

[Mühendislik Fakültesi](https://www.konya.edu.tr/muhendislikvemimarlik)

[Bilgisayar Mühendisliği Bölümü](https://www.konya.edu.tr/bilgisayarmuhendisligi)

Uygulama Tasarımı Dersi Proje Formu

|  |
| --- |
| Proje Başlığı |
| Çocukların Eğitimine Yönelik İnteraktif Görüntülü Görüşme Uygulaması |

|  |  |
| --- | --- |
| Öğrenci Bilgileri | |
| Öğrenci No | 19010011007 |
| Öğrenci Ad Soyad | Muhammed Said TAYLAN |

|  |
| --- |
| Dr.  Hüseyin Haklı |

Mart 2022

Konya

İçindekiler

[1. Proje Konusunun çözmeyi hedeflediği problem .3](#_Toc51147498)

2. Problemin Çözümü ve Özgün Değer…………….…………………………………...…...3

[3. Projenin Amacı ve Hedefi 4](#_Toc51147499)

[4. Projenin iş-zaman çizelgesi 5](#_Toc51147500)

[5. Projede kullanılan donanımlar ve yazılımlar ile ilgili bilgileri 7](#_Toc51147501)

[5.1. Yazılımlar 7](#_Toc51147502)

[5.2. Donanımlar 6](#_Toc51147503)

[6. Projenin yapım aşamaları …………………………………………………………………7](#_Toc51147504)

[Kaynaklar 8](#_Toc51147505)

1. Problemin Belirlenmesi

Özellikle 2020 yılında ülkemizde de baş gösteren COVID-19 pandemi süreci nedeniyle gerek eğitim gerek sağlık gerek ekonomi olmak üzere neredeyse bütün faaliyet ve iş alanları sekteye uğradı. Bundan büyük ölçüde paylarını alanlar şüphesiz öğrenciler oldu. Yüzyüze olması gereken, öğretmen ile öğrencinin gerek sözlü gerek görsel temasının gerekliliğinin özellikle çocuk için çok büyük önem arz ettiği yüzyüze eğitim uzun süre çevrimiçi olarak devam etti. Çevrimiçi eğitimi sağlamak için birçok çevrimiçi görüntülü konuşma uygulaması ortaya çıktı, görüntülü eğitim platformları yaygınlaştı. Bu uygulamalar kişiler arası iletişimi oldukça kolay bir hale getiriyordu. Her ne kadar yol katedilse de hala, yaş farketmeksizin, eğitime özgü bir görüntülü sohbet uygulamasının eksikliği farkedilmektedir. Bahsedilen klasik görüntülü sohbet uygulamalarının, bir eğitim platformu olarak kullanılmasının bazı problemleri bulunmaktadır. Eğitimin kalitesini artırmak için bu problemlerin çözülmesi ve sınıf ortamının ve öğretmen-öğrenci samimiyetinin sağlanabilmesi gereklidir. Hele ki yaş ortalaması küçük olan öğrenciler için zaten öğretmen-öğrenci ilişkisinin zayıf olduğu çevrimiçi dönemde öğrencileri ekran başında tutmak ek çaba ve çekici unsurları gerektirmektedir.

Bu problemlerden bazıları:

1. Eğitim verilen yaş kitlesini göz önünde bulundurmadan uygulama tasarımının ve renk seçiminin aynı olması ve özelleştirilememesi ya da belli seviyede özelleştirilebilmesi
2. Görüşmeye özel bir puanlama ve ödül sisteminin mevcut olmaması
3. Özellikle çocuklar için çekici bir unsur barındırmamaları
4. Problemin Çözümü ve Özgün Değer

Mevcut görüntülü konuşma uygulamalarında mevcut olan oda yöneticiliği, kullanıcıları sessize alma, ekran paylaşımı, sohbet ekranı gibi temel özellikler mevcut olacaktır. Bu özelliklere ek olarak ayrıca her bir ayrı görüşme ayrı bir odada gerçekleştirilme; odaları adlandırma; diğer kullanıcılara beğeni emojisi gönderme; kullanıcının, kendisine gelen beğeni emojilerini kimlerin gönderdiğini görebilme; belli bir grup için kalıcı olarak kullanılacak odaları birer sınıfa dönüştürme ve böylece oda ayarlarını ve gelişimleri kaydetme; katılımcıların, kendi ekran görünümlerini değiştirebilme imkanı sunmaktadır. Sayılan ek özellikler bütün görüşme türleri için mevcuttur. Ancak elbette, eğitimi temel alan bir uygulamada yukarıda bahsedilen problemlere çözüm olacak başka ek özellikler de olmalıdır. Bunlar şöyle sıralanabilir:

1. Görüşme odalarının “herkes için” ve “eğitim için” olmak üzere iki türü bulunacaktır.
2. Eğitim için olan odalarda aşağıdaki ek özellikler bulunacaktır:
   1. Puanlama sistemiyle öğretmenler öğrencilere puan verebilecek
   2. Katılımcılar, konuşmakta olan katılımcıya beğeni dışında belli bazı emojileri gönderebilecek
   3. Sınıf ya da oda farketmeksizin, öğrencilere verilen puanlar, öğrencilerin profiline yansıyacak
   4. Sınıf olarak belirlenmeyen normal odalardaki görüşme sonlarında toplam puanlar, beğeni sayıları ekranda gösterilecek ve puanlamada ilk üçe girenler ekranda farklı biçimlerde gösterilecektir. Öğretmen dilerse bu bilgileri cihazına kaydedebilecek
   5. Öğrenciler ders sonunda kazandığı puanları animasyonlu bir şekilde görebilecek
3. Projenin Amacı ve Hedefi

Proje kapsamında özellikle çocuklar için görüntülü konuşma uygulaması geliştirilecektir. Çocukların derslere katılım yüzdesinin artırılması, derslerin daha çekici hale getirilmesi, görüşmelerin “oyunlaştırılması” ve böylece eğitim kalitesinin artırılması hedeflenmektedir.

Bilindiği üzere çocuklar, herhangi çekici bir unsurun bulunmadığı ortamlardan çok çabuk sıkılırlar. Özellikle görselliğin çok önemli bir dönem olduğu çocukluk döneminde video görüşmelerinde mümkün olduğunca fazla görselliğe başvurulması; ders süresi boyunca çocukların, sıkıcı bir anlatıcı-dinleyici ortamından ziyade interaktif bir öğretmen-öğrenci ortamıyla başbaşa bırakılmaları; puanlama sistemiyle birlikte çocuklar için oldukça itici bir güç olan rekabet ortamının da oluşturulması amaçlanmaktadır.

