
**T.C**
**DOKUZ EYLÜL ÜNİVERSİTESİ**
**FEN FAKÜLTESİ**
**BİLGİSAYAR BİLİMLERİ BÖLÜMÜ**

# BULUT BİLİŞİM PROJESİ:BURADAA

**Soner VATANSEVER**

**Alican ERDURMAZ**

**Bayram GÜN**

**Umut ÖZKAYA**

**Danışman: Dr. Öğr. Üyesi Can ATILGAN**

**Haziran, 2021**

**İZMİR**

**TEŞEKKÜR**

Bu proje üzerindeki çalışmalarımızda her zaman bize destek olan : Dr. Öğr. Üyesi Can ATILGAN'a teşekkürlerimizi sunuyoruz.

# İÇİNDEKİLER

## ŞEKİLLER TABLOSU

## ÖZET

Buradaa, derslerde yoklama alınırken oluşan zaman kaybını, malzeme gereksinimliğini azaltan ve derslerde öğrencilerin başkalarının yerine attıkları imzaları engellemeyi mobil cihazlar aracılığıyla kolaylaştırılması ve takibinin otomatikleştirilmesini amaçlar. Ayrıca öğrencilerin anlık olarak ders takibini, devamsızlık bilgisini, ders programını ve saatlerini gözlemleyebilmesini sağlar. Bunun yanı sıra öğretmenin işlemlerini kolaylaştıran ve bu süre zarfında harcanan enerjinin azaltmasını hedefler.

Buradaa; mobil uygulaması için React-Native kullanılarak çapraz platform desteği, web servis tarafı için NestJS,GraphQL , web arayüzü için React, veri tabanı için PostreSQL



**Anahtar Kelimeler:**

Çapraz platform, Mobil aplikasyon, Web uygulaması, bulut platform, GPS kontrol, hız

## ABSTRACT

Buradaa, aims to prevent the loss of time and material requirements while taking attendance in the lessons, and to prevent the signatures of the studens on behalf of others in the lessons, to facilite automate the follow up through mobile devices, In addition, it allows students to instantly follow lessons,absenteeism information, lesson schedule and hours. In addition it aims to facilate the teachers operations and to reduce the energy spent during this time.

Buradaa, Cross platform support using React-Native for mobile app, NestJS,GraphQL,PostgreSQL for web service side, React for web interface, PostgreSQL for database



**Keywords:**

Cross platform, Mobile app, Web app, cloud platform, Gps control, speed

# 1. GÖZLENEN PROBLEMLER VE ÇÖZÜM YÖNTEMLERİ

Günümüzde mobil uygulama, web uygulama alanlarında yaşanan teknolojik gelişmeler ile birlikte dijital platformlar artık hayatımızın her alanında büyük bir yer kaplamaktadır, bu yüzden tüm sektörlerin dijitalleşme alanına yönelmek amacıyla adımlar atarak, gelişen teknolojiye adapte olmak değerlerini arttırmaya yönelik çalışmalar yapılması zorunlu bir hale gelmiştir.

Bu nedenle bilgi edinme yollarının sanal ortamlara taşındığı günümüzde eğitim sektöründe yer alan ve sürekli artmaya devam eden üniversitelerin bilgi işleme süreçlerinde yaşadığı karmaşıklık ve iş yükü gözlemlenmiştir. Öğretmen, bilgi işlem ve öğrenci arasındaki iş birliğinin düşük seviyede olması gözlemlenen bir diğer problemdir.

İlgili üniversitelerin yerleşkelerindeki verilerin tek bir uygulamada toplayarak bu verileri daha kolay işleyip, takibini kolaylaştırarak bilgi işlemi, öğretmen ve öğrenci arasındaki iletişimin artması sağlanacaktır. Bilgi işlem Buradaa web uygulaması ile birlikte sınıf, öğrenci ve öğretmenleri daha az iş yükü ile ekleyebilecek, öğretmen derslerini Buradaa mobil uygulaması ile daha rahat gözlemleyip ders sırasında yoklama işlemini ve sınıf içi yoklama takibini daha kolay gerçekleştirecektir. Öğrenciler Buradaa mobil uygulaması ile ders saatlerini ve devamsızlık haklarını uygulama üzerinden istedikleri anda kolaylıkla gözlemleyebileceklerdir. Öğrenciler Buradaa mobil uygulması ile haftalık ders programını, ders saatini ve dersin bulunduğu dersliği ve ders sırasında yoklama işlemini arayüz üzerinden sağlayabilecektir. Ayrıca bilgi işlem yönetimsel süreçlerin yanı sıra öğrencilerin bilgilerini rahatlık ile gözlemleyebilcektir.

### 1.1 Bulut Bilişim Nedir?

**[1]** Bulut bilişim, internet üzerinden, bulut servisleri tarafından yönetilen uzak bir veri merkezinde barındırılan, uygulamalar, sunucular (fiziksel sunucular ve sanal sunucular), veri depolama, geliştirme araçları, ağ oluşturma yetenekleri ve daha fazlası gibi bilgi işlem kaynaklarına isteğe bağlı erişimdir. sağlayıcı (veya CSP). CSP, bu kaynakları aylık bir abonelik ücreti karşılığında kullanıma sunar veya kullanıma göre faturalandırır.

Geleneksel şirket içi BT ile karşılaştırıldığında ve seçtiğiniz bulut servislerine bağlı olarak, bulut bilişim aşağıdakileri yapmanıza yardımcı olur:

Daha düşük BT maliyetleri: Bulut, kendi şirket içi altyapınızı satın alma, yükleme, yapılandırma ve yönetme maliyetlerinin ve çabalarının bir kısmını veya çoğunu boşaltmanıza olanak tanır.

Bulut sayesinde kuruluşunuz, BT'nin bir isteğe yanıt vermesini, destekleyici donanımı satın alıp yapılandırmasını ve yazılım yüklemesini haftalar veya aylarca beklemek yerine, kurumsal uygulamaları dakikalar içinde kullanmaya başlayabilir. Bulut ayrıca belirli kullanıcıları, özellikle geliştiricileri ve veri bilimcilerini, yazılım ve altyapıyı destekleme konusunda kendilerine yardımcı olmaları için güçlendirmenize olanak tanır.

Bulut, esneklik sağlar; yavaş dönemlerde kullanılmayan fazla kapasiteyi satın almak yerine, trafikteki ani artış ve düşüşlere yanıt olarak kapasiteyi artırıp azaltabilirsiniz. Uygulamalarınızı dünyanın her yerindeki kullanıcılara daha yakın bir şekilde yaymak için bulut sağlayıcınızın küresel ağından da yararlanabilirsiniz.

'Bulut bilişim' terimi, bulutun çalışmasını sağlayan teknolojiyi de ifade eder. Bu, bir tür sanallaştırılmış BT altyapısını (sunucular, işletim sistemi yazılımı, ağ iletişimi ve özel yazılımlar kullanılarak soyutlanan diğer altyapılar) içerir, böylece fiziksel donanım sınırlarına bakılmaksızın bir havuzda toplanabilir ve bölünebilir. Örneğin, tek bir donanım sunucusu birden çok sanal sunucuya bölünebilir.

Sanallaştırma, bulut sağlayıcılarının veri merkezi kaynaklarından maksimum düzeyde yararlanmalarını sağlar. Şaşırtıcı olmayan bir şekilde, birçok şirket, geleneksel BT altyapısına kıyasla maksimum kullanım ve maliyet tasarrufu gerçekleştirebilmek ve son kullanıcılarına aynı self servis ve çevikliği sunabilmek için şirket içi altyapıları için bulut dağıtım modelini benimsemiştir.

Evde veya işte bir bilgisayar veya mobil cihaz kullanıyorsanız, Google Gmail veya Salesforce gibi bir bulut uygulaması, Netflix gibi medya akışı veya Dropbox gibi bulut dosya depolaması olsun, neredeyse kesinlikle her gün bir tür bulut bilgi işlem kullanırsınız. Yakın tarihli bir ankete göre, kuruluşların %92'si bugün bulut kullanıyor ve çoğu önümüzdeki yıl içinde bulutu daha fazla kullanmayı planlıyor.

**Bulut Bilişim Ne Zaman Yaygınlaşmıştır?**

Konsept teknik olarak 1990'ların ortalarında ortaya çıkarken, bulut bilişim 2000'li yılların ortalarından itibaren "büyük 3"ün ortaya çıkmasıyla ana akım haline geldi: Google Cloud Platform, Microsoft Azure ve Amazon Web Services (AWS). Bu modern kaynak paylaşımı anlayışı iş düşünülerek yapılmış olsa da, bulut bilişim, 2014 yılında AWS Lambda'nın sunucusuz bilişimi tanıtmasıyla başlayarak, geliştiricilere giderek daha fazla hizmet vermeye başladı. [2]

### 1.2 Bulut Bilişim Servisleri

Servis odaklı mimari, "Her şey servis olarak" (EaaS veya XaaS, [2] veya basitçe aas kısaltmalarıyla) savunsa da, bulut bilişim sağlayıcıları "servislerini" farklı modellere göre sunarlar; bunlardan NIST'e göre üç standart model Servis Olarak Altyapı (IaaS), Servis Olarak Platform (PaaS) ve Servis Olarak Yazılım (SaaS)'dir. [3] Bu modeller artan bir soyutlama sunar; bu nedenle genellikle bir yığındaki katmanlar olarak tasvir edilirler: altyapı, platform ve servis olarak yazılım, ancak bunların birbiriyle ilişkili olması gerekmez. Örneğin, temel PaaS veya IaaS katmanları kullanılmadan fiziksel makinelerde uygulanan SaaS sağlanabilir ve tersine bir program IaaS üzerinde çalıştırılabilir ve SaaS olarak doğrudan erişilebilir.

**Servis Olarak Altyapı**

"Servis olarak altyapı" (IaaS), fiziksel bilgi işlem kaynakları, konum, veri bölümleme, ölçekleme, güvenlik, yedekleme vb. gibi temel ağ altyapısının çeşitli alt düzey ayrıntılarını soyutlamak için kullanılan üst düzey API'ler sağlayan çevrimiçi servisler anlamına gelir. hypervisor sanal makineleri misafir olarak çalıştırır. Bulut işletim sistemi içindeki hiper yönetici havuzları, çok sayıda sanal makineyi ve servisleri müşterilerin değişen gereksinimlerine göre yukarı ve aşağı ölçekleme yeteneğini destekleyebilir. Linux kapsayıcıları, doğrudan fiziksel donanım üzerinde çalışan tek bir Linux çekirdeğinin yalıtılmış bölümlerinde çalışır. Linux grupları ve ad alanları, kapsayıcıları izole etmek,

güvenli hale getirmek ve yönetmek için kullanılan temel Linux çekirdek teknolojileridir. Konteynerleştirme, hiper yönetici ek yükü olmadığından sanallaştırmadan daha yüksek performans sunar. IaaS bulutları genellikle sanal makine disk görüntü kitaplığı, ham blok depolama, dosya veya nesne depolama, güvenlik duvarları, yük dengeleyiciler, IP adresleri, sanal yerel alan ağları (VLAN'lar) ve yazılım paketleri gibi ek kaynaklar sunar.

[4]NIST'in bulut bilişim tanımı, IaaS'yi "tüketicinin, işletim sistemlerini ve uygulamaları içerebilen keyfi yazılımları dağıtabildiği ve çalıştırabildiği yer" olarak tanımlar. Tüketici, temel bulut altyapısını yönetmez veya kontrol etmez, ancak işletim sistemleri, depolama ve depolama üzerinde kontrole sahiptir.

**Servis Olarak Platform**

NIST'in bulut bilişim tanımı, platformu bir servis olarak şu şekilde tanımlar:

[3] Servis ağlayıcı tarafından desteklenen programlama dilleri, kitaplıklar, servisler ve araçlar kullanılarak oluşturulan, tüketici tarafından oluşturulan veya satın alınan uygulamaların bulut altyapısına dağıtılmasıdır. Tüketici, ağ, sunucular, işletim sistemleri veya depolama dahil olmak üzere temel bulut altyapısını yönetmez veya kontrol etmez, ancak dağıtılan uygulamalar ve muhtemelen uygulama barındırma ortamı için yapılandırma ayarları üzerinde kontrole sahiptir.

PaaS satıcıları, uygulama geliştiricilerine bir geliştirme ortamı sunar. Sağlayıcı genellikle geliştirme için araç seti ve standartlar ve dağıtım ve ödeme kanalları geliştirir. PaaS modellerinde bulut sağlayıcıları genellikle işletim sistemi, programlama dili yürütme ortamı, veritabanı ve web sunucusu içeren bir bilgi işlem platformu sunar. Uygulama geliştiricileri, temel donanım ve yazılım katmanlarını doğrudan satın almak ve yönetmek yerine yazılımlarını bir bulut platformunda geliştirir ve çalıştırır. Bazı Paa'larda, temel bilgisayar ve depolama kaynakları uygulama talebine uyacak şekilde otomatik olarak ölçeklenir, böylece bulut kullanıcısının kaynakları manuel olarak ayırması gerekmez. [5]

**Yazılım Olarak Servis**

Bulut altyapısı üzerinde çalışan uygulamaların kullanabilmesidir. Uygulamalara, bir web tarayıcısı (örneğin, web tabanlı e-posta) gibi bir ince istemci arabirimi veya bir program arabirimi aracılığıyla çeşitli istemci cihazlarından erişilebilir. Tüketici, sınırlı kullanıcıya özel uygulama yapılandırma ayarları istisnası dışında, ağ, sunucular, işletim sistemleri, depolama ve hatta bireysel uygulama yetenekleri dahil olmak üzere temel bulut altyapısını yönetmez veya kontrol etmez.

