('/getMessages', async (request, response) => {
  const userId = request.body.userId;
  const toUserId = request.body.toUserId;
  const messages = {}
  if (userId === '') {
    messages.error = true;
    messages.message = `userId cant be empty.`;
    response.status(200).json(messages);
  } else {
    const result = await helper.getMessages( userId, toUserId);
    if (result === null) {
      messages.error = true;
      messages.message = `Internal Server error.`;
      response.status(500).json(messages);
    } else {
      messages.error = false;
      messages.messages = result;
      response.status(200).json(messages);
    }
  }
});
```

AngularJS için gerekli olan js dosyaları /js dizini içerisinde bulunmaktadır.

app.js: AngularJs'in başlatıldığı dosyadır.

app.services.js: AngularJs hizmeti kodunu içerir.

auth.controller.js: Bu, uygulamanın giriş ve kayıt kısmı için kullanılan denetleyici dosyasıdır.

home.controller.js: Yine uygulamanın ana sayfasında kullanılan bir denetleyici dosyasıdır.

4.8.app.js

```
'use strict';
const app = angular.module('app', ['ngRoute', 'ui.bootstrap']);

/*
 * configuring our routes for the app
 */
app.config(function ($routeProvider, $locationProvider) {
  $routeProvider
    .when('/', {
      templateUrl: '/views/pages/auth.html',
      controller: 'authController'
    })
    .when('/home/:userId', {
      templateUrl: '/views/pages/home.html',
      controller: 'homeController'
    });

  $locationProvider.html5Mode(true);
});

app.factory('appService', ($http) => {
  return new AppService($http)
});
```

4.9.app.service.js

```
class AppService{
  constructor($http){
    this.$http = $http;
    this.socket = null;
  }
  httpCall(httpData){
    if (httpData.url === undefined || httpData.url === null || httpData.url === ''){
      alert(`Invalid HTTP call`);
    }
    const HTTP = this.$http;
    const params = httpData.params;
    return new Promise( (resolve, reject) => {
      HTTP.post(httpData.url, params).then( (response) => {
        resolve(response.data);
      }).catch( (response, status, header, config) => {
        reject(response.data);
      });
    });
  }
  connectSocketServer(userId){
    const socket = io.connect( { query: `userId=${userId}` });
    this.socket = socket;
  }
  socketEmit(eventName, params){
    this.socket.emit(eventName, params);
  }
  socketOn(eventName, callback) {
    this.socket.on(eventName, (response) => {
      if (callback) {
        callback(response);
      }
    });
  }
  getMessages(userId, friendId) {
    return new Promise((resolve, reject) => {
      this.httpCall({
        url: '/getMessages',
        params: {
          'userId': userId,
          'toUserId': friendId
        }
      }).then((response) => {
        resolve(response);
      }).catch((error) => {
        reject(error);
      });
    });
  }
}
```

4.10. auth.controller.js

```
app.controller('authController', function ($scope, $location, $timeout, appService) {

    $scope.data = {
        regUsername : '',
        regPassword : '',
        usernameAvailable : false,
        loginUsername : '',
        loginPassword : ''
    };

    let TypeTimer;
    const TypingInterval = 800;

    $scope.initiateCheckUserName = () => {
        $scope.data.usernameAvailable = false;
        $timeout.cancel(TypeTimer);
        TypeTimer = $timeout( () => {
            appService.httpCall({
                url: '/usernameCheck',
                params: {
                    'username': $scope.data.regUsername
                }
            })
            .then((response) => {
                $scope.$apply( () =>{
                    $scope.data.usernameAvailable = response.error
                    ? true : false;
                });
            })
            .catch((error) => {
                $scope.$apply(() => {
                    $scope.data.usernameAvailable = true;
                });
            });
        }, TypingInterval);
    }

    $scope.clearCheckUserName = () => {
        $timeout.cancel(TypeTimer);
    }
});
```



```
$scope.registerUser = () => {
  appService.httpCall({
    url: '/registerUser/',
    params: {
      'username': $scope.data.regUsername,
      'password': $scope.data.regPassword
    }
  })
  .then((response) => {
    $location.path(`/home/${response.userId}`);
    $scope.$apply();
  })
  .catch((error) => {
    alert(error.message);
  });
}