# Projenin iş-zaman çizelgesi

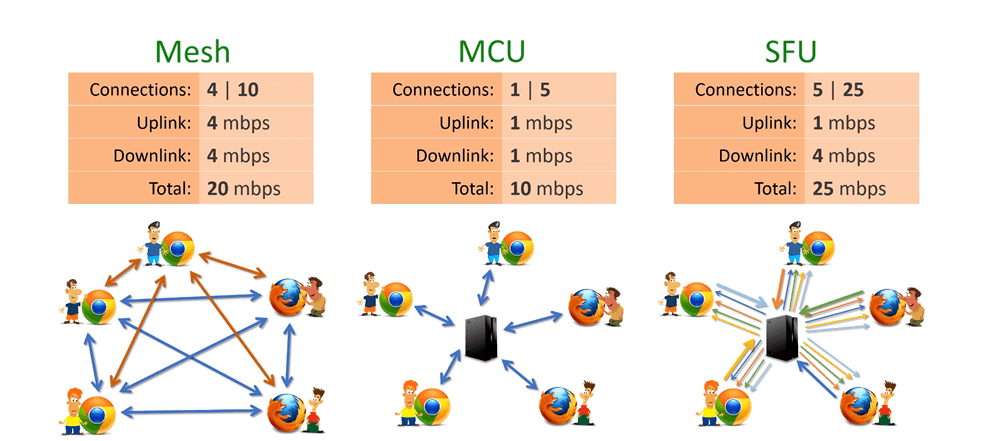
İş-zaman çizelgesi

|  |  |  |  |
| --- | --- | --- | --- |
| İP.  No. | İş Paketlerinin Adı ve Hedefleri | Zaman Aralığı (Haftalık) | Başarı Ölçütü |
| 1 | Projenin Fikir Aşamasının Tamamlanması ve Raporun Hazırlanması: Yazıya geçirilmeden kabataslak bir şekilde mevcut proje fikri yazıya dökülüp ayrıntıları düşünülecektir. Proje fikrinin; problemi, çözümü ve hedefleri ayrıntılarıyla birlikte rapor hazırlanacaktır. | 1. hafta | Akla gelmiş olan bir fikrin her ayrıntısını düşünerek yazıya geçirmek büyük önem arzetmektedir. Havada asılı duran bir fikir uçup gitmeye çok müsaittir. Ayrıca fikrin yazılı halinin mevcut olması raporu kolay bir şekilde hazırlamaya da yardımcı olacaktır. Bu sebepten dolayı proje fikrinin tamamlanmasının projenin başarısına katkısı 15% oranındadır. |
| 2 | Eğitim Süreci: Projede kullanılacak olan teknolojiler araştırılıp pekiştirmeler yapılacaktır. Araştırma sürecinin neredeyse tamamını kapsayacak olan, WebRTC teknolojisidir. Hemen hemen bütün görüntülü konuşma uygulamalarının olmazsa olmazı, yaklaşık 10 yıl önce ortaya çıkan WebRTC teknolojisi ile çalışmaktadır. WebRTC, sunucuya çok az yük bindirerek uçtan uca anlık iletişim sağlamaktadır. WebRTC kullanılarak hem görüntü ve ses hem de ekran hem de gerekirse veri paylaşımı yapılabilmektedir.  Ayrıca NestJS framework’ü öğrenilecek.  Bunların dışında öğrenilmesi gereken ayrıntılar da olacaktır. | 2 - 6. haftalar | Görüntülü konuşma uygulamalarının bel kemiği olan, modern tarayıcılarda dahili olarak gelen WebRTC teknolojisini yeter seviyede öğrenmek projenin geleceği için çok önemlidir. Bu sebeple eğitim sürecinin projenin başarısına katkısı 15% oranındadır. |
| 3 | Sunucunun İnşa Edilmesi: Projenin sunucu kısmı Typescript dili ile yazılacağı için Typescript diliyle ilgili küçük bir tekrarın ardından NestJS framework’ü kullanılarak sunucu oluşturulacaktır. Nesne yönelimli felsefesi ile yapılacaktır. WebRTC sunucuya yük bindirmese de sunucu ile küçük işlemleri vardır. | 7., 8. ve 9. haftalar | Proje, bir internet uygulaması olduğu için elbette sunucu çok ama çok önemlidir. Tüm mesajlaşma, görüntü-ses, ekran paylaşımı işlemleri; veritabanı işlemleri sunucudan dolaylı veya direkt olarak geçeceği için sunucunun inşa edilmesinin projenin başarısına katkısı 25% oranındadır. |
| 4 | Veritabanı Modellemesi ve Sunucuya Bağlanması:  Sisteme kayıtlı olan kullanıcıların bilgileri, oda ve sınıf bilgileri, eğitim sınıfları için puan ve beğeni bilgileri ve daha birçok veri kaydedilecektir. Bunun için veritabanı modellemesi hakkında oturup düşünmek gerekmektedir. Veritabanını oluşturulduktan sonra veritabanının sunucuya bağlanması gerekmektedir. Veritabanı sunucuya bağlandıktan sonra veritabanı ile sunucu arasındaki işlemler senkron ilerleyecektir. | 10. ve 11. haftalar | Projede veritabanı işlemleri oldukça fazla kullanılacaktır. Veritabanının olmaması demek kullanıcılarım olmaması demek olduğu için veritabanı düşünülmek zorundadır. Böyle kritik bir konumda olduğu için veritabanı modellemesi ve bağlanmasının projenin başarısına katkısı 15% oranındadır. |
| 5 | Arayüz Tasarımı ve Sunucu-İstemci İletişimi:  Projenin hedeflerinden birisi mümkün olduğunca fazla görsellik ve çekici bir tasarım olduğu için nasıl bir tasarım yapılacağı konusunda düşünmek gerekmektedir. Ayrıca istemci tarafındaki işlemler de azımsanamayacak kadar fazladır. Verilerin sunucudan alınması, kullanıcıya uygun şekilde gösterilmesi, interaktivite, kullanıcı tarafından tasarımın özelleştirilebilmesi bunlardan sadece birkaçıdır. | 12., 13. ve 14. haftalar | Bu projenin iki köşetaşı vardır: Sunucu ve istemci. Sunucu inşa etmek her ne kadar zor olsa da sunucu tarafını kullanıcı görmemektedir. Kullanıcının ilgilendiği alan istemci tarafı ve tasarımlardır. Bu yüzden olduğunda akıcı, hızlı ve estetik bir arayüz tasarlayıp kullanıcı bilgisayar iletişimini sezgisel bir hale getirmek bu projenin en büyük amaçlarındandır. Bu yüzden arayüz tasarımı ve sunucu-istemci iletişiminin projenin başarısına katkısı 30% oranındadır. |