Servis olarak yazılım (SaaS) modelinde, kullanıcılar uygulama yazılımlarına ve veritabanlarına erişim kazanır. Bulut sağlayıcıları, uygulamaları çalıştıran altyapıyı ve platformları yönetir. SaaS, bazen "isteğe bağlı yazılım" olarak adlandırılır ve genellikle kullanım başına ödeme temelinde veya bir abonelik ücreti kullanılarak fiyatlandırılır.[6]SaaS modelinde, bulut sağlayıcıları bulutta uygulama yazılımını kurar ve çalıştırır ve bulut kullanıcıları yazılıma bulut istemcilerinden erişir. Bulut kullanıcıları, uygulamanın çalıştığı bulut altyapısını ve platformu yönetmez. Bu, uygulamayı bulut kullanıcısının kendi bilgisayarlarına kurma ve çalıştırma ihtiyacını ortadan kaldırarak bakım ve desteği basitleştirir. Bulut uygulamaları, değişen iş talebini karşılamak için çalışma zamanında görevleri birden çok sanal makineye klonlayarak elde edilebilen ölçeklenebilirliklerinde diğer uygulamalardan farklıdır. Yük dengeleyiciler, işi sanal makineler kümesine dağıtır. Bu süreç, yalnızca tek bir erişim noktası gören bulut kullanıcısı için şeffaftır. Çok sayıda bulut kullanıcısını barındırmak için bulut uygulamaları çok kiracılı olabilir, yani herhangi bir makine birden fazla bulut kullanıcılı kuruluşa hizmet edebilir.

**Sunucusuz Bilgi İşlem Veya Fonksiyon Olarak Servis**

Sunucusuz bilgi işlem, bulut sağlayıcısının, istekleri yerine getirmek için gerektiği şekilde sanal makineleri başlatmayı ve durdurmayı tamamen yönettiği ve isteklerin, sanal makine başına değil, isteği karşılamak için gereken kaynakların soyut bir ölçüsüyle faturalandırıldığı bir bulut bilişim kod yürütme modelidir. saatte. [7][Adına rağmen, aslında sunucular olmadan kod çalıştırmayı içermez. [7] Sunucusuz bilgi işlem, sistemin sahibi olan işletme veya kişinin arka uç kodunun çalışması için sunucular veya sanal makineler satın alması, kiralaması veya sağlaması gerekmediği için bu şekilde adlandırılmıştır.

Servis olarak işlev (FaaS), olaylara yanıt olarak çalışan bulutta bireysel işlevlerin devreye alınmasını sağlamak için sunucusuz bilgi işlemden yararlanan, servis tarafından barındırılan bir uzaktan prosedür çağrısıdır. [8] FaaS, bazıları tarafından sunucusuz bilgi işlem çatısı altına girerken, bazıları da terimleri birbirinin yerine kullanır. [9]

Sunucusuz bilgi işlem, bulut sağlayıcısının, istekleri yerine getirmek için gerektiği şekilde sanal makineleri başlatmayı ve durdurmayı tamamen yönettiği ve isteklerin, sanal makine başına değil, isteği karşılamak için gereken kaynakların soyut bir ölçüsüyle

faturalandırıldığı bir bulut bilişim kod yürütme modelidir. saatte. [7] Adına rağmen, aslında sunucular olmadan kod çalıştırmayı içermez. [7] Sunucusuz bilgi işlem, sistemin sahibi olan işletme veya kişinin arka uç kodunun çalışması için sunucular veya sanal makineler satın alması, kiralaması veya sağlaması gerekmediği için bu şekilde adlandırılmıştır.

Servis olarak işlev (FaaS), olaylara yanıt olarak çalışan bulutta bireysel işlevlerin devreye alınmasını sağlamak için sunucusuz bilgi işlemden yararlanan, servis tarafından barındırılan bir uzaktan prosedür çağrısıdır. [8] FaaS, bazıları tarafından sunucusuz bilgi işlem çatısı altına girerken, bazıları da terimleri birbirinin yerine kullanır. [9]

### 1.3 Bulut Bilişim Kullanım Alanları

Bulut bilişim şüphesiz büyüyen bir pazardır ve bulut bilişim servislerinin birçok avantajlı kullanımı vardır. Yeni işletmeler hızla buluta geçiyor ve yazılımların test edilmesi ve geliştirilmesi, iletişimi, depolanması ve dağıtımı için ideal ortam haline geldi.

Bulut bilişimin tüm bu kullanımları ve yukarıda özetlenen faydaları göz önüne alındığında, bulut bilişimi bugün işinize dahil etmemeniz ve rekabet avantajı elde etmemeniz için hiçbir neden yoktur.

**Dosya Depolama**

Verilerinizi nasıl saklayacağınız ve bunlara nasıl erişeceğiniz konusunda birçok seçenek vardır. Dizüstü bilgisayarınızda sabit sürücü, verileri yedeklemek ve aktarmak için kullandığınız harici sabit sürücü, ağ dosya paylaşımları, USB sürücüler ve daha fazlası var.

Bu kadar çok depolama seçeneği mevcutken, bulut depolamayı benzersiz kılan nedir?

Bulut depolamanın çekici olmasının ana nedeni, dosyalara kolaylıkla erişilip düzenlenebilir olmasıdır. Tek ihtiyacınız olan bir internet bağlantısı ve dosyalarınıza herhangi bir cihazdan, her yerden erişebilirsiniz.

Dosya ve nesne depolama dahil olmak üzere çeşitli bulut depolama türleri mevcuttur. Bunların her biri, paylaşılan dosya sistemlerinden blok tabanlı birimlere ve yedekleme ve arşivleme sistemlerine kadar farklı kullanım durumlarına uyar.

Amazon S3, DropBox veya OneDrive gibi bulut bilgi işlem depolama servisleriyle, ihtiyaçlarınıza ve bütçenize göre depolamayı artırmak veya azaltmak için güvenli erişim ve ölçeklenebilirlik sağlanır. Bu nedenle, bu tür depolama yalnızca güvenli değil, aynı zamanda son derece uygun maliyetlidir.

**Büyük Veri Analizi**

Bugün, büyük veri toplayarak müşterilerinizin pazar eğilimlerini, işletmenizin satış performansı ve daha fazlası hakkında analiz yapabilirsiniz.

Her büyüklükteki şirket, sayısız nedenden dolayı büyük veriye ihtiyaç duyar. Bazıları bunu ticari büyüme için yeni fırsatlar keşfetmek için toplarken, diğerleri bunu karmaşık sorunlara çözüm bulmak için yapıyor. Ancak büyük verileri toplamak ve analiz etmek kolay olmuyor. Yüksek bir fiyat etiketi ile gelen geniş bilgi işlem kaynaklarının kullanılmasını gerektirir.

Bulut bilişim için gereken kaynakları satın alacak olsaydınız, pazarlama gibi diğer temel hizmetler için ayrılan bütçeleri kısmanız gerekebilir.

Bulut bilişimin birincil yararı, kullandığın kadar öde fiyatlandırma stratejisiyle gelmesidir. Bu, kullanılmayan zaman için ödeme yapmanız gerekmeyeceği anlamına gelir, işinize önemli miktarda para tasarrufu sağlar. Kaynaklara yalnızca ihtiyacınız olduğunda erişir ve onlar için ödeme yaparsınız. Şüphesiz bulut bilişim, büyük veri analitiğini basit, kullanışlı ve ucuz hale getiriyor

**Veri Yedekleme Ve Arşivleme**

Bugün, siber suçların çok olduğu bir dünyada yaşıyoruz. Hiçbir gün, büyük veri ihlali vakaları olmadan geçmeyecek, bu da zaman zaman çok sayıda işletme için ölümcül hale geliyor.

Geleneksel veri yedekleme yöntemleri, verilerin uzun süre yedeklenmesinde etkili olduğu kanıtlanmıştır. Fakat virüs tehdidi ve taşınabilir olmaları nedeniyle kaybolabilir ve modern işletmeler için bir tehdit oluşturabilirler.

Bulut tabanlı yedekleme ve arşivleme bu zorluklara bir çözümdür. Uygulanması kolaydır ve maksimum veri güvenliği sağlar. Bu yaklaşımla, hassas dosyalarınızı bulut tabanlı depolama sistemlerine yedekleyebilir veya arşivleyebilirsiniz. Bu, verileriniz bir şekilde tehlikeye girse bile verilerinizin hala sağlam olduğuna dair güvence sağlar.

Bazı bulut bilişim servisleri, ihtiyaçlarınızı karşılamak için yedeklemeleri planlamanıza olanak tanır. Ayrıca, bulut yedeklemelerinizi şifreleyebilir ve bilgisayar korsanlarının erişmesini imkansız hale getirebilirsiniz.

Bulut depolama ile, istediğiniz kadar yer bulabilir ve ihtiyacınız kadar veri depolayabilir ve yalnızca kullandığınız şey için ödeme yapabilirsiniz.

### 1.4 Neden Bulut Tabanlı Platformu Tercih Ettik?

**Ölçeklenebilirlik**

Bulut bilişimin birincil faydalarından biri, ölçeklenebilirliğidir. Örneğin, bir BT çözümünü kolayca (ve hızlı bir şekilde) ölçeklendirebilmek, iş üzerinde anında ve geniş kapsamlı bir etkiye sahip olabilecek bir şeydir. Bir ortamı isteğe bağlı olarak ölçeklendirmek geçmişte mümkün değildi. Kuruluşlar donanım kurulumlarının boyutu ve işlem gücü ile sınırlıydı. Bulutla birlikte bu sınırlama ortadan kalktı. Bulut, işletmelerin teknik kaynakları yönetme biçiminde gerçekten devrim yarattı. Artan ihtiyaçlar doğrultusunda kaynak artırımı kolaylıkla sağlanır.

**Düşük Maliyet**

Bulut tabanlı platformu tercih etmemizin nedenlerinden düşük maaliyetli olmasıdır. Tüm veriler servis sağlayacının veri merkezinde saklandığından donanım, yazılım satın almanıza gerek kalmaz. Sadece kullandığımız servisin ücreti kadar ödeme yaparsınız. Ayrıca büyüyen bir şirketin ek bir donanım edinmesi ya da genişleyen ağ kurmasını gerektiren masraflar söz konusu değildir.

**Suncu Bakımı Muafiyeti**

Servis sağlayıcı verilerin depolandığı sunucunun bakımını sağlamakla yükümlüdür. Kullanıcı ya da müşterilerin böyle bir sorumlulukları yoktur.

**Zaman Tasarrufu**

Servis sağlayıcısı sunucunun tedarik süreçleri, işletim sisteminin kurulması, yapılandırması gibi işlemlerden sorumlu olduğundan kullanıcıların bu tür konulara endişe duymasına gerek yoktur.

**Güvenlik**

Genelde işletmelerin veri merkezleri tek bir coğrafi bölgede konumlandığından kötü hava koşulları, elektrik kesintileri gibi sorunlara maruz kalabilirler. Ancak Microsoft, Google ve Amazon gibi genel bulut sağlayıcıları, birçok farklı bölgede geniş bir sunucu ağına sahiptir. Yani veri merkezlerinden biri çökse bile diğerleri sayesinde herhangi bir aksama yaşanmaması garantilenmeye çalışılır. Bu da çoğu işletmenin kendi kendine sağlayabileceğinden daha yüksek bir güvenilirlik seviyesi anlamına gelir.

Veriler fiziksel olarak zarar görmez ve verilerin fiziksek olarak ele geçirilme durumu yaşanmaz. Şifreleme algoritmaları ile korunduğundan dışarıdan olabilecek müdahaleleri engeller. Güvenlik güncellemesi gerektiği durumlarda gerekli güncellemeler servis sağlayacıları tarafından sisteme uygulanır.

# 2.KULLANILAN TEKNOLOJİLER

## 2.1 Buradaa Web Uygulaması

Web Uygulamasında React Js kullanılmıştır. Açık kaynak olmasının yanı sıra her gün büyüyen bir topluluğa sahiptir. Bu sayede uygulama geliştirme aşamasında aldığımız beklenmedik bir hata karşısında oldukça hızlı bir şekilde çözüme ulaşmamıza olanak sağlamıştır. Mobil uygulama tarafında React-Native kullandığımızdan ötürü ortak komponent kullanımına olanak sağlaması, bu sayede web'te veya mobil tarafta geliştirilen bir komponent yapısını sorunsuz bir şekilde hem web tarafında hem de mobil uygulama tarafında kullanabilmektedir. React'ın sağladığı "state" yöntemi sayesinde sunduğu rahatlık ve iş yükünü azaltmasından dolayı Web uygulamasını gelişirirken React Js teknolojisi kullanılmıştır.