$scope.loginUser = () => {
  appService.httpCall({
    url: '/login/',
    params: {
      'username': $scope.data.loginUsername,
      'password': $scope.data.loginPassword
    }
  })
  .then((response) => {
    $location.path(`/home/${response.userId}`);
    $scope.$apply();
  })
  .catch((error) => {
    alert(error.message);
  });
}
});
```

BÖLÜM 5

5. Ekran Görüntüleri

5.1.Kullanıcı Girişi

LoginRegister

Username

Enter username

Password

Enter password

Login

5.2.Kullanıcı Kayıt

LoginRegister

Username

Enter username

Password

Enter password

Register

5.3. Anlık Mesajlaşma Ekranı ve Online Kullanıcı Listesi

Welcome kardelen

Chat History With ecemokan

Merhaba Kardelen, nasılsın?

İyiym teşekkür ederim, sen nasılsın?

İyiym ben de, teşekkür ederim.

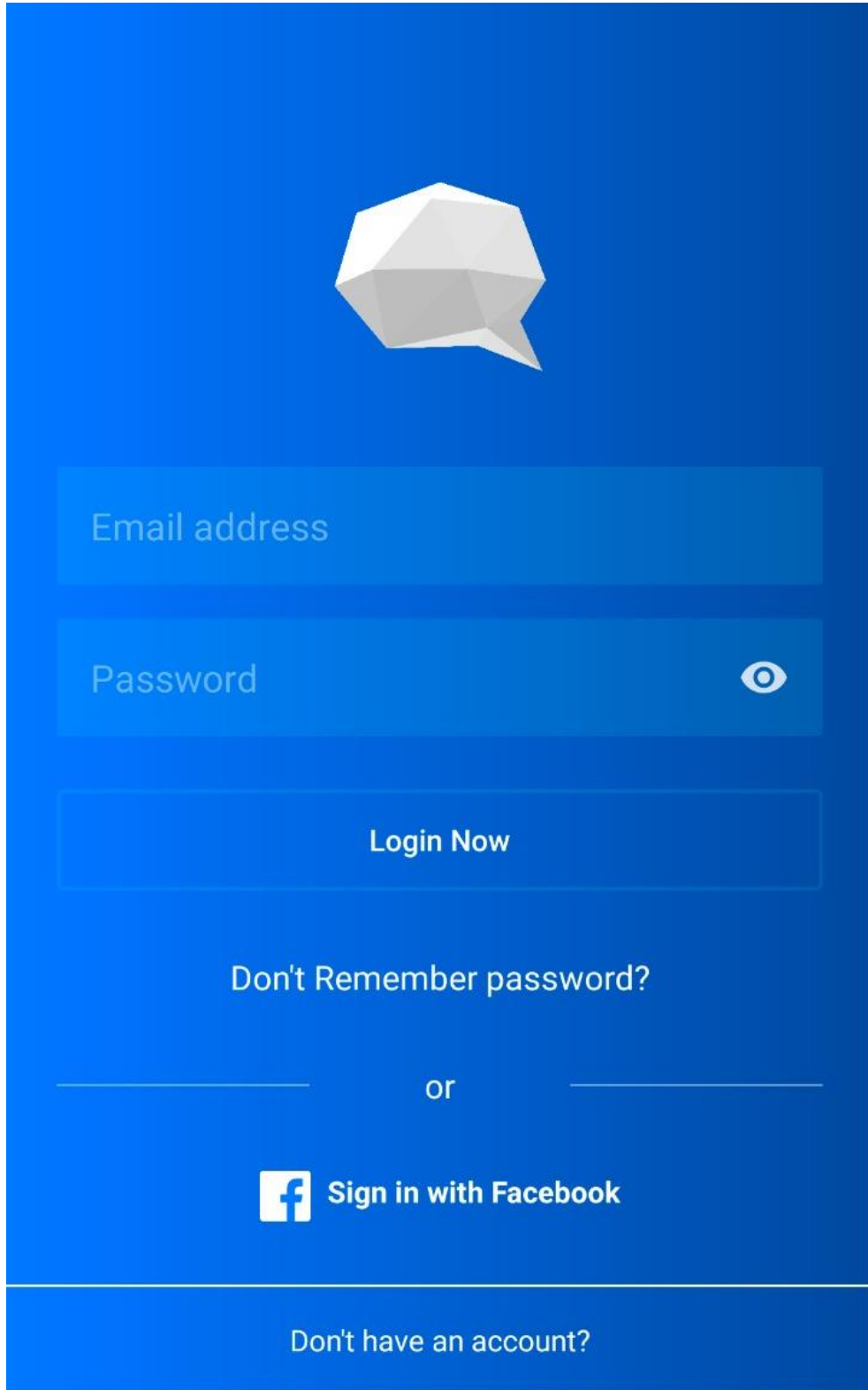
Type and hit Enter

Chat list

admin

ecemokan

5.4.Android Platform Ekran Görüntüsü



A mockup of an Android login screen with a solid blue background. At the top center is a white, low-poly, geometric shape resembling a speech bubble or a stylized letter 'A'. Below this, there are two input fields: the first is labeled 'Email address' and the second is labeled 'Password'. To the right of the password field is a white eye icon. Below the input fields is a button labeled 'Login Now'. Underneath the button is the text 'Don't Remember password?'. Below this is a horizontal line with the word 'or' in the center. Below the line is a Facebook logo followed by the text 'Sign in with Facebook'. At the bottom of the screen is a white horizontal line, and below it is the text 'Don't have an account?'.


Email address

Password

Login Now

Don't Remember password?

or

 Sign in with Facebook

Don't have an account?

Kaynakça

- ❖ Stefan Buttigieg - Learning Node.js for Mobile Application Development
- ❖ Caio Ribeiro Pereira - Building APIs with Node.js
- ❖ Obaid, Z., Sabonchi, A. ve Akay, B.- Klasik Kriptoloji Yöntemlerinin Karşılaştırılması
- ❖ Çağla Özyılmaz – Kriptolojiye Giriş
- ❖ Bora Kaşmer - Angular ve NodeJs Üzerinde SocketIO Kullanılarak Real Time Data Bildirilmesi