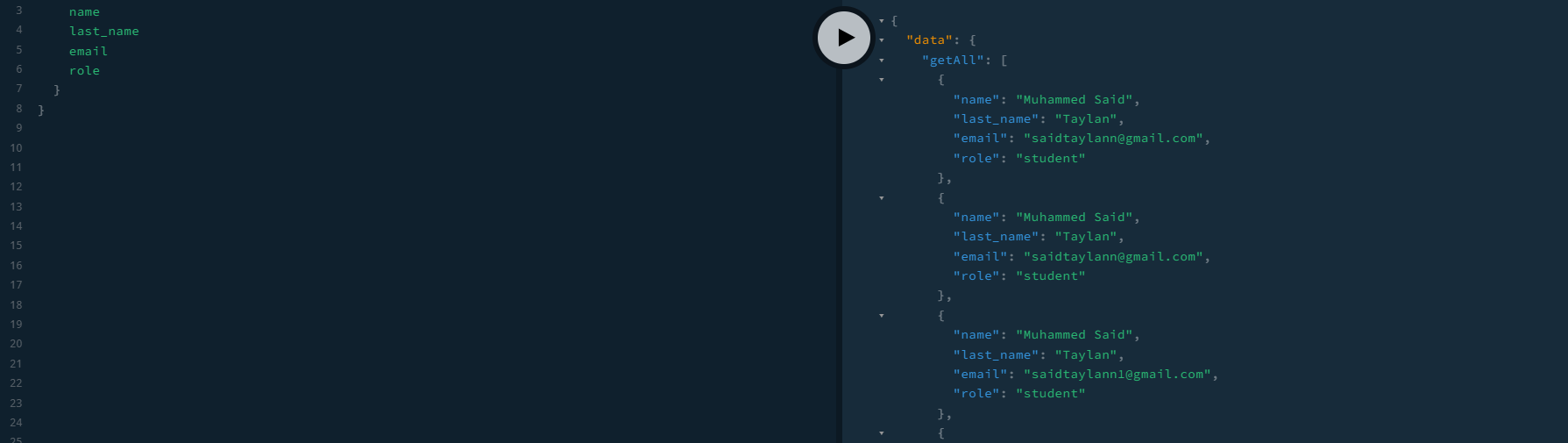
# Projede kullanılacak yöntem, donanımlar ve yazılımlar ile ilgili bilgiler

1. Yazılım
   1. Javascript: Web uygulamaları ve sunucu inşa etmek için kullanılan bir programlama dilidir. Projede istemci tarafında kullanılacaktır.
   2. Typescript: Javascript üzerine bina edilmiş bir programlama dilidir. Projede sunucu tarafında kullanılacaktır.
   3. MongoDB: NoSQL bir veritabanıdır. Projede verileri depolamak için kullanılacak veritabanıdır.
   4. NodeJS: Temel internet gereksinimlerini karşılayarak sunucu inşa etmeyi sağlayan bir Javascript çalışma zamanı ortamıdır. Projede sunucu inşa etmek için kullanılacak olan framework, NodeJS üzerine bina edilecektir.
   5. NestJS: Sunucuyu inşa etmek için kullanılacak olan Javascript framework’üdür. NodeJS üzerine inşa edilecektir.
   6. WebRTC: Tarayıcılar arasında görüntü ve ses aktarılmasını sağlayan uçtan uca bir teknolojidir. Tarayıcılarda dahili olarak gelir. Javascript için tarayıcılar tarafından bir API olarak sağlanır. Ayrıca projede, mesajlar da dahil olmak üzere bütün veri aktarımını sağlayacaktır.
   7. ION SFU: GO dili ile inşa edilmiş olan bir SFU sunucusudur. WebRTC istemcileri arasında iletişimi sağlayan bir sunucu görevini görecektir. RPC yöntemi ile ION-SFU metotları Jaavascript tarafından çağırılabilecektir.
   8. VueJS: Bir Javascript Front-end framework’üdür. Sunucu ile iletişim ve bütün mantıksal işlemler bu framework üzerinde dönecektir.
   9. Tailwind: Bir CSS framework’üdür. Projede tasarım ve arayüz için saf CSS yerine Tailwind kullanılacaktır.
   10. Docker: Proje için yalıtılmış bir ortam sağlar. Projenin gerektirdiği bütün bağımlılıkları kaydeder. Proje başka bir cihaza ya da ortama aktarıldığında Docker’in tek bir komutuyla proje için gerekli her şey hazır olur. Platform ve ortam bağımlılığını ortadan kaldırır. Bütün proje bir Docker Container’i içinde gerçekleştirilecektir.
2. Donanım
   1. Modern web tarayıcılarını destekleyen kamera ve mikrofonu olan herhangi bir cihaz: Proje, bir görüntülü konuşma uygulaması olduğundan dolayı tarayıcısı olan herhangi bir cihaz yeterlidir. Elbette görüntü ve sesin gidebilmesi için bir kamera veya mikrofon duruma göre gereklidir.