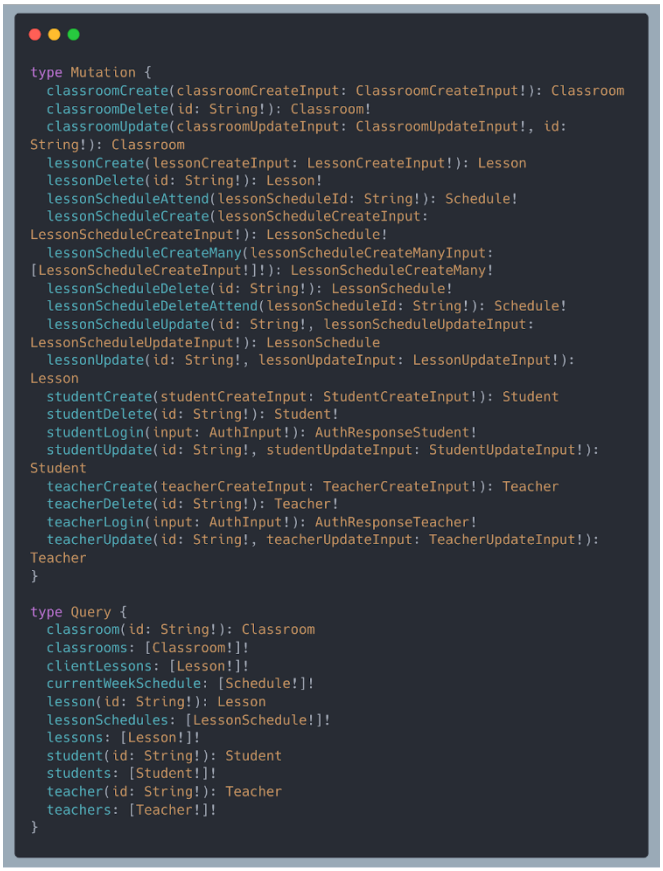
## 2.2 Buradaa Mobil Uygulaması

Akıllı telefonlarda bu sistemi kullanabilmek, öğrenci ve öğretmenlerin kullanımına sunabilmek için bir mobil uygulama geliştirilmesi gerekmektedir. Mobil platformlarda ios ve android tabanlı iki farklı işletim sistemi bulunduğu için iki farklı işletim sisteminde de uygulamayı geliştirmek gerekmektedir. Bu doğrultuda React-Native programlama dili sayesinde bu iki platforma da aynı anda uygulama çıkarma imkanını mümkün kılmaktadır. React, yalnızca görünüm katmanıyla ilgilenen çok basit ve hafif bir kütüphanedir. Angular veya Ember gibi diğer MV kütüphaneleri gibi büyük değildir. Herhangi bir Javascript geliştiricisi, temel bilgileri anlayabilir ve yalnızca birkaç gün eğitim okuduktan sonra harika bir web uygulaması geliştirmeye başlayabilir.

React'ın güçlü yönlerinden biri, iyi bir soyutlama sağlamasıdır, bu da kullanıcıya herhangi bir karmaşık iç öğeyi göstermediği anlamına gelir. React öğrenmek bir bonus ile birlikte gelir. React-Native dokümantasyonunda yazdığı gibi "Bir kez yaz, her yerde çalıştır." kütüphanesi değil, "Bir kez öğren, her yerde yaz" kütüphanesidir. Web için yazdığınız kodun birebir aynısını kullanamayacak olsanız da aynı metodolojiyi ve aynı mimariyi kullanabilirsiniz.

React-Native platformlar arası bir köprü niteliği görerek yazılan kodları her platforma özel bir şekilde native kodlarına çevirmemize olanak sağlamaktadır. Bu köprü görevini sağlmasına rağmen performans olarak da oldukça iyi durumdadır.

### 2.3 Buradaa Veri Tabanı Ve Cloud İşlemleri



```
type Mutation {
  classroomCreate(classroomCreateInput: ClassroomCreateInput!): Classroom
  classroomDelete(id: String!): Classroom!
  classroomUpdate(classroomUpdateInput: ClassroomUpdateInput!, id:
String!): Classroom
  lessonCreate(lessonCreateInput: LessonCreateInput!): Lesson
  lessonDelete(id: String!): Lesson!
  lessonScheduleAttend(lessonScheduleId: String!): Schedule!
  lessonScheduleCreate(lessonScheduleCreateInput:
LessonScheduleCreateInput!): LessonSchedule!
  lessonScheduleCreateMany(lessonScheduleCreateManyInput:
[LessonScheduleCreateInput!]!): LessonScheduleCreateMany!
  lessonScheduleDelete(id: String!): LessonSchedule!
  lessonScheduleDeleteAttend(lessonScheduleId: String!): Schedule!
  lessonScheduleUpdate(id: String!, lessonScheduleUpdateInput:
LessonScheduleUpdateInput!): LessonSchedule
  lessonUpdate(id: String!, lessonUpdateInput: LessonUpdateInput!):
Lesson
  studentCreate(studentCreateInput: StudentCreateInput!): Student
  studentDelete(id: String!): Student!
  studentLogin(input: AuthInput!): AuthResponseStudent!
  studentUpdate(id: String!, studentUpdateInput: StudentUpdateInput!):
Student
  teacherCreate(teacherCreateInput: TeacherCreateInput!): Teacher
  teacherDelete(id: String!): Teacher!
  teacherLogin(input: AuthInput!): AuthResponseTeacher!
  teacherUpdate(id: String!, teacherUpdateInput: TeacherUpdateInput!):
Teacher
}

type Query {
  classroom(id: String!): Classroom
  classrooms: [Classroom!]!
  clientLessons: [Lesson!]!
  currentWeekSchedule: [Schedule!]!
  lesson(id: String!): Lesson
  lessonSchedules: [LessonSchedule!]!
  lessons: [Lesson!]!
  student(id: String!): Student
  students: [Student!]!
  teacher(id: String!): Teacher
  teachers: [Teacher!]!
}
```

**Şekil 1** GraphQL Uç Noktaları

Buradaa (mobil/web) uygulaması için geliştirilen geri-uç (back-end) yazılımlarda, NestJs, TypeScript, GraphQL, PostgreSQL gibi teknolojilerden yararlanıldı. Cloud sunucu için ise AWS'nin veri tabanı için kullanılan RDS(Geleneksel Veri Tabanı Sunucusu) aracı, GraphQL sorguları için Apollo aracı kullanıldı.

Buradaa uygulamasının geri-uçta güç aldığı bu gereçlerden kısaca bahsetmek gerekirse;

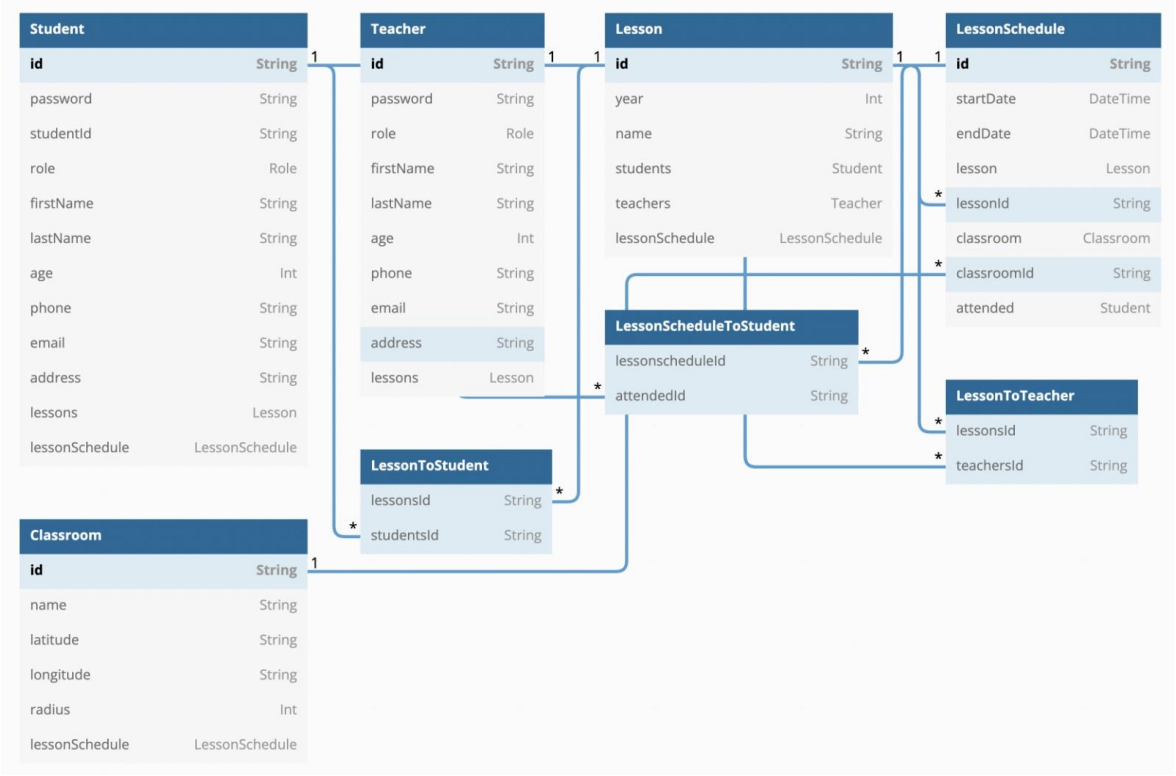
GraphQL, API'lar için açık kaynak kodlu, sorgulama (query) ve manipülasyon (manipulation) dilidir. FaceBook tarafından 2015 senesinde şirketin ihtiyaçları sebebiyle doğmuş ve zamanla geleneksel API'ların sorunlarına çözümler getirmiştir. Geleneksel API

lar istekler sonucunda oluşan tüm veriyi gönderir ve alır. GraphQL bu req-res işlemleri sırasında veri tabanından gelen ve veri tabanından giden verileri ihtiyaç doğrultusunda filtreleyerek veri akışı sırasında yaşanabilecek karmaşıklıkların ve performans düşmelerinin önüne geçmeyi amaçlar.

NestJs, Kamil Myśliwiec tarafından sunucu-tarafı (server-side) uygulamaların geliştirilmesini kolaylaştırmak amacıyla geliştirilmiş bir Node.js Framework'üdür. Express.js ve Node.js'i içinde barındıran progressive JavaScript kullanan framework, TypeScript'i desteklemektedir ve uygulama içerisinde kullanılmasının tercih sebeplerinden birisi budur. Birkaç başka tercih sebebi ise ölçeklenebilir olması, test edilebilirlik açısından kolaylık sunması, mimari bakımından büyük sorunlara etkili çözümler getiriyor olması, sürdürülebilir ve gevşek-bağımlılık (loosly-coupled) sağlamasıdır.

Typescript, nesne yönelimli programlamanın (object orianted programming) bütün gereçlerini sağlayan, tip güvenli (type safety), derlenen (compilable) olan JavaScript tabanlı bir programlama dilidir. Nesne yönelimli çalışmalarda ve GraphQL gibi kompleks obje yapılarıyla çalışan API'ları kullanırken first-code desteği ve ile tip güvenli çalışması nedeniyle yapılan uygulama esnasında tip karmaşasıyla karşılaşma riskini ortadan kaldırır ve otomatik API şeması (schema) çıkarmasıyla yazılımcının işini epey hafifleten, hızlı ve güvenli programlar yazmaya olanak tanıyan yapısı nedeniyle tercih ettiğimiz bir dil oldu.

SQL/PostgreSQL, Ekibin aşina olduğu bir yöntem. Veri tabanı oluşturmak için yeni bir yönteme gerek duyulmadı. İlişkisel bir veri tabanı oluşturarak, birbiriyle ilişkili verilerin (ders-öğretici-öğrenci, öğrenci-ders, öğretici-ders, ders-sınıf-öğrenci vs.) veri tabanında tablolar halinde bulunması ve bu sayede üzerinde yapılmak istenen işlemlerin (silme, kaydetme, değiştirme) daha optimize yapılabilmesi için. Bir veriyi birçok kere kullanmadan tablolar halinde düzenlenmesi daha erişilebilir bir veri-tabanı oluşturuyor. Aynı zamanda GraphQL'inde benzer structured bir yapısı olması nedeni tercih sebeplerimiz arasında yer alıyor. GraphQL ile yapılacak sorguların SQL ile benzerlik taşıması yazılım geliştirme sürecini hızlandırıyor ve aynı zamanda daha az hata ile yapılabilmesini sağlıyordu.




**Şekil 2** İlişkisel Veri Tabanı Şeması

AWS(Amazon Web Services); bir web servisi olarak 2002 yılında kurulan AWS, 2006 yılından bu yana bir bulut-hesaplama(cloud-computing) sağlayıcısı olarak hayatına devam etmektedir.

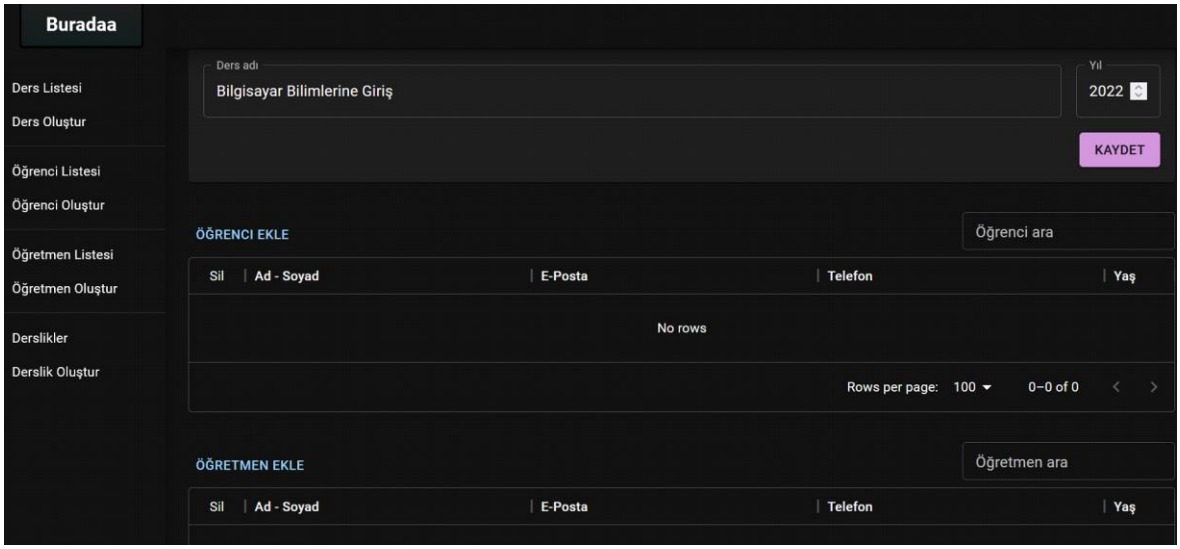
Peki neden bulut(cloud) tabanlı bir uygulama geliştirmek istedik ve neden AWS'nin ürünlerini bu yönde kullanmayı tercih ettik? Bu sorulara açık bir cevap vermek gerekirse, tamamen fizibilite ve hız konuları ön plana çıkmaktadır. Örneğin, fizibilite bir uygulama için veya bir işlem türü için fiziksel bir sunucu kurmak meşakkatli ve aynı zamanda maliyetli (zaman ve para bakımından) bir iştir. Bu sebeple bulut-hesaplama çözümleri hemen her şekilde çalışabilecek portatif bir uygulama/hesaplama geliştirmeye gerçekçi ve yenilikçi bir çözüm getirmektedir. İkinci olarak, AWS ürünlerini kullanmayı tercih etme sebebimiz öğrenci ve geliştiriciler için 1 yıllık kiralama bedeli almamalarından ve çözümlerinin basit ve anlaşılır olmasından dolayıdır.

Sonuç olarak, hızlı, fizibilitesi yüksek, güvenli ve hem kullanıcı hem geliştirici dostu bir geri-uç (back-end) çözümlerini uygulamamızda kullanmaya çalıştık. Uygulamamızın gerçek hayat problemine sağladığı çözümü en iyi ve en güvenilir şekilde yapması gerektiği bilinciyle seçimlerimizi yaptık.

# 3. UYGULAMALAR

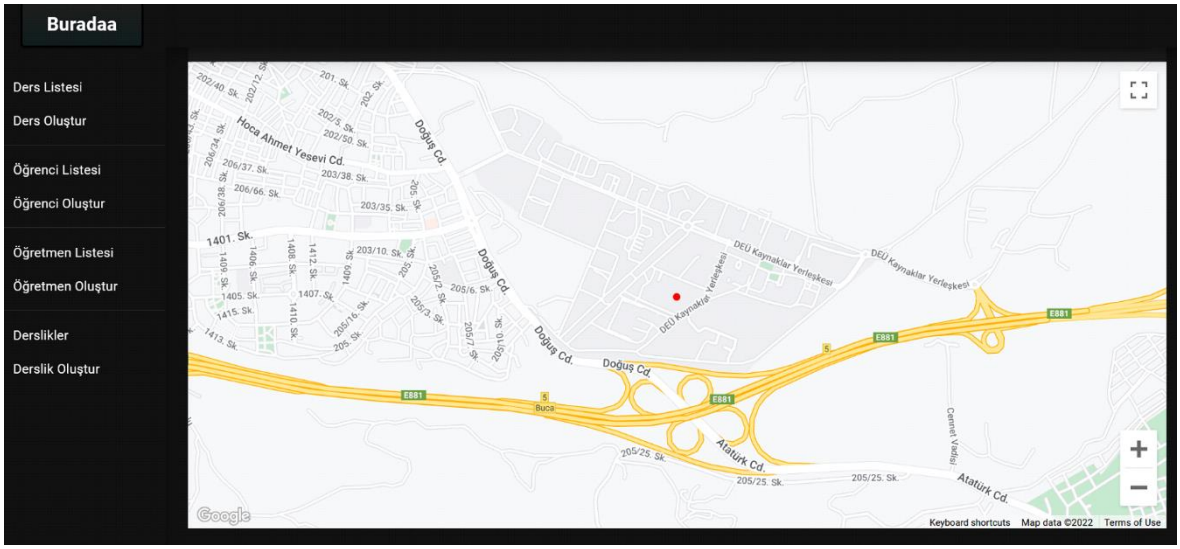
## 3.1 Buradaa Web Uygulaması

Buradaa Web uygulamasına giriş yapıldıktan sonra ders oluşturma işleminde öncelikle dersin adı, öğrencileri ve öğretmeninin eklenmesi gerekir. Dersin ilgili verileri girildikten sonra dersin işleneceği derslikler hakkında bilgilerin (derslik adı, dersliğin kodu, dersliğin konum bilgisi ve dersliğin genişliği) girilmesi gerekir.



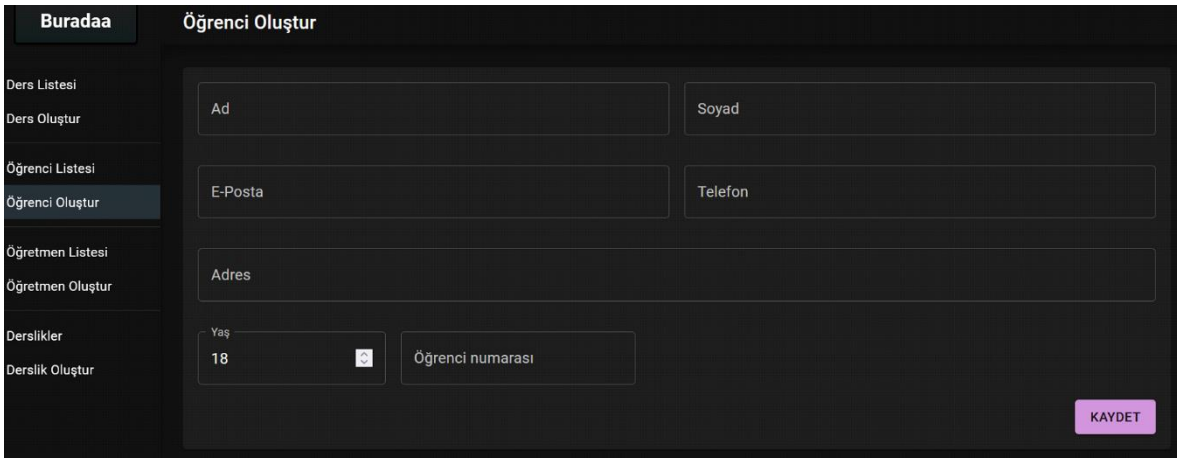

**Şekil 3** Ders Oluşturma İşleminde Öğrenci ve Öğretmen Ekleme Ekranı

Şekil 3'de bulunan ekrandan dersliğin konumu seçilir ve bilgileri girilir. Bu bilgiler uygulamanın ana amaçlarından olan yoklama alınması esnasında öğrencilerin ilgili ders saatinde dersliğin konumu merkez alınarak hesaplanan dersliğin alanında olup olmadıklarının tespitinde kullanılmaktadır.

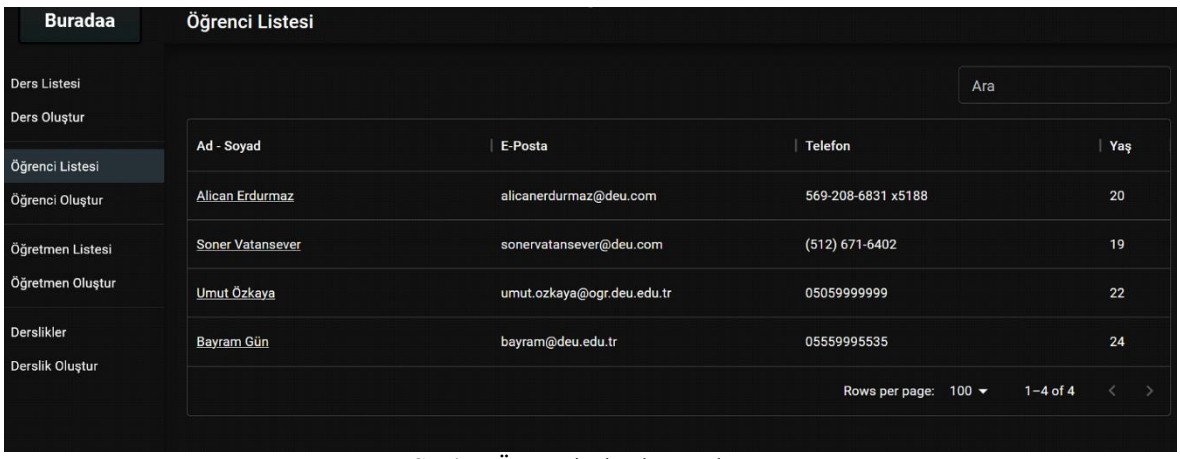



**Şekil 4** Dersliğin Konum Bilgilerin Giriş Ekranı

Şekil 4'de görülen ekrandan öğrenciler oluşturulur. Bilgileri girilen öğrenciler veritabanında tutulur ve gerek duyulan yerlerde bu veriler kullanılır.

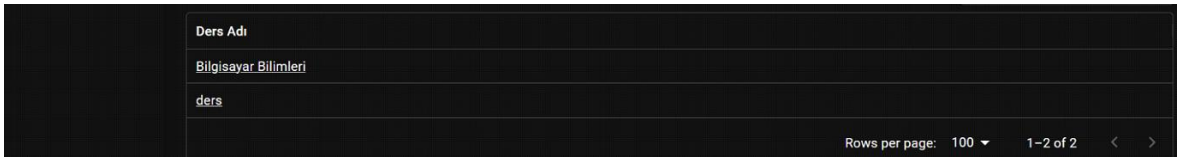



**Şekil 5** Öğrenci Oluşturma Ekranı



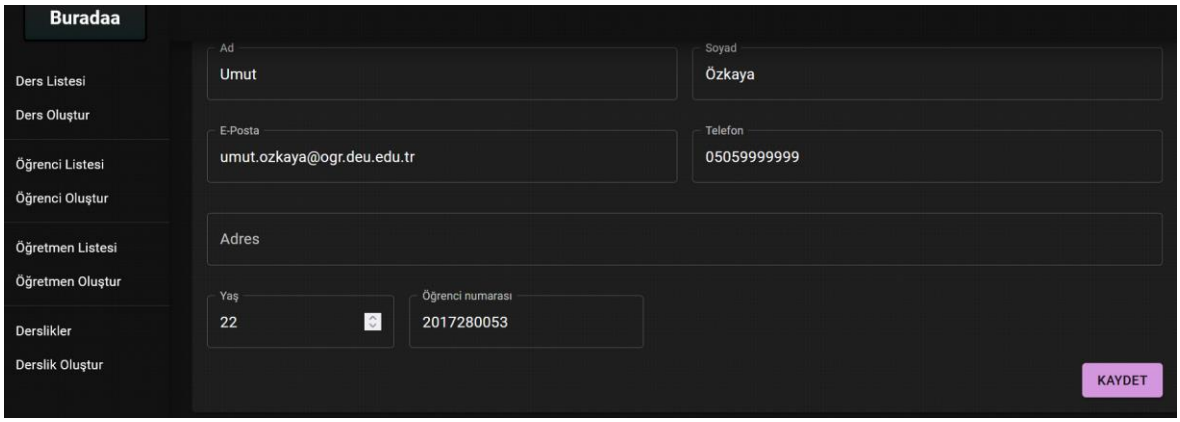

**Şekil 6** Öğrenci Listeleme Ekranı

Öğrenci Listesi sekmesinde okulda bulunan bütün öğrenciler görülebilmektedir. Öğrenci isimlerine tıklayıp onlar hakkında detaylı bilgi alınabilir veya öğrenci güncelle butonuna basarak öğrenci bilgileri güncellenebilir. Öğrenci isminin sağında bulunan simgeye tıklandığı zaman öğrenci silinebilir.




**Şekil 7** Ders Listeleme Ekranı

Ders listeleme ekranında veri tabanında kayıtlı bulunan tüm derslerin listesi görüntülenebilir. Herhangi bir dersin üzerine tıklanarak ders ile ilgili daha detaylı bilgilere erişileibilir.