# Projenin yapım aşamaları

1. Hafta: İlk olarak taslak şeklinde sadece fikirde olan proje bütün ayrıntılarıyla düşünüldü ve oldukça kapsamlı bir not alma işlemi gerçekleştirildi. “Bu uygulama neyi hedefliyor?”, “Neden çocuklara yönelik?”, “Hangi platformlarda var olacak?”, “Piyasada var olan hangi uygulamaya benzemektedir?”, “Bu projenin muadilleri ile benzer yönleri ve özgün yönleri nelerdir?”, “Hangi teknolojiler kullanılacak?” sorularına tek tek cevap arandı ve bulundu, detaylı olarak yazıldı. Bu sorular ve cevapları yazıya geçirildikten sonra rapor hazırlandı.  
   Görüntülü konuşma uygulamalarının hangi teknolojilerle çalıştığı araştırıldı. Bunun sonucunda yaklaşık 11 yıldır var olan, Google tarafından geliştirilen WebRTC teknolojisine rastlanıldı. Piyasada mevcut olan bir çok görüntülü konuşma uygulaması WebRTC ile yapılmıştır. WebRTC sadece görüntülü konuşmayı sağlamaz. WebRTC, üç temel sütundan oluşur: Tarayıcıdan kamera, ses ve ekran paylaşımının alınabilmesi için Javascript objesi; Alınan verilerin diğer alıcı uzak taraflara gönderilebilmesini sağlayan RTCPeerConnection objesi; görüntü ve ses dışında isteğe bağlı olan diğer tür verilerin gönderilebilmesini sağlayan RTCDataChannel objesi. WebRTC, verileri uçtan uca gönderir. Peki karşı tarafın IP adresini nereden bilebilir? İşte burada bir Signal Server devreye giriyor. STUN adı verilen bir mekanizma ile, sunucular ile paylaşım yapılacak olan tarafın IP bilgisi ve diğer gerekli bilgiler alınır ve artık sunucuya ihtiyaç duyulmadan sadece istemciler arasında uçtan uca iletişim sağlanabilir. WebRTC; güvenliği, paketlemeyi, paket açmayı ve diğer gerekli işlemleri kendisi yapmaktadır. WebRTC öğrenmek için birçok kaynak dolaşıldı ancak tam olarak anlaşılamadı. Son olarak Google Developers Youtube Kanalı’nda[1] yayınlanan 3 WebRTC tanıtım videosu izlendi ve WebRTC’nin nasıl çalıştığı ile ilgili kesin bir kanıya sahip olundu.
2. İlk haftada araştırılan WebRTC’de öğrenilen bazı bilgilerin yanlış olduğunu farkına varıldı. STUN Server; karşı tarafın değil, kullanıcının kendisinin Public IP adresini sağlıyor. Ancak STUN sunucusunun kullanılmasının mümkün olmadığı zamanlarda (güvenlik duvarları ve başka etkenler) kullanıcılar uçtan uca değil de bir TURN server üzerinden medya alışverişlerini gerçekleştirirler. Bu da uçtan uca bağlantının olmaması demektir. Ancak STUN server sorunsuz olarak %80’in üzerinde çalışma oranı sunmaktadır. İnternette WebRTC ile ilgili uygulamalı çok fazla örnek bulunamamıştı. Uzun bir araştırma sonucunda uygulamalı bir Youtube video serisine [2] denk gelindi. Videonun Github deposu bulunmadığı için videolardaki kodlar tek tek yazıldı. Ancak serinin sonunda, videoları izleyerek yazılan uygulama çalışmadı. Her ne kadar çalışmasa da WebRTC’nin hangi sırayla çalıştığı hakkında pratik yapılmış oldu. Bu fikre sahip olduktan sonra başka WebRTC örnekleri arandı. Ayrıca WebRTC’nin bir dökümantasyonu olan Mozilla Developer platformunda WebRTC’nin çalışma sırası [3] okundu. Github deposundaki kodların gerek sunucu gerek istemci tarafında değişiklikler yapıldı çünkü kod hiçbir şekilde çalışmıyordu. Öğrenilen bilgiler ışığında değişiklikler yapıldıktan sonra bazı hatalarla karşılaşıldı. Fakat bunlar çok kısa bir süre içerisinde çözüldü. Ancak bir problem vardı. Diğer kullanıcının videosu karşı tarafa ulaşmıyordu. Bütün bu denemeler Chrome internet tarayıcısı üzerine yapıldı. Firefox üzerinde de denemeler yapmak için aynı adrese Firefox’ta da girildi ancak yine videonun karşı tarafa ulaşmaması hariç Firefox’ta daha farklı hatalarla karşılaşıldı. Karşılaşılan bu hataların çözülebilmesi için 3 gün boyunca internette sayısız site ve forumda araştırma yapıldı. Birçok farklı örnek incelendi. WebRTC ile ilgili temel birçok örnek barındıran web sitesinden [4] kodun bazı parçaları alınarak bizim kodumuzla harmanlandı. Çalışması beklenmesine rağmen çalışmadı. Bunun ardından 2 gün daha problemin çözümü için araştırma yapıldı ancak herhangi bir şeye ulaşılamadı. Bütün bunlar olurken Firefox tarayıcısında var olan problem hala devam ediyordu. Bu problem ile ilgili de araştırma yapıldı ancak bir şeye ulaşılamadı. Tam olarak nedeni anlaşılamayan bir şekilde birkaç değişiklikle problem çözüldü. Ancak başka bir sorun daha baş gösterdi: Arama yapanın Chrome, cevaplayanın Firefox olduğu durumda; Firefox tarayıcısındaki aynı sorun baş gösteriyordu. Ancak şu aşamada önemsenmeyerek atlandı. “Problem acaba, kodda bulunan oda mantığında bulunabilir mi?” diyerek koddaki oda mantığı silindi. Böylece uygulamaya giren herkes direkt olarak birbiriyle ilişki kurabilirdi. Ancak bu hiç çalışmadı. Bunun üzerine eski koda geri dönülüp IceCandidate kod bölümü tekrar gözden geçirildi. Çünkü problemin orada olması muhtemeldi. IceCandidate içerisinde Signal Server’a “room” bilgisinin gönderilmediği fark edildi. Bu bilgi gerekli yere eklendi ve Signal Server’dan diğer istemciye gelen IceCandidate bilgisi ilgili fonksiyonun yapıcısına doğru bir şekilde eklendi. Bunun sonucunda iki kişi, sorunsuz şekilde karşılıklı bağlantı kurabilir duruma ulaştı.  
   Ancak bu uygulamada sadece iki kişi karşılıklı iletişim kurabilmektedir. “Birden çok kullanıcı ile WebRTC nasıl kullanılır?” benzeri Google aramaları yapıldı. Görülen bütün örnekler üç ya da dört tane bilgisayar değişkeni oluşturup yine belirli sayıda kullanıcı arasında iletişim kurmaktaydı. Ancak malum ki bizce istenilen uygulama, sayısı değişmekle beraber çoklu katılımcı arasında iletişim kurmak. Daha önceden de yapıldığı gibi Youtube üzerinden bunla ilgili konferanslar ve dersler izlendi. Denk gelinen bir video [5]ile uçtan uca WebRTC’nin çoklu katılımcı durumunda pek verimli olmadığı görüldü. Çünkü her bir yayıncıdan çıkan video, yayındaki diğer bütün kullanıcılara ulaşıyor.
3. Mesh, MCU, SFU ile ilgili temel bilgiler edinilse de tam olarak ne oldukları anlaşılamamıştı. Şimdi ise videoların üzerinden bir kez geçilerek ve WebRTC alanında oldukça iyi bir Youtube kanalında MCU ve SFU ile ilgili video [6] izlenerek ne olduğu tam olarak anlaşıldı. Bir yayında 10 kişinin daha olduğu düşünülürse bir kişiye n2-1 tane video yayını gelmesi demek ki bu da hem ağı, hem işlemciyi yoran bir işlem. Bu yapıya Mesh ismi verilmektedir. Bunun için uçtan uca iletişim yerine ara katman olarak bir sunucu kullanılmaktadır. MCU ve SFU olarak iki farklı çeşidi vardır. MCU, her kullanıcıdan gelen görüntü-sesleri birleştirip tek bir parça haline getirerek her bir kullanıcıya sadece bir veri göndermekte ve bu birleşik yayınlar istemci tarafında açılmaktadır. Bu, her ne kadar internet dostu olarak görülse de istemci tarafında işlemciye çok büyük bir yük bindirmektedir. Hele ki kullanıcılar telefondan katılıyorlarsa telefonun kaynakları göz önüne alındığında çok büyük problemlere yol açacaktır. SFU sunucu; her kullanıcıdan gelen verileri sunucuda toplar. Daha sonra bunları teker teker diğer kullanıcılara dağıtır. Saf WebRTC’den daha düşük veri iletim sayısı gerektirmektedir. Ayrıca SFU, MCU’dan daha fazla tercih edilmektedir.  
     