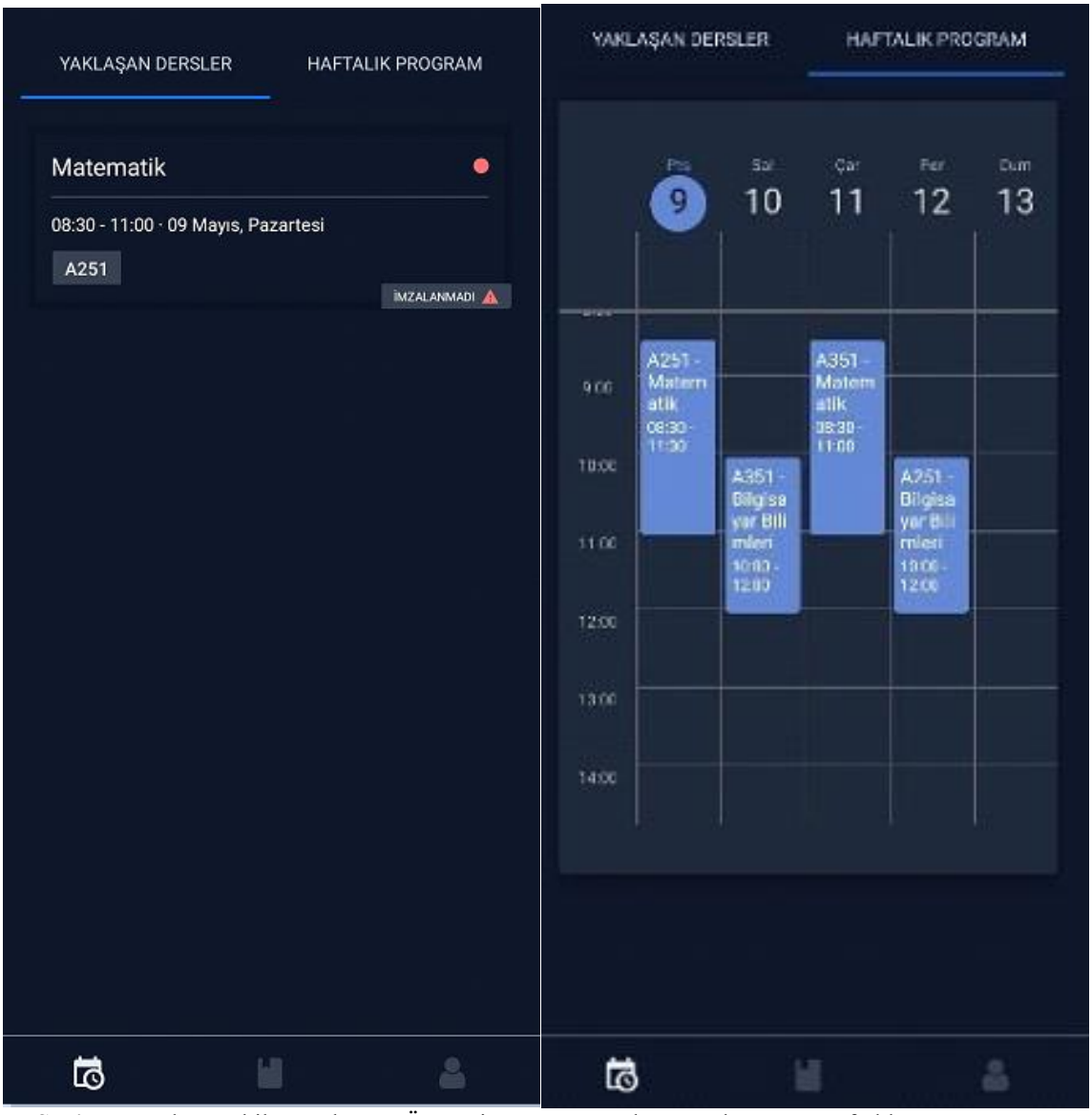



**Şekil 8** Buradaa Web Uygulaması Öğrenci Bilgileri Görüntülenme Ekran Görüntüsü

Güncellemek istenen öğrencinin bilgileri dolu bir şekilde gelmektedir. Kullanıcı gerekli yerleri düzelterek öğrenci bilgilerini güncelleyebilir.

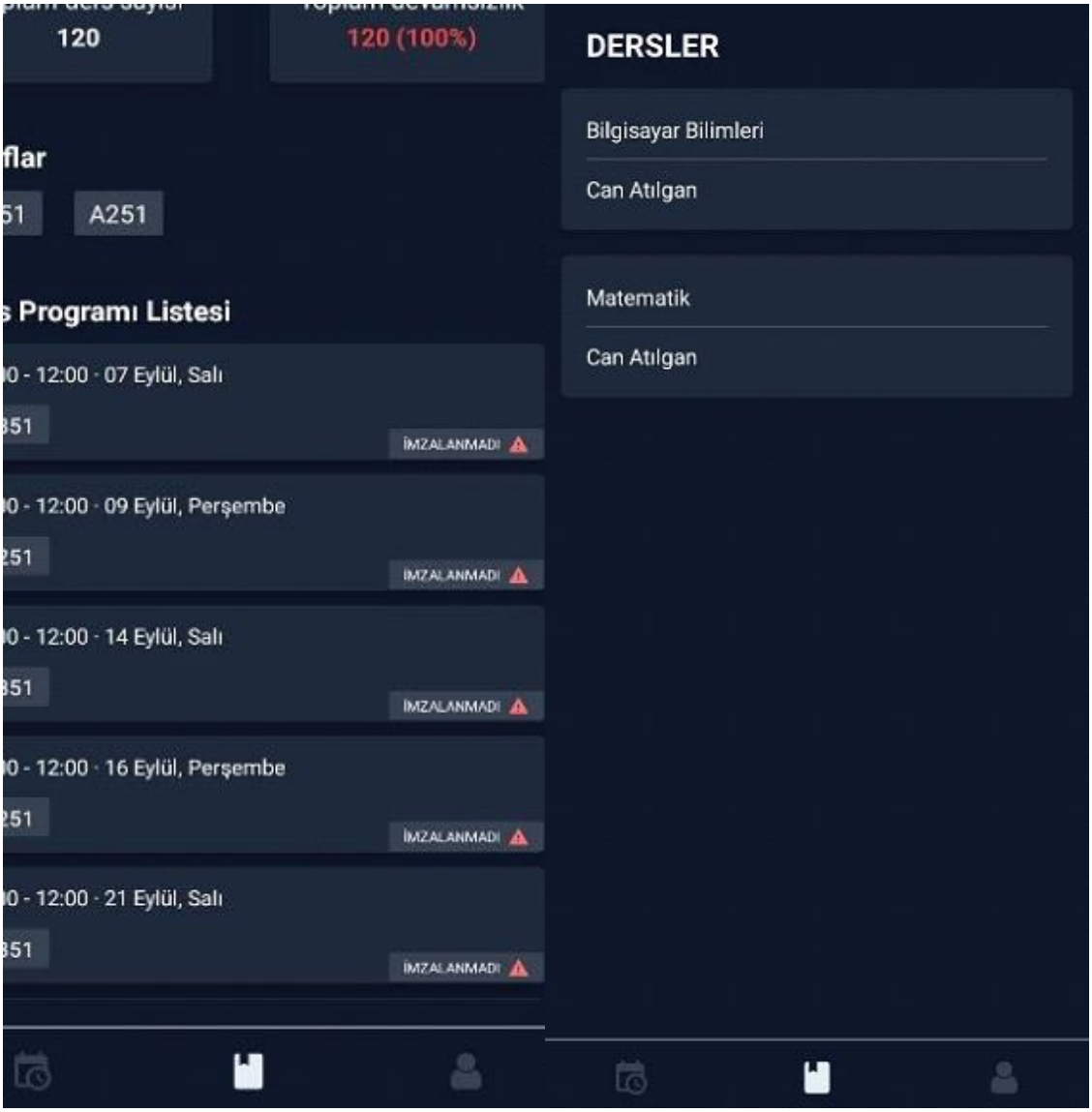
### 3.2 Buradaa Mobil Uygulaması

Buradaa Öğrenci Mobil uygulamasına giriş yapıldıktan sonra karşılaşılan ilk ekranda alınan dersler ve yaklaşan dersler gösterilir ve haftalık olarak ders programını görüntüleyebildiği ekran görüntüsü.




**Şekil 9** Buradaa Mobil Uygulaması Öğrenci Ders Görüntülenme Ekranı Ve Haftalık Ders Programı Çizelgesi

Öğrenciler devamsızlık bilgilerini öğrenmek istediği dersi seçtikten sonra ders çizelgesinde devamsızlık hakkını ve yüzdesini görüntüleyebilir.




**Şekil 10** Buradaa Mobil Uygulaması Devamsızlık Takip Ekranı

# 4. KULLANIM SENARYOSU

## 4.1 Admin Kullanım Senaryosu

Öğrenci oluşturma:

1. https://buradaa-admin.netlify.app/ adresine girilir.
2. Soldan **Öğrenci Oluştur** seçilir.
3. Gerekli bilgiler doldurulur.
4. Kaydet'e tıklanır.

Öğretmen oluşturma

1. https://buradaa-admin.netlify.app/ adresine girilir.
2. Soldan **Öğretmen Oluştur** seçilir.
3. Gerekli bilgiler doldurulur.
4. Kaydet'e tıklanır.

Ders oluşturma

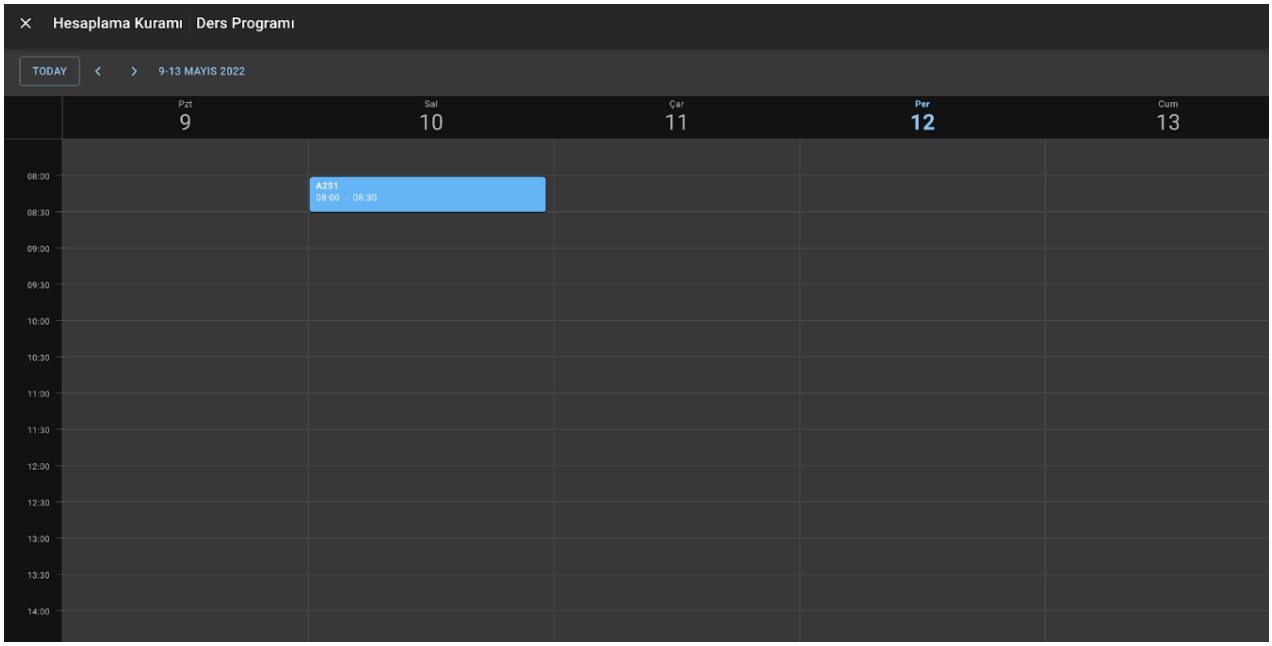
1. https://buradaa-admin.netlify.app/ adresine girilir.
2. Soldan **Ders Oluştur** seçilir.
3. Gerekli bilgiler doldurulur.
4. Kaydet'e tıklanır.

Derslik oluşturma

1. https://buradaa-admin.netlify.app/ adresine girilir.
2. Soldan **Derslik Oluştur** seçilir.
3. Gerekli bilgiler doldurulur.
4. Seçilen koordinatlar daha sonrasında öğrencinin sınıfta olup olmadığının tespitinde kullanılacaktır.
5. Kaydet'e tıklanır.

Ders programı ekleme

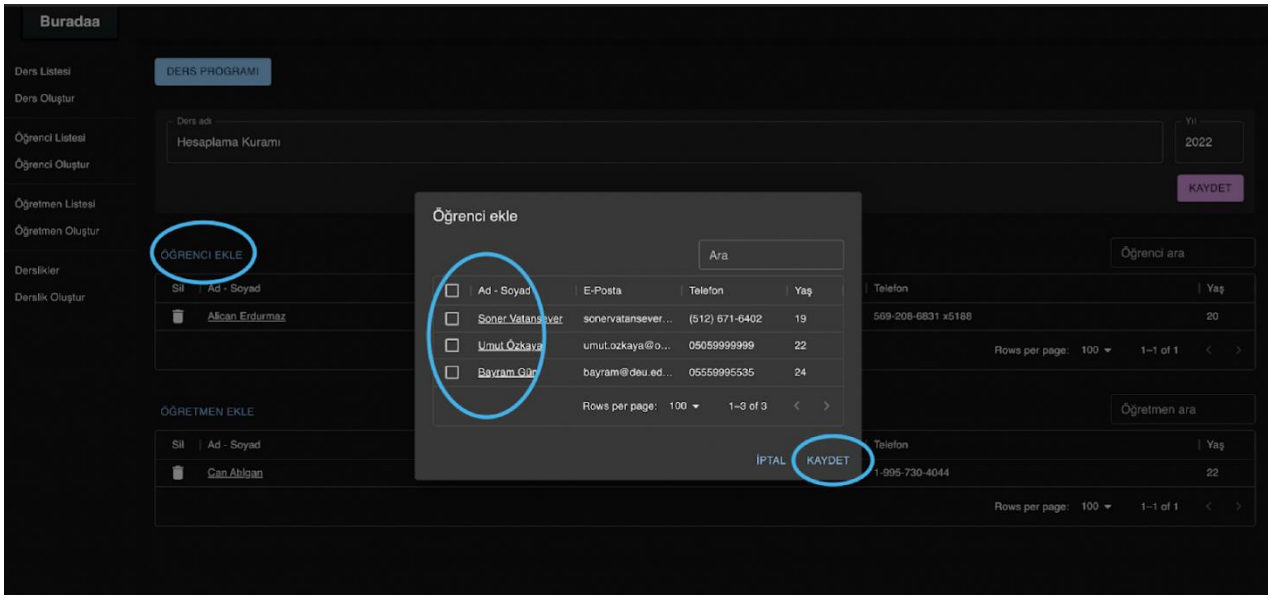
1. https://buradaa-admin.netlify.app/ adresine girilir.
2. Soldan **Ders Oluştur** seçilir.
3. Yukarıdan Ders programı seçilir.
4. Açılan takvimden bir tarih seçilir.
5. Gerekli bilgiler doldurulur.
6. Kaydet'e tıklanır.




**Şekil 11** Ders Programı Ekran Görüntüsü

Derse öğrenci ekleme

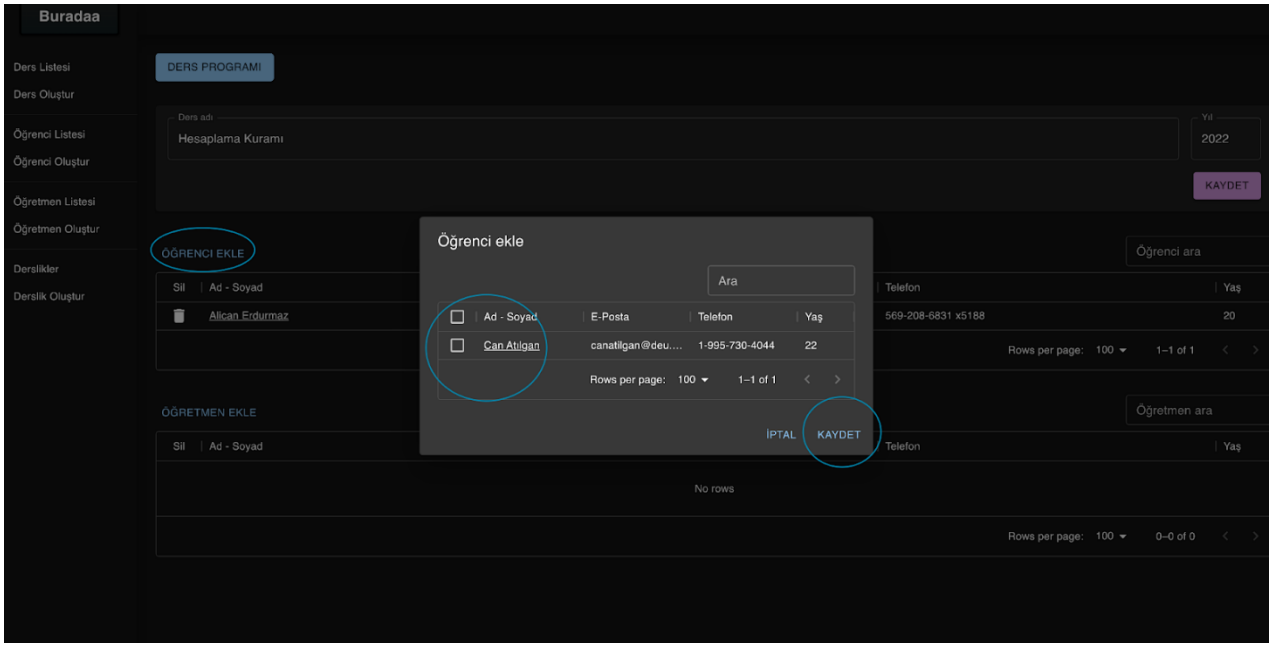
1. https://buradaa-admin.netlify.app/ adresine girilir.
2. Soldan **Ders Listesi** seçilir.
3. İstenilen ders seçilir.
4. Öğrenci ekle'ye tıklanılır.
5. Öğrenci seçilir.
6. Kaydet'e tıklanır.




**Şekil 12** Öğrenci Ekleme Ekran Görüntüsü

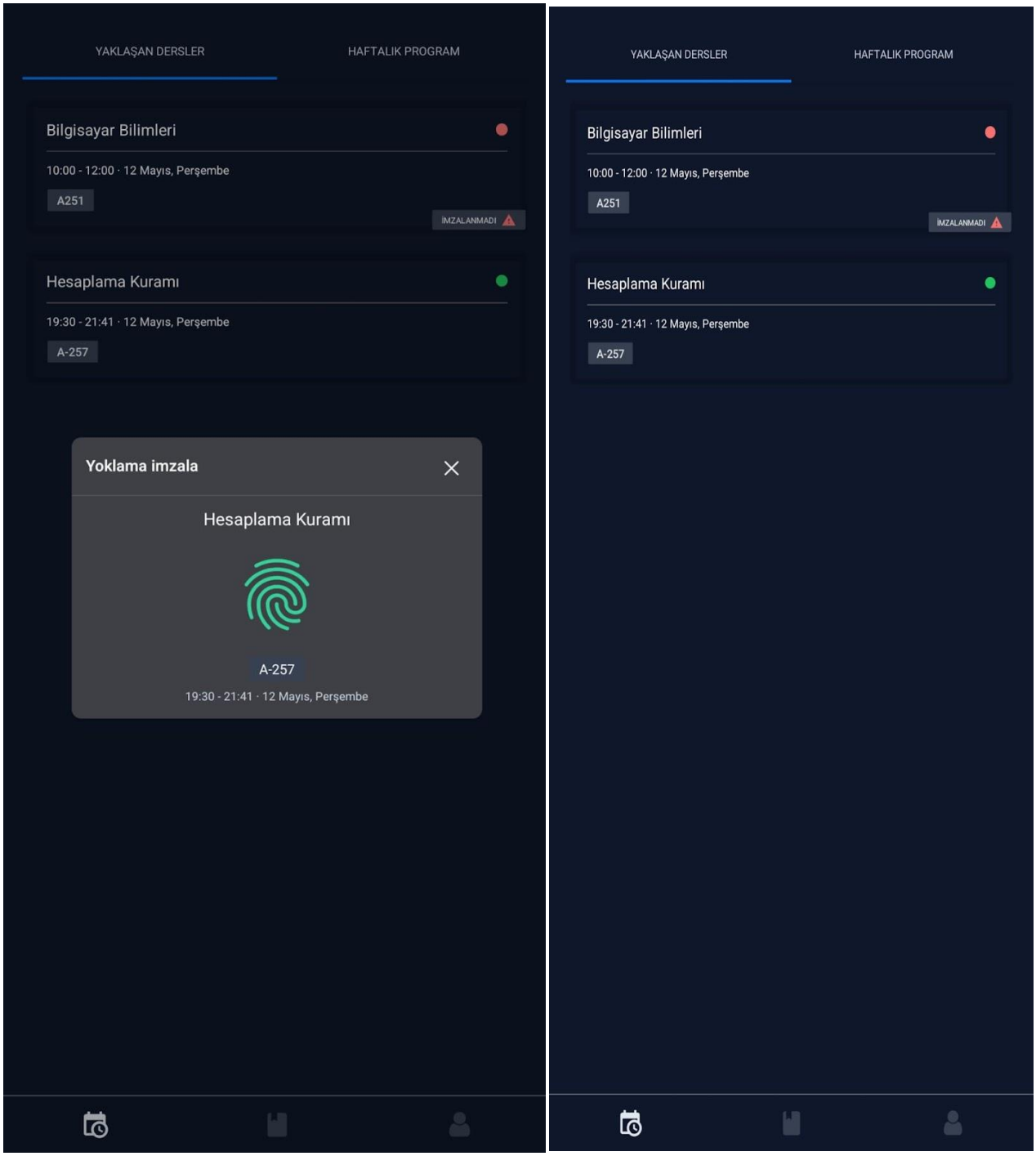
Derse öğretmen ekleme

1. https://buradaa-admin.netlify.app/ adresine girilir.
2. Soldan **Ders Listesi** seçilir.
3. İstenilen ders seçilir.
4. Öğretmen ekle'ye tıklanılır.
5. Öğretmen seçilir.
6. Kaydet'e tıklanır.



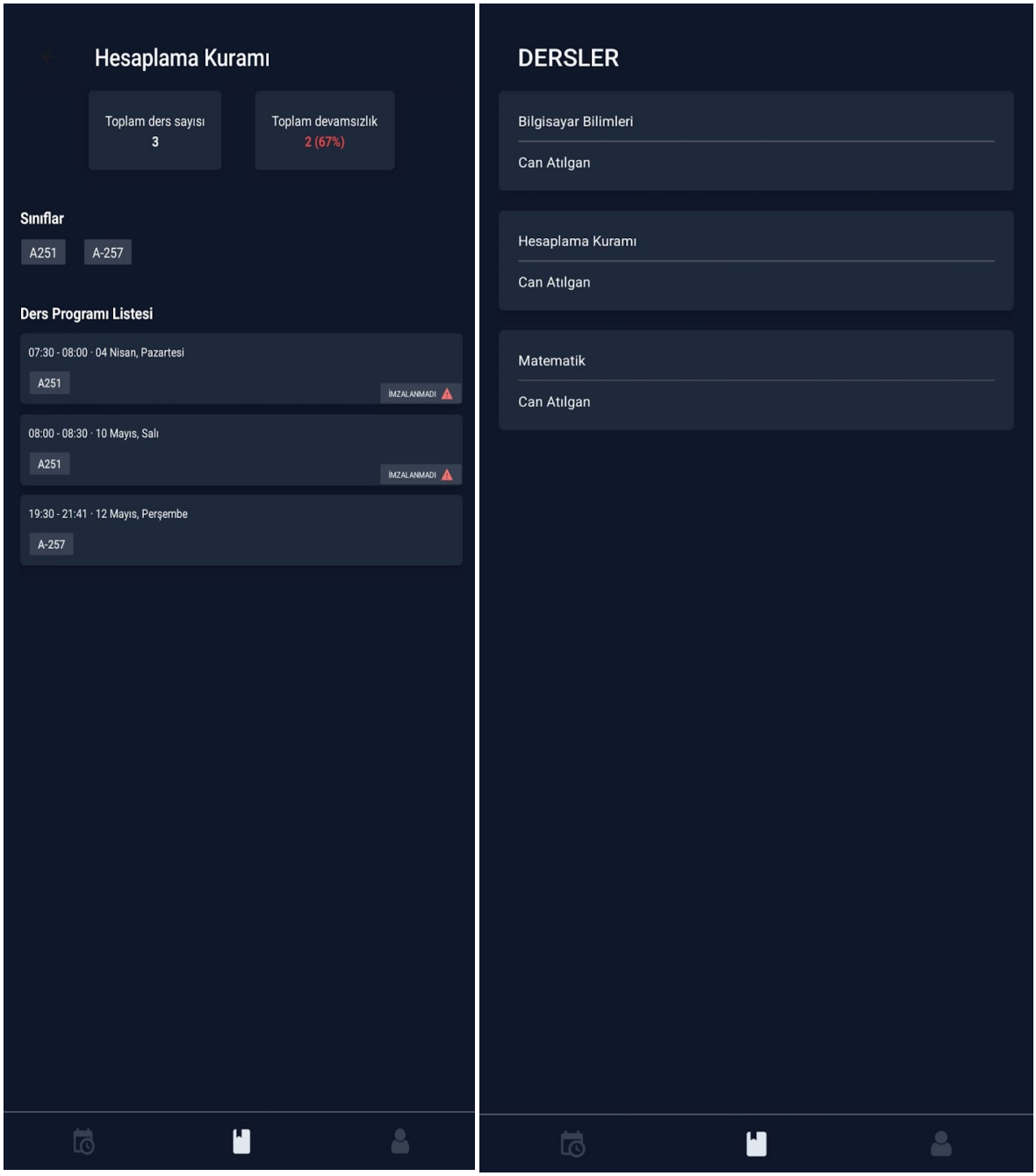

**Şekil 13** Derse Öğretmen Ekleme Ekran Görüntüsü

### 4.2 Mobil Kullanım Senaryosu



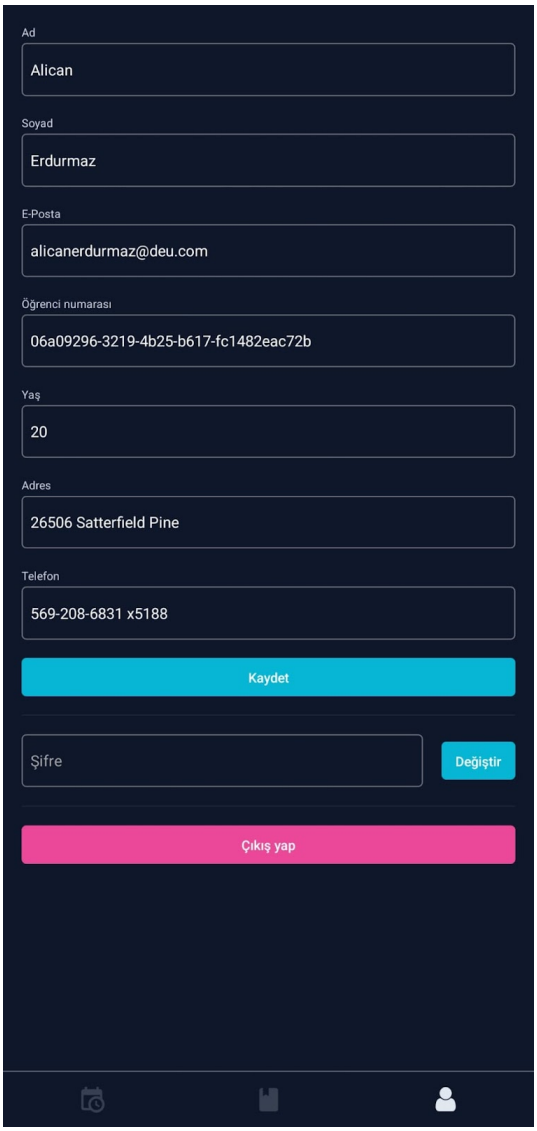

**Şekil 14** Uygulama Ana Giriş Ekran Görüntüsü

Uygulama ana ekranında yaklaşan dersler görülür. Eğer ders saati başladıysa ve ders bitmemişse öğrenci derse tıklayıp imza atabilir.




**Şekil 15** Uygulama İmza Ekranı ve Derslerin Ekran Görüntüsü

Uygulamanın 2. Sekmesinde öğrencinin kayıtlı olduğu tüm dersler görülür. Ders detayına tıklandığında, o derse ait tüm ders programı listelenir. Toplam devamsızlık sayısı ve o derse ait sınıfların bilgisi mevcuttur. Bir Sınıfa tıklandığında Google Maps'den sınıf konumu görülebilir.