   Aşağıdaki görselde solda bulunan yapı Mesh adı verilen klasik WebRTC iletişim sistemidir. Saf WebRTC ile iletişim kurulduğunda oluşacak olan durumdur. Önceki haftada da uygulaması yapıldığı üzere iki kişilik iletişim kolay olsa da bu sayı arttığında, görselde olduğu gibi, çok fazla ok işareti yani yayın bulunmaktadır. Görselin ortasında bulunan MCU sunucu yapısında görüldüğü üzere her kullanıcı için kullanıcı ile sunucu arasında çift taraflı tek bir ok bulunmaktadır. Bu da internet gereksinimini artırmasına rağmen işlemci gereksinimini artırmaktadır. Görselin sağında bulunan SFU sunucu yapısında görüldüğü üzere her kullanıcı ile sadece sunucunun ilişkisi bulunmaktadır. Her kullanıcı tarafından sunucuya bir yayın giderken, diğer kullanıcılardan her bir kullanıcıya diğer kullanıcı sayısı kadar yayın gelmektedir. 

Kaynak:[miro.medium.com/max/1000/0\*VketTspbtHNGP9ho.png](https://miro.medium.com/max/1000/0*VketTspbtHNGP9ho.png)  
  
Bütün bu karşılaştırmaların ve ayrıca başka araştırmaların da ışığında SFU’nun daha çok tercih edilen ve daha uygun bir sunucu olduğu kanısına varıldı. Bu demek oluyor ki projede medya sunucu olarak SFU kullanılacak. Birçok SFU medya sağlayıcı bulunmakta. Ancak hem proje için hem de ücretsiz bir şekilde yapabilmek için SFU’nun nasıl inşa edileceği araştırılmaya başlandı. Javascript tarafında Mediasoup kütüphanesinde rastlanıldı. Mediasoup’un kendi dökümantasyonu olsa da kolay anlaşılabilir değildi. Herhangi bir Türkçe ya da İngilizce kaynak da bulunmamaktadır. Ancak Youtube’da İngilizce bir Mediasoup videosuna [7] denk gelindi. İlk videoda Mediasoup’un ne olduğu ve nasıl çalıştığı anlatılırken diğer bir videoda ise Mediasoup’un uygulaması yapıldı [8]. Ancak anlatan kişinin bir akademisyen ya da eğitmen olması hasebiyle İngilizcesi tam olarak bizce anlaşılamadı. Bu yüzden videoya son verildi. Bunun ardından Youtube’da başka bir video eğitim serisine denk gelindi [9]. Bu seride Mediasoup ile çok basit bir görüntülü konuşma uygulaması yapılıyordu. Sekiz videoluk video serisinin yedinci videosuna gelinmişti ki videoyu hazırlayan kişi tarafından da Mediasoup’un karmaşık olduğu söylendi ki zaten Mediasoup’a azıcık bakılsa dahi bu, rahatlıkla anlaşılabilmektedir. Video serisini izlerken kodlamalar anlık olarak yapıldı ve Mediasoup genel olarak anlaşıldı. Bütün bunlardan sonra Medisoup’tan vazgeçme kararı alındı ve ilk olarak, duyulan ancak önem verilmeyen, GO dilinde yazılmış olan ION SFU paketine bakmaya karar verildi.