**Şekil 16** Öğrenci Bilgilerinin Ekran Görüntüsü

Uygulamanın 3. Sekmesinde öğrenci bilgileri mevcuttur. Öğrenci istediği takdirde bilgilerini güncelleyebilir

## KAYNAKÇA

React JS Güncel Dökümantasyon


[1] «Cloud Learn: IBM,» 18 08 2020. [Çevrimiçi]. Available: https://www.ibm.com/cloud/learn/cloud-computing.

[2] Duan, Yucong; Fu, Guohua; Zhou, Nianjun; Sun, Xiaobing; Narendra, Nanjangud; Hu, Bo, «Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends,» *IEEE 8th International Conference on Cloud Computing*, pp. 621-628, 2015.

[3] P. Mell ve T. Grance, «The NIST Definition of Cloud Computing,» National Institute of Standards and Technology, 2011.

[4] A. Amies, H. Sluiman, Q. G. Tong ve G. N. Liu, Infrastructure as a Service Cloud Concepts, IBM Press, 2012.

[5] M. Boniface ve e. al, «Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds,» *International Conference on Internet and Web Applications and Services (ICIW)*, no. 5, pp. 155-160, 2010.

[6] «https://reactjs.org/docs/getting-started.html,» 2022. [Çevrimiçi].

[7] «https://www.apollographql.com/docs/,» 2022. [Çevrimiçi].

[8] «https://www.prisma.io/docs/,» 2022. [Çevrimiçi].

[9] «https://reactnative.dev/docs/getting-started,» 2022. [Çevrimiçi].

[10] «https://www.postgresql.org/docs/,» 2022. [Çevrimiçi].

[11] «https://reactnative.dev,» [Çevrimiçi].