1. ION SFU daha derinlemesine araştırıldı. Mediasoup, metotlar sağlayarak bir SFU sunucu kurmayı sağlarken ION SFU direkt olarak bir SFU sağlar. Kodu indirip çalıştırmak yeterlidir. Her ne kadar hazır bir SFU olsa da, “Acaba ne kadar özelleştirilebilir?” sorusu akıllara geliyordu. Çünkü yayınları kaydetme, anlık olarak başka platformlara yayın yapma, Javascript’e GO üzerinden API açma gibi özellikler eklemek zor olabilirdi. Bütün bunlar görüldüğünde “Mediasoup’a geri mi dönsek?” sorusu akıllara geldi ve bunun üzerine yine uzun süre düşünüldü ve araştırma yapıldı. Ancak Mediasoup’un örnek demo uygulamasına bakıldığında pek de stabil aktarım yapamadığı görülünce bu fikirden vazgeçildi ve ne olursa olsun ION SFU ile gidilmeye karar verildi. ION ve Mediasoup’a alternatif başka açık kaynak sunucular da var ancak GO’ya daha fazla aşina olunması sebebiyle ION seçildi. Yoksa özellikle Jitsi, oldukça yaygın ve kaliteli bir SFU sunucudur. Özelleştirme ile ilgili kaygılar hakkında ne yapılabilir diye bakıldı. Ancak internette ION SFU hakkında çok bir bilgi olduğu söylenemez. Bu yüzden bunlar bir süreliğine göz ardı edilerek ION SFU, Github hesabından bilgisayara klonlandı. Youtube’da Medisoup video serisinin izlendiği aynı kanaldan ION SFU ile bire çok görüntülü konuşma uygulamasının nasıl yapıldığı izlendi[10]. ION SFU şöyle çalışmaktadır: ION SFU zaten sabit bir sunucudur. Sunucudaki metotlar, fonksiyonlar istemci tarafından RPC yardımıyla çağrılır. RPC, istemci tarafından sunucudaki bir fonksiyonu tetiklemek anlamına gelmektedir. ION’un tarayıcılar ve Javascript için ion-sdk-js adında bir SDK’si bulunmaktadır. Bunu kullanarak ION SFU kullanılır. Az önce belirtilen video izlendikten sonra çoka çok yani aynı anda birden çok kullanıcının yayın yapabildiği görüntülü konuşma uygulamasının yapımı izlendi [11]. Bütün bunlardan sonra “iletişim kısmı temel olarak tamamdır” diyerek ION SDK’nın istemci tarafında kullanılacak olan Javascript framework’ü VueJS’te kullanımı izlendi[12].
2. Projeye başlama kararı verildi. Bunun için, “kullanıcı uygulamaya girdiği zaman nasıl bir ekranla karşılaşacak? Neye tıkladığı zaman nasıl bir karşılık alacak? Uygulama akışı nasıl olacak?“ gibi sorulara cevap bulundu ve not edildi. Geçen haftalarda da bahsedildiği üzere ION SFU hazır bir sunucu olduğu için özelleştirmesi de kısıtlıydı. Bunun için SFU sunucunun ayrı, diğer işlemler için gereken sunucunun ayrı olmasına karar verildi. Belki ileriki süreçte bu iki sunucu API ile iletişim kurabilir. Bu nedenle projeye ilk olarak sunucu tarafını inşa ederek başlamak gerekiyordu. Bunun için bir Javascript framework’ü olan Express tercih edilecekti. Ancak önceki bilgilere de dayanılarak NestJS adlı diğer bir framework araştırıldı. Bunun sonucunda, NestJS’in, ExpressJS ve Fastify adı verilen iki framework’ü beraber kullanan bir framework olduğu görüldü. NestJS ile birlikte Fastify kullanmak, Express kullanmaktan yaklaşık iki kat hız artırışına neden olacaktı. Bu nedenle NestJS’nin kendi dökümantasyonu [13] üzerinden NestJS öğrenilmeye başlandı. Bu arada NestJS, Typescript kullanan bir framework olduğu için uzun süredir kullanılmamış ve unutulmaya yüz tutmuş olan Typescript, kendi dökümantasyonu [14] üzerinden tekrar edildi. Ayrıca NestJS’in her yerinde kullanılan Decorator’ler araştırıldı. Daha önce hiç tecrübe edilmemiş olan Decorator’ler ile ilgili Python dilinde oldukça fazla kaynak bulunduğu için Python’da Decorator kullanımı araştırıldı ve Typescript’te kullanımı ile bağdaştırıldı. Çok ayrıntılı olarak girilmese de temel olarak ne olduğu açıklığa kavuşturuldu. NestJS dökümantasyonuna geri dönüldü ve okunmaya devam edildi. Ancak gerek İngilizce dil yeterliliğinin zaman zaman yetersiz kalması, gerek kavramlara yabancılık gibi nedenlerden ötürü Youtube üzerinden eğitimi için bakıldı. Elbette tahmin edildiği üzere tatmin edici bir Türkçe kaynak bulunamadı. Bunun üzerine İngilizce bir video serisi[15] izlenmeye başladı. Dökümantasyon okunduktan sonra videoların izlenmesi işleri gayet kolaylaştırdı. Serinin yaklaşık 1/3’ü bittikten sonra izlemeye ara verildi. Dökümana geri gidildi. Birkaç başlık daha okundu ve son okunan yerler ile ilgili Youtube’da video arayışına girildi. Bunun sonucunda AngularJS ve NestJS ile bir blog sitesi uygulaması yapan bir video serisine [16] rastlandı. NestJS’te pratik yapmak, gerçek hayat kodunu görebilmek için izlenmeye başlandı.
3. Hem NestJS’nin kendi dökümantasyonunda hem de geçen hafta bahsedilen video serisinde görüldü ki NestJS’te, RxJS kütüphanesinden aktarılan Observable adı verilen bir nesne çok kullanılmaktadır. Bunun üzerine RxJS üzerine Youtube’da video arandı. Bunun sonucunda Türkçe bir RxJS video serisine [17] denk gelindi. Ne olduğu ve fikir vermesi açısından birkaç video izlendi. RxJS, reaktif programlama sağlayan bir kütüphanedir. Prosedürel programlamada, kod çalıştırılır ve satır satır okunarak istenen işlemler yapılır. İki değişken toplanıp bir değişkene atandığında ve daha sonra toplanan değişkenlerden birinin değeri değiştirildiğinde, toplamanın sonucunun atandığı değişkenin değeri değişmez. Reaktif programlama tam olarak bunu tersini sağlar. Verileri, yaşam süreleri boyunca takip eder. Değerleri değiştiğinde, kendisi ile ilgisi olan verileri de değiştirir. Blog uygulaması yapan video serisine devam edildi. Dördüncü videoya gelindikten sonra, artık projeye temel de olsa başlanılması gerektiği düşünüldü. Ancak iki engel vardı. Projede REST yerine GraphQL kullanılmak isteniyordu. İkinci olarak ise anlık iletişim için kullanılacak olan Websocket’lerin NestJS’te kullanımına bakılacaktı. GraphQL’i öğrenebilmek için arayışa girildi. GraphQL hakkında temel seviye bilgiye sahip olunduğu düşünülerek NestJS’te GraphQL kullanımı için Youtube’dan video izlendi. İzlenmek için karar kılınan videonun [18] yarısına gelindiğinde dahi akıllarda REST ile GraphQL’in farkı tam olarak oturmuyordu. Bunun için o videolardan ayrılınarak GraphQL’i daha iyi öğrenmek için arayışa girildi. Youtube’da Türkçe bir Meetup videosuna[19] denk gelinerek GraphQL hakkında teorik bilgi edinildi. REST’ten ayrıştığı noktalar daha iyi anlaşılmış oldu. Bunun ardından birkaç küçük daha İngilizce video izleyince GraphQL genel hatlarıyla netleşti. Yarıda bırakılan NestJS ile GraphQL videosuna[18] geri dönülerek videonun tamamı bitirildi. Bu arada NestJS’in kendi dökümanına gidilerek GraphQL kullanım bölümü [20] de tüketildi. Dökümantasyonda GraphQL için NestJS’nin önerdiği Apollo kütüphanesine [21] kısa bir göz atıldı. Anlaşıldığı kadarıyla Apollo temel olarak, GraphQL sorguları için oldukça zor olan Cache sistemini ve data işlemlerini kolaylaştırmaktadır. Daha sonra GraphQL’i pratikte daha iyi anlayabilmek için bir tane daha GraphQL videosu[22] izlendi. Projede veritabanı olarak MongoDB kullanılacağı için NestJS’te MongoDB kullanımı için NestJS’nin kendi dökümantasyonundaki ilgili bölüm[23] okundu. Pratikte kullanımını görebilmek için MongoDB ve NestJS kullanımı[24] izlendi. Proje boyunca anlık mesajlaşma ve iletişim için Websocket kullanılması düşünülüyordu. Ancak bundan önce Websocket’in alternatifleri araştırıldı. Buna rağmen websocket’in işlevini tam olarak yerine getirebilen bir şeye rastlanamadı. Bunun için Websocket yerine WebRTC’nin RTCDataChannel [25] objesi ile anlık mesajlaşma ve her türlü veri alışverişinin yapılmasına karar verildi.  
   MongoDB ve GraphQL için yeterli bilgiye sahip olunduğu düşünülerek projeye başlamaya karar verildi. İlk olarak projeyi tamamen Docker üzerinde inşa etme fikri gözden geçirildi ve denendi. Ancak daha sonra projeyi bitirip arından Docker’a kurmaya karar verildi. Böylece projenin özellikle sürdürülebilirdiği ve taşınabilirliği kolaylaşacak. NestJS CLI komutuyla proje başlatıldı. Projede RESTful yerine GraphQL tekniği kullanılacağı için projedeki, REST için kullanılan ‘app.controller.ts’ dosyası ve mantıksal işlemlerini yapacak olan ve ana klasörde bulunan ‘app.service.ts’ dosyası silindi. Çünkü proje kendi içinde temel olarak ve şuan için öngörülebilen ‘User’, ‘Validation’ ve ‘Room’, ‘Class’ modüllerinden oluşacak. Bu modüllerin de kendi gerekli dosyaları bulunmaktadır. Ana klasördeki ‘app.module.ts’ dosyası dışında temel bir dosyaya ihtiyaç duyulmamaktadır. İlk olarak User modülü oluşturulmakla başlandı. NestJS’nin kendi CLI komutları ile otomatik olarak modül klasörü ve içinde gerekli dosyaları oluşturuldu. Her oluşturulan modülde RESTful kullanılacağı varsayılarak ‘Controller’ dosyası da gelmektedir ancak proje boyunca ‘Controller’ dosyaları kullanılmayacaktır. Graphql’in istekleri kontrol eden dosyasına ‘Resolver’ adı verilmektedir. Bu yüzden ‘User’ modülü için ‘user.resolver.ts’ dosyası yine CLI aracılığıyla oluşturuldu. GraphQL ve MongoDB entegrasyonları yapıldı. GraphQL’de kod yazma işlemi ilk kez deneyimlendiği için uzun süreler alsa da sonunda, veritabanında kayıtlı tüm kullanıcıları getirme ve veritabanına yeni kullanıcı kaydetme işlemi tamamlandı. Geçen haftalarda bahsedilen ve NestJS ile Blog uygulaması yapan video serisi [16] izlenmeye baştan başlandı. Böylece daha emin adımlarla, daha doğru yollarla ve yeni bilgiler ve yöntemler öğrenilerek proje devam etmiş olacak. Ayrıca proje, Github platformunda “Private Repository” olarak tutulacak. User Modülünde temel CRUD işlemleri yeni kullanıcı sorgulama, oluşturma, güncelleme, silme işlemleri ve bu işlemlerin veritabanı ile bağlantısı yapıldı. Kullanıcı tarafından, kullanıcı oluşturma, güncelleme gibi işlemler ve ayrıca diğer tüm kullanıcı taraflı giriş isteyen alanlara temel seviye bir doğrulayıcı eklendi. Girilen bilgi metin mi, tamsayı mı, doldurulmuş mu, “email” mi gibi doğrulamaları yapmaktadır.  
     
   
4. Temel seviyede kullanıcı giriş işlemi tamamlandı. Kullanıcı giriş yapmak için sunucuya email adresi ve parolasını gönderdiğinde eğer bu bilgilerle bir kullanıcı veritabanında kayıtlıysa kullanıcıyı geri döndürüyor ve ayrıca bir de bir “accessToken” ekliyor ki bu token, Javascript aleminde oldukça yaygın kullanılan JWT tipinde oluyor. JWT, içine bilgileri eklenerek şifrelenen bir token. Bu tokene kullanıcının ad, soyad, email ve rol bilgileri eklenerek istemciye gönderilecek Kullanıcı, yetki gerektiren işlemler ve sayfalarda bu token yardımıyla gezebilecek. Eğer kullanıcının token süresi dolmuş ise, geçersiz ise ya da yoksa kullanıcı giriş yapamayacak. Bu token istemcinin tarayıcısında depolanacak. Kullanıcı sisteme kayıt olmak için istek attığında gerekli tüm bilgileri dolduracak register endpoint’i veritabanına kullanıcıyı kaydettikten sonra kendiliğinden login işlemini gerçekleştirecektir.  
   metin içeren bir resim

   Açıklama otomatik olarak oluşturuldu  
     
   metin içeren bir resim

   Açıklama otomatik olarak oluşturuldu  
   Javascript’te objeler referans veri tipleridir. Yani bir fonksiyona parametre olarak bir obje geçirirseniz, fonksiyon içindeyken objede yaptığınız her değişiklik orijinal objeye de yansıyacaktır. Kayıt işleminde, “böyle bir kullanıcı mevcut mu?” sorusu için şifre karşılaştırmasında şifreler yanlış çıkmaktaydı. Uzunca süren denemeler sonucunda problemin, objenin doğasından kaynaklandığı anlaşıldı. Endpoint’e gelen “Body” objesini referans kırmadan direkt “createUser” fonksiyonuna geçirildiği için bu hata ile karşılaşılmaktaydı. “Hash”lenen parola, orijinal parola ile yer değiştiriyordu.  
   Kayıt olurken girilen email adresine sahip başka bir kullanıcı varsa, “böyle bir kullanıcı mevcut” hatasını istemciye gönderilme işlemi yapıldı. metin içeren bir resim

   Açıklama otomatik olarak oluşturuldu   
   Kullanıcının belirli endpoint’lere sadece giriş yapmışsa istek atabilmesi için üretilen JWT access\_token’in her istekten önce çözümlenip JWT içindeki bilgiye sahip bir kullanıcının sistemde var olup olmadığının sorgulanması gerekmektedir. Doğrulama işlemleri için NestJS’te var olan bir yöntem mevcuttur: Guards. Atılan istek, ilgili endpoint’e ulaşmadan hemen önce çalışır ve gerekli işlemler orada gerçekleştirilebilir. Guards ise genellikle koruma işlemleri için kullanılır. JWT tokenin çözümlenmesi ve Request objesine eklenmesi bu aşamada yapılacaktır. JWT’nın kendi kütüphanesi yerine NestJS’nin önerdiği Passport modülü kullanılarak yapıldı. Request ile Header içinde Authorization alanı olarak gönderilen ‘Berarer $token’ değişkenini çözümleyecek ve eğer JWT token, geçerli bir token ise ve süresi dolmamışsa geçişe izin verecek. Ancak bu kullanıcı mevcut mu sorgulamasını yapmayacak. Bu işlemlerin böyle olması gerekiyordu. Bu işlem için iki adet dosya gerekiyor: JWT çözümleme işlemi için ‘jwt.strategy.ts’ ve Guard işlemi için ‘jwt.guard.ts’. Bunun nasıl yapıldığı ve Guard’ın nasıl çalıştığının anlaşılması için birçok video izlendi. Zaten takip edilen video serisinin[16] videosunda bunun nasıl yapıldığı anlatılıyordu. Ancak video serisi RESTful olarak yapıldığı için ve RESTful ve GraphQL farklı şekilde çalışıtğı için orada çalışan kod burada çalışmayacaktı ve çalışmadı. Bunun için birçok farklı video izlendi. Uygulandı ama bir türlğ çalışmadı. Atılan istek Guard’dan hiçbir zaman geçemiyor ve uygulama hata veriyordu.
5. Guard hatasının çözümü için farklı yollara başvuruldu. Github’tan başka kişilerin kodları indirilerek uygulama ile entegre edilmeye çalışıldı. Yine olmadı. Orijinal proje dışında rahatça denemeler yapmak için proje dosyası kopyalandı ve onun üzerinde denemeler yapıldı, başka kodlar denendi. Olumlu sonuç alınamadı.Elbette bu süreç bir umutsuzluk, zaman kaybı getirdi.
6. Hiçbir değişiklik yapılmadan Guard süreci tekrarlandı. Ancak farkedildi ki test etme sürecinde bir hata yapılıyor. JWTAuthGuard adlı JWT çözümleme Guard’ı yanlış endpoint’te kullanılıyor. Halbuki kullanıcı giriş yaparken kullanıcıdan herhangi bir token istenmemesi gerek çünkü zaten giriş yapmamış ki token mevcut olsun. Kullanıcı giriş yaptıktan sonra istek atarken giriş istenen adreslerde kullanılması gerek. Bu anlaşıldıktan sonra yetki gerektiren kullanıcıları listeleme, kullanıcı ekleme, kullanıcı silme, kullanıcı gücelleme ve bunlar gibi işlemlerin önüne bu Guard konuldu. Ve GraphQL ile deneme yapılmaya girildi. Bir kullanıcı oluşturuldu. Daha sonra bu kullanıcı giriş yaptırıldı. Bunun sonucunda bir ‘access\_token’ elde edildi. Bu token kopyalanarak isteğin Header kısmına “’Authorization’: ‘Bearer $token’” olarak eklendi ve kullanıcı listele isteği atıldı ve çalıştı. Haftalarda denenip çalışmayan problem aslında herhnagi bir hatadan değil hatalı test yapıldığı için kaynaklanıyormuş.  
   ‘User’ modülü içinde bulunan ‘login’ ve ‘validateUser’ metotları ve ‘login’ resolver’i Auth modülü içine taşındı.  
   Bir kullanıcı kayıt olurken eğer mevcutsa, ‘bu kullanıcı zaten mevcut’ mesajının olduğu kod parçası silindiği için tekrardan yazıldı.
7. Kullanıcı sisteme kayıt olduğu zaman email adresinin geçerli ve çalışan bir email adresi olduğunun anlaışması için kayıt işleminin hemen ardından email adresine bir doğrulama maili gönderilme işlemi yapıldı. Bunun için kullanıcıya özel bir Hash’lenmiş değer(token) üretildi. Sadece mail token’leri için MongoDB üzerinde bir collection(table) oluşturuldu ve her token kullanıcının ID bilgisi ile ilişkilendirildi. Bu token uygun linkin sonuna eklenerek kullanıcının email adresine bir bilgilendirme mesajı gönderildi ve o gönderilen linke tıklaması söylendi.  
   Mesajda her ne kadar makyaj problemleri olsa da bunlar ileriki süreçte halledilecek. Kullanıcı linke tıkladıktan sonra uygulamanın front-end’inde ilgili yere yönlendirilecek. Front-end üzerinden de uygulamanın back-end’ine istek atılacak ve yukarıdaki görselde bulunan link ayrıştırılarak içinden token alınacak. Token, veritabanında kayıtlı mı diye bakılacak. Eğer sistemde böyle bir token varsa kullanıcının, veritabanında default olarak ‘Pending’ şeklinde duran status bilgisi ‘Active’ durumuna çekilecek. Email adresini doğrulamayan yani status bilgisi Pending olan kullanıcılar sistemde belirli özelliklere erişemeyecek. Ardından Token, yer kaplamaması için veritabanı üzerinden silinecek.  
   Bu token işlemi için Token, JWT kütüphanesi ile üretilmişti ancak bundan vazgeçildi ve Hash fonksiyonu ile kullanıcının email adresi imzalanarak(hash) oluşturuldu.  
   Bir kullanıcının temel işlemleri bittiği için artık “room” modülü oluşturuldu. Oda oluşturma ve listeleme sorguları yapıldı. Test edilmesi için GraphQL test arayüzüne girildi. Ancak farkedildi ki User ve Room modülü içindeki fonksiyonların adları aynı. Yani REST API’de ‘user/’ adresine istek atılarak kullanıcılar listelenirdi. Oda listelemek için ‘room/’ adresine istek atılırdı. Ancak GraphQL’de her ikisi içinde getAll adlı bir fonksiyon var ve birbirinden ayrıştırılabilecek bir yanları yok. Saf Javascript’te GraphQL’in bunun için bir çözümü var. Room ve User için isim belirlenebiliyor. Ancak araştırmalar sonucu NestJS’te böyle bir şeyin mevcut olduğunda rastlanmadı. Forumlada, çok zorlayarak ve çok da garip yollarla yapan kullanıcılara rastlandı. Ancak böyle bir şey yapılmak istenmedi. Bunun için GraphQL’den RESTful’a geçme kararı verildi. GraphQL için hazırlanan bütün dosyalar silindi. Gerekli dosyaların yerleri ve isimleri değiştirilid. Resolver dosyaları(GraphQL isteklerinin geldiği uç noktaları tutan ve yöneten dosya) silinerek Controller dosyaları(Rest API isteklerinin geldiği uç noktaları tutan ve yöneten dosya) oluşturuldu. Gerekli bütün düzenlemeler yapıldıktan bir süre sonra artık sistem RESTful olarak çalışabilir hale geldi. Room modülü içinde gerekli CRUD(od oluşturma, getirme, güncelleme, silme) işlemleri yapıldı. JWTAuth Guard koruması oda işlemlerinin her birine eklendi çünkü giriş yapmayan birisi bunları yapamamalı.

# Kaynaklar

1. https://www.youtube.com/googlecode
2. https://www.[youtube.com/watch?v=t7VcK7y3Olg](https://www.youtube.com/watch?v=t7VcK7y3Olg)
3. https://www.developer.mozilla.org/en-US/docs/Web/API/WebRTC\_API/Connectivity
4. <https://github.com/webrtc/samples/blob/gh-pages/src/content/peerconnection/channel/js/main.js>
5. <https://www.youtube.com/watch?v=Y1mx7cx6ckI>
6. <https://www.youtube.com/watch?v=tHHpLraJKpQ>
7. <https://www.youtube.com/watch?v=DOe7GkQgwPo>
8. <https://www.youtube.com/watch?v=FLxU6ftLJsE>
9. <https://www.youtube.com/playlist?list=PLa2dCM6b-1M5TZu-9_s-OFuuqo22GeUJq>
10. <https://www.youtube.com/watch?v=ZIwW82Mx3Rg>
11. <https://www.youtube.com/watch?v=HMxjVdek7Wg>
12. <https://www.youtube.com/watch?v=XomegdckYXg>
13. <https://docs.nestjs.com>
14. <https://www.typescriptlang.org/docs>
15. <https://www.youtube.com/playlist?list=PL_cUvD4qzbkw-phjGK2qq0nQiG6gw1cKK>
16. <https://www.youtube.com/playlist?list=PLVfq1luIZbSmJIuw_EZVP9mFiMED5fGIn>
17. <https://www.youtube.com/playlist?list=PLQVXoXFVVtp1v1_D_8ocGOsWFGvK1Ha-E>
18. <https://www.youtube.com/watch?v=0WgO3-HVH94>
19. <https://www.youtube.com/watch?v=s90xJgr-w6Y>
20. <https://docs.nestjs.com/graphql/quick-start>
21. <https://www.apollographql.com>
22. <https://www.youtube.com/watch?v=geYvdbpo3cA>
23. <https://docs.nestjs.com/techniques/mongodb>
24. <https://www.youtube.com/watch?v=ulfU5vY6I78>
25. <https://www.html5rocks.com/en/tutorials/webrtc/datachannels>