## EKLER

### Buradaa Admin Paneli Kaynak Kodları

```
import type { AppProps } from 'next/app'
import { ApolloProvider } from '@apollo/client'
import { ThemeProvider } from '@emotion/react'
import { CssBaseline } from '@mui/material'
import timezone from 'dayjs/plugin/timezone'
import utc from 'dayjs/plugin/utc'
import dayjs from 'dayjs'
import Layout from '../src/components/Layout'
import { client } from '../src/lib/apollo'
import { theme } from '../src/lib/theme'

dayjs.extend(utc)
dayjs.extend(timezone)
dayjs.tz.setDefault('Europe/Istanbul')

import './globalStyles.css'

function MyApp({ Component, pageProps }: AppProps) {
 return (
   <ApolloProvider client={client}>
     <ThemeProvider theme={theme}>
       <CssBaseline />
       <Layout>
         <Component {...pageProps} />
       </Layout>
     </ThemeProvider>
   </ApolloProvider>
 )
}

export default MyApp

import { LoadingButton } from '@mui/lab'
import { Paper, Box } from '@mui/material'
import React, { ReactNode } from 'react'
import { FormProvider, UseFormReturn } from 'react-hook-form'
import Form from '../Form/Form'
import Input from '../Form/Input'
import MapInput from '../Form/MapInput'

interface ClassroomFormProps<T> {
```

```
 onSubmit: any
 methods: UseFormReturn<T, any>
 loading: boolean
 children?: ReactNode
}

const ClassroomForm = <T extends {}>({ onSubmit, methods, loading, children }:
ClassroomFormProps<T>) => {
 return (
   <FormProvider {...methods}>
     <Form onSubmit={onSubmit}>
       <Paper sx={{ padding: '16px' }}>
         <Box display='flex' alignItems='center'>
           <Input name='name' label='Ad' fullWidth sx={{ mr: 2 }} />
           <Input type='number' name='radius' label='Alan' fullWidth />
         </Box>
         <Box display='flex' alignItems='center' mt={4}>
           <Input name='latitude' label='Enlem' sx={{ mr: 2 }} fullWidth />
           <Input name='longitude' label='Boylam' fullWidth />
         </Box>
         <LoadingButton
           disabled={loading}
           loading={loading}
           variant='contained'
           color='secondary'
           type='submit'
           sx={{ marginTop: '16px', marginLeft: 'auto', display: 'block' }}>
           Kaydet
         </LoadingButton>
       </Paper>

       <MapInput />

       {children}
     </Form>
   </FormProvider>
 )
}

export default ClassroomForm

import { LoadingButton } from '@mui/lab'
import { Paper, Box } from '@mui/material'
import { ReactNode } from 'react'
import { FormProvider, UseFormReturn } from 'react-hook-form'
import Form from '../Form/Form'
import Input from '../Form/Input'

interface LessonFormProps<T> {
 onSubmit: any
 methods: UseFormReturn<T, any>
```

```
 loading: boolean
 children?: ReactNode
}

const LessonForm = <T extends {}>({ onSubmit, methods, loading, children }:
LessonFormProps<T>) => {
 return (
   <FormProvider {...methods}>
     <Form onSubmit={onSubmit}>
       <Paper sx={{ padding: '16px' }}>
         <Box display='flex' alignItems='center' justifyContent='space-between'>
           <Input name='name' label='Ders adı' sx={{ mr: 2 }} fullWidth />
           <Input name='year' label='Yıl' type='number' sx={{ width: '100px' }} />
         </Box>
         <LoadingButton
           disabled={loading}
           loading={loading}
           variant='contained'
           color='secondary'
           type='submit'
           sx={{ marginTop: '16px', marginLeft: 'auto', display: 'block' }}>
           Kaydet
         </LoadingButton>
       </Paper>
       {children}
     </Form>
   </FormProvider>
 )
}

export default LessonForm

import { LoadingButton } from '@mui/lab'
import { Paper, Box } from '@mui/material'
import React, { ReactNode } from 'react'
import { FormProvider, UseFormReturn } from 'react-hook-form'
import Form from '../Form/Form'
import Input from '../Form/Input'

interface StudentFormProps<T> {
 onSubmit: any
 methods: UseFormReturn<T, any>
 loading: boolean
 children?: ReactNode
}

const StudentForm = <T extends {}>({ onSubmit, methods, loading, children }:
StudentFormProps<T>) => {
 return (
   <FormProvider {...methods}>
     <Form onSubmit={onSubmit}>
```

```
        <Paper sx={{ padding: '16px' }}>
          <Box display='flex' alignItems='center'>
            <Input name='firstName' label='Ad' sx={{ mr: 2 }} fullWidth />
            <Input name='lastName' label='Soyad' fullWidth />
          </Box>
          <Box display='flex' alignItems='center' mt={4}>
            <Input name='email' label='E-Posta' type='email' sx={{ mr: 2 }} fullWidth
/>
            <Input name='phone' label='Telefon' type='tel' fullWidth />
          </Box>
          <Input name='address' label='Adres' sx={{ mt: 4 }} fullWidth />
          <Input name='age' label='Yaş' type='number' sx={{ width: '200px', mt: 4, mr:
2 }} />
          <Input name='studentId' label='Öğrenci numarası' sx={{ width: '250px', mt: 4
}} />
          <LoadingButton
            disabled={loading}
            loading={loading}
            variant='contained'
            color='secondary'
            type='submit'
            sx={{ marginTop: '16px', marginLeft: 'auto', display: 'block' }}>
            Kaydet
          </LoadingButton>
        </Paper>
        {children}
      </Form>
    </FormProvider>
  )
}

export default StudentForm

import { LoadingButton } from '@mui/lab'
import { Paper, Box } from '@mui/material'
import React, { ReactNode } from 'react'
import { FormProvider, UseFormReturn } from 'react-hook-form'
import Form from '../Form/Form'
import Input from '../Form/Input'

interface TeacherFormProps<T> {
  onSubmit: any
  methods: UseFormReturn<T, any>
  loading: boolean
  children?: ReactNode
}

const TeacherForm = <T extends {}>({ onSubmit, methods, loading, children }:
TeacherFormProps<T>) => {
  return (
    <FormProvider {...methods}>
```

```
      <Form onSubmit={onSubmit}>
        <Paper sx={{ padding: '16px' }}>
          <Box display='flex' alignItems='center'>
            <Input name='firstName' label='Ad' sx={{ mr: 2 }} fullWidth />
            <Input name='lastName' label='Soyad' fullWidth />
          </Box>
          <Box display='flex' alignItems='center' mt={4}>
            <Input name='email' label='E-Posta' type='email' sx={{ mr: 2 }} fullWidth
/>
            <Input name='phone' label='Telefon' type='tel' fullWidth />
          </Box>
          <Input name='address' label='Adres' sx={{ mt: 4 }} fullWidth />
          <Input name='age' label='Yaş' type='number' sx={{ width: '200px', mt: 4 }} />
          <LoadingButton
            disabled={loading}
            loading={loading}
            variant='contained'
            color='secondary'
            type='submit'
            sx={{ marginTop: '16px', marginLeft: 'auto', display: 'block' }}>
            Kaydet
          </LoadingButton>
        </Paper>
        {children}
      </Form>
    </FormProvider>
  )
}

export default TeacherForm

import { gql, useMutation } from '@apollo/client'
import { useRouter } from 'next/router'
import { useForm } from 'react-hook-form'
import ClassroomForm from '../../components/ClassroomForm'
import { CLASSROOMS_QUERY } from '../Classrooms/Classrooms'
import { ClassroomCreateVariables, ClassroomCreate } from
'./__generated__/ClassroomCreate'

type ClassroomCreateInput = ClassroomCreateVariables['classroomCreateInput']

const CLASSROOM_CREATE = gql`
  mutation ClassroomCreate($classroomCreateInput: ClassroomCreateInput!) {
    classroomCreate(classroomCreateInput: $classroomCreateInput) {
      id
    }
  }
`

const ClassroomCreate = () => {
  const router = useRouter()
```

```
 const [classroomCreate, { loading }] = useMutation<ClassroomCreate,
ClassroomCreateVariables>(CLASSROOM_CREATE, {
   refetchQueries: [{ query: CLASSROOMS_QUERY }],
   awaitRefetchQueries: true,
 })

 const formMethods = useForm<ClassroomCreateInput>({
   defaultValues: {
     latitude: '38.423733',
     longitude: '27.142826',
     name: '',
     radius: 100,
   },
 })

 const onSubmitHandler = async (data: ClassroomCreateInput) => {
   const result = await classroomCreate({
     variables: {
       classroomCreateInput: {
         ...data,
         latitude: data.latitude.toString(),
         longitude: data.longitude.toString(),
       },
     },
   })
   if (!result.data?.classroomCreate) return
   router.push({ pathname: '/classroom/[id]', query: { id:
result.data.classroomCreate.id } })
 }

 return <ClassroomForm methods={formMethods} loading={loading}
onSubmit={formMethods.handleSubmit(onSubmitHandler)} />
}

export default ClassroomCreate

import { useMutation } from '@apollo/client'
import { FC } from 'react'
import { useForm } from 'react-hook-form'
import ClassroomForm from '../../components/ClassroomForm'
import { ClassroomUpdateInput } from '../../globalTypes'
import { ClassroomUpdateVariables } from '../Classrooms/__generated__/ClassroomUpdate'
import { CLASSROOM_DETAIL_QUERY, CLASSROOM_UPDATE } from './ClassroomQuery'
import { ClassroomFragment } from './__generated__/ClassroomFragment'
import { ClassroomUpdate } from './__generated__/ClassroomUpdate'

interface StudentDetailProps {
 classroom: ClassroomFragment
}

const StudentDetail: FC<StudentDetailProps> = ({ classroom }) => {
```

```
 const { id, name, latitude, longitude, radius } = classroom

 const [classroomUpdate, { loading }] = useMutation<ClassroomUpdate,
ClassroomUpdateVariables>(CLASSROOM_UPDATE, {
   refetchQueries: [{ query: CLASSROOM_DETAIL_QUERY, variables: { id } }],
 })

 const formMethods = useForm<ClassroomUpdateInput>({
   defaultValues: {
     name,
     latitude,
     longitude,
     radius,
   },
 })

 const onSubmitHandler = async (data: ClassroomUpdateInput) => {
   console.log(data)
   await classroomUpdate({
     variables: {
       classroomUpdateId: id,
       classroomUpdateInput: data,
     },
   })
 }

 return <ClassroomForm methods={formMethods} loading={loading}
onSubmit={formMethods.handleSubmit(onSubmitHandler)} />
}

export default StudentDetail

import { gql, useQuery } from '@apollo/client'
import { GridColDef } from '@mui/x-data-grid'
import { useMemo } from 'react'

import DataTable, { primaryColumn, TableLink, textColumn } from '../../DataTable'
import { routes } from '../../util/routes'
import { ClassroomsQuery } from './__generated__/ClassroomsQuery'

export const CLASSROOMS_QUERY = gql`
 query ClassroomsQuery {
   classrooms {
     id
     name
     latitude
     longitude
   }
 }
`
```

```
const Classrooms = () => {
 const { data, loading } = useQuery<ClassroomsQuery>(CLASSROOMS_QUERY)

 const columns: GridColDef[] = useMemo(
   () => [
     primaryColumn({
       field: 'name',
       headerName: 'Derslik Adı',
       renderCell: ({ row, id }) => <TableLink href={{ pathname: routes.classroom,
query: { id } }} text={row.name} />,
     }),
     textColumn({
       field: 'latitude',
       headerName: 'Enlem',
     }),
     textColumn({
       field: 'longitude',
       headerName: 'Boylam',
     }),
   ],
   []
 )

 return (
   <DataTable loading={!data || loading} rows={data?.classrooms || []}
columns={columns} searchableFields={['name']} />
 )
}

export default Classrooms

import { gql, useMutation } from '@apollo/client'
import { useRouter } from 'next/router'
import React from 'react'
import { useForm } from 'react-hook-form'
import LessonForm from '../../components/LessonForm'
import { LESSON_QUERY } from '../Lessons/Lessons'
import { LessonCreate, LessonCreateVariables } from './__generated__/LessonCreate'

type LessonCreateInput = LessonCreateVariables['lessonCreateInput']

const LESSON_CREATE = gql`
 mutation LessonCreate($lessonCreateInput: LessonCreateInput!) {
   lessonCreate(lessonCreateInput: $lessonCreateInput) {
     id
   }
 }
`

const LessonCreate = () => {
 const router = useRouter()
```

```
 const [lessonCreate, { loading }] = useMutation<LessonCreate,
LessonCreateVariables>(LESSON_CREATE, {
   refetchQueries: [{ query: LESSON_QUERY }],
   awaitRefetchQueries: true,
 })

 const formMethods = useForm<LessonCreateInput>({
   defaultValues: {
     name: '',
     year: new Date().getFullYear(),
     studentIds: [],
   },
 })

 const onSubmitHandler = async (data: LessonCreateInput) => {
   const result = await lessonCreate({
     variables: {
       lessonCreateInput: data,
     },
   })
   if (!result.data?.lessonCreate) return
   router.push({ pathname: '/lesson/[id]', query: { id: result.data.lessonCreate.id }
})
 }

 return <LessonForm methods={formMethods} loading={loading}
onSubmit={formMethods.handleSubmit(onSubmitHandler)} />
}

export default LessonCreate

import { useMutation } from '@apollo/client'
import { Box, Button } from '@mui/material'
import { GridColDef } from '@mui/x-data-grid'
import { FC, useCallback, useMemo, useState } from 'react'
import { useForm } from 'react-hook-form'
import LessonForm from '../../components/LessonForm'
import DataTable, { deleteAction, isDeleteRowClicked, primaryColumn, TableLink,
textColumn } from '../../DataTable'
import { LessonUpdateInput } from '../../globalTypes'
import { routes } from '../../util/routes'
import LessonDetailAddStudents from
'./LessonDetailAddStudents/LessonDetailAddStudents'
import LessonDetailAddTeachers from './LessonDetailAddTeachers'
import LessonDetailSchedule from './LessonDetailSchedule'
import { LESSON_DETAIL_QUERY, LESSON_UPDATE } from './LessonQuery'
import { LessonQuery_lesson } from './__generated__/LessonQuery'
import { LessonUpdate, LessonUpdateVariables } from './__generated__/LessonUpdate'

enum ModalState {
 Closed,
```

```
 Students,
 Teachers,
 Schedule,
}

interface LessonDetailProps {
 lesson: LessonQuery_lesson
}

const LessonDetail: FC<LessonDetailProps> = ({ lesson }) => {
 const [modal, setModal] = useState<ModalState>(ModalState.Closed)
 const [lessonUpdate, { loading }] = useMutation<LessonUpdate,
LessonUpdateVariables>(LESSON_UPDATE, {
   refetchQueries: [{ query: LESSON_DETAIL_QUERY, variables: { id: lesson.id } }],
 })

 const formMethods = useForm<LessonUpdateInput>({
   defaultValues: {
     name: lesson.name,
     year: lesson.year,
     studentIds: lesson.students.map((s) => s.id),
     teacherIds: lesson.teachers.map((t) => t.id),
   },
 })

 const studentColumns: GridColDef[] = useMemo(
   () => [
     deleteAction(),
     primaryColumn({
       field: 'firstName',
       headerName: 'Ad - Soyad',
       renderCell: ({ row, id }) => (
         <TableLink href={{ pathname: routes.student, query: { id } }}
text={`${row.firstName} ${row.lastName}`} />
       ),
     }),
     textColumn({
       field: 'email',
       headerName: 'E-Posta',
     }),
     textColumn({
       field: 'phone',
       headerName: 'Telefon',
     }),
     textColumn({
       width: 75,
       field: 'age',
       headerName: 'Yaş',
     }),
   ],
   []
```

```
  )

  const teacherColumns: GridColDef[] = useMemo(
    () => [
      deleteAction(),
      primaryColumn({
        field: 'firstName',
        headerName: 'Ad - Soyad',
        renderCell: ({ row, id }) => (
          <TableLink href={{ pathname: routes.teacher, query: { id } }}
text={`${row.firstName} ${row.lastName}`} />
        ),
      }),
      textColumn({
        field: 'email',
        headerName: 'E-Posta',
      }),
      textColumn({
        field: 'phone',
        headerName: 'Telefon',
      }),
      textColumn({
        width: 75,
        field: 'age',
        headerName: 'Yaş',
      }),
    ],
    []
  )

  const onSubmitHandler = useCallback(
    async (data: LessonUpdateInput) => {
      const year = data?.year ?? new Date().getFullYear()
      await lessonUpdate({
        variables: {
          lessonUpdateId: lesson.id,
          lessonUpdateInput: { ...data, year: Number(year) },
        },
      })
    },
    [lesson.id, lessonUpdate]
  )

  const handleAddStudent = useCallback(
    async (studentIds: string[]) => {
      const list = formMethods.getValues()?.studentIds ?? []
      list.push(...studentIds)
      formMethods.setValue('studentIds', list)
      formMethods.handleSubmit(onSubmitHandler)()
      setModal(ModalState.Closed)
    },
```

```
    [formMethods, onSubmitHandler]
  )

  const handleAddTeacher = useCallback(
    async (teacherIds: string[]) => {
      const list = formMethods.getValues()?.teacherIds ?? []
      list.push(...teacherIds)
      formMethods.setValue('teacherIds', list)
      formMethods.handleSubmit(onSubmitHandler)()
      setModal(ModalState.Closed)
    },
    [formMethods, onSubmitHandler]
  )

  const handleDeleteStudent = useCallback(
    (id: string) => {
      const list = formMethods.getValues().studentIds?.filter((studentId: string) =>
studentId !== id) ?? []
      formMethods.setValue('studentIds', list)
      formMethods.handleSubmit(onSubmitHandler)()
    },
    [formMethods, onSubmitHandler]
  )

  const handleDeleteTeacher = useCallback(
    (id: string) => {
      const list = formMethods.getValues().teacherIds?.filter((teacherId: string) =>
teacherId !== id) ?? []
      formMethods.setValue('teacherIds', list)
      formMethods.handleSubmit(onSubmitHandler)()
    },
    [formMethods, onSubmitHandler]
  )

  return (
    <Box>
      <Box mb={4}>
        <Button variant='contained' onClick={() => setModal(ModalState.Schedule)}>
          Ders Programı
        </Button>
      </Box>

      <LessonForm methods={formMethods} loading={loading}
onSubmit={formMethods.handleSubmit(onSubmitHandler)}>
        <Box marginTop={4}>
          <DataTable
            loading={loading}
            headerSlot={<Button onClick={() => setModal(ModalState.Students)}>Öğrenci
ekle</Button>}
            rows={lesson.students}
            columns={studentColumns}
```

```
            searchableFields={['firstName', 'lastName', 'email', 'phone']}
            density='compact'
            searchPlaceholder='Öğrenci ara'
            onCellClick={(row) => {
              if (isDeleteRowClicked(row)) handleDeleteStudent(row.id as string)
            }}
          />
        </Box>

        <Box marginTop={4}>
          <DataTable
            loading={loading}
            headerSlot={<Button onClick={() => setModal(ModalState.Teachers)}>Öğretmen
ekle</Button>}
            rows={lesson.teachers}
            columns={teacherColumns}
            searchableFields={['firstName', 'lastName', 'email', 'phone']}
            density='compact'
            searchPlaceholder='Öğretmen ara'
            onCellClick={(row) => {
              if (isDeleteRowClicked(row)) handleDeleteTeacher(row.id as string)
            }}
          />
        </Box>

        {modal === ModalState.Schedule && (
          <LessonDetailSchedule
            lessonId={lesson.id}
            lesson={lesson}
            schedules={lesson.schedules}
            onClose={() => setModal(ModalState.Closed)}
          />
        )}

        {modal === ModalState.Students && (
          <LessonDetailAddStudents
            studentsIdsAlreadyInLesson={formMethods.getValues('studentIds') || []}
            onClose={() => setModal(ModalState.Closed)}
            onSubmit={handleAddStudent}
          />
        )}

        {modal === ModalState.Teachers && (
          <LessonDetailAddTeachers
            teachersIdsAlreadyInLesson={formMethods.getValues('teacherIds') || []}
            onClose={() => setModal(ModalState.Closed)}
            onSubmit={handleAddTeacher}
          />
        )}
      </LessonForm>
    </Box>
```

```
 )
}

export default LessonDetail

import { gql, useQuery } from '@apollo/client'
import { GridColDef } from '@mui/x-data-grid'
import { useMemo } from 'react'
import DataTable, { primaryColumn, TableLink, textColumn } from '../../DataTable'
import { routes } from '../../util/routes'

export const LESSON_QUERY = gql`
 query LessonsQuery {
   lessons {
     id
     name
     year
     studentCount
     teachers {
       firstName
       lastName
     }
   }
 }
`

const Lessons = () => {
 const { data, loading } = useQuery(LESSON_QUERY)

 const columns: GridColDef[] = useMemo(
   () => [
     primaryColumn({
       field: 'name',
       headerName: 'Ders adı',
       flex: 1,
       renderCell: ({ value, id }) => <TableLink href={{ pathname: routes.lesson,
query: { id } }} text={value} />,
     }),
     textColumn({
       field: 'teachers',
       headerName: 'Öğretmenler',
       valueGetter: ({ row }) => {
         const teachers = row.teachers.map(({ firstName, lastName }: any) =>
`${firstName} ${lastName}`).join(', ')
         return teachers
       },
     }),
     textColumn({
       field: 'year',
       headerName: 'Yıl',
       width: 75,
```

```
      }),
      textColumn({
        width: 125,
        field: 'studentCount',
        headerName: 'Öğrenci Sayısı',
        type: 'number',
      }),
    ],
    []
  )

  return (
    <DataTable loading={!data || loading} rows={data?.lessons || []} columns={columns}
searchableFields={['name']} />
  )
}

export default Lessons

import { gql, useMutation } from '@apollo/client'
import { useRouter } from 'next/router'
import { useForm } from 'react-hook-form'
import StudentForm from '../../components/StudentForm'
import { STUDENTS_QUERY } from '../Students/Students'
import { StudentCreateVariables, StudentCreate } from './__generated__/StudentCreate'

type StudentCreateInput = StudentCreateVariables['studentCreateInput']

const STUDENT_CREATE = gql`
  mutation StudentCreate($studentCreateInput: StudentCreateInput!) {
    studentCreate(studentCreateInput: $studentCreateInput) {
      id
    }
  }
`

const StudentCreate = () => {
  const router = useRouter()
  const [studentCreate, { loading }] = useMutation<StudentCreate,
StudentCreateVariables>(STUDENT_CREATE, {
    refetchQueries: [{ query: STUDENTS_QUERY }],
    awaitRefetchQueries: true,
  })

  const formMethods = useForm<StudentCreateInput>({
    defaultValues: {
      firstName: '',
      lastName: '',
      email: '',
      phone: '',
      age: 18,
```

```
      address: '',
      studentId: '',
    },
  })

  const onSubmitHandler = async (data: StudentCreateInput) => {
    const result = await studentCreate({
      variables: {
        studentCreateInput: { ...data, age: Number(data.age), password: data.studentId
},
      },
    })
    if (!result.data?.studentCreate) return
    router.push({ pathname: '/student/[id]', query: { id: result.data.studentCreate.id
} })
  }

  return <StudentForm methods={formMethods} loading={loading}
onSubmit={formMethods.handleSubmit(onSubmitHandler)} />
}

export default StudentCreate

import { useMutation } from '@apollo/client'
import { Box, Button } from '@mui/material'
import { GridColDef } from '@mui/x-data-grid'
import { FC, useMemo } from 'react'
import { useForm } from 'react-hook-form'
import StudentForm from '../../components/StudentForm'
import DataTable, { primaryColumn, TableLink } from '../../DataTable'
import { StudentUpdateInput } from '../../globalTypes'
import { routes } from '../../util/routes'
import { STUDENT_DETAIL_QUERY, STUDENT_UPDATE } from './StudentQuery'
import { StudentFragment } from './__generated__/StudentFragment'
import { StudentUpdate, StudentUpdateVariables } from './__generated__/StudentUpdate'

interface StudentDetailProps {
  student: StudentFragment
}

const StudentDetail: FC<StudentDetailProps> = ({ student }) => {
  const { email, firstName, lastName, lessons, phone, age, id, address,studentId } =
student

  const [studentUpdate, { loading }] = useMutation<StudentUpdate,
StudentUpdateVariables>(STUDENT_UPDATE, {
    refetchQueries: [{ query: STUDENT_DETAIL_QUERY, variables: { id } }],
  })

  const formMethods = useForm<StudentUpdateInput>({
    defaultValues: {
```

```
      firstName,
      lastName,
      email,
      phone,
      age,
      address,
      studentId,
    },
  })

  const onSubmitHandler = async (data: StudentUpdateInput) => {
    await studentUpdate({
      variables: {
        studentUpdateId: id,
        studentUpdateInput: { ...data, age: Number(data.age) },
      },
    })
  }

  const lessonsColumn: GridColDef[] = useMemo(
    () => [
      primaryColumn({
        field: 'id',
        headerName: 'Ders Adı',
        renderCell: ({ row, id }) => <TableLink href={{ pathname: routes.lesson, query:
{ id } }} text={row.name} />,
      }),
    ],
    []
  )

  return (
    <Box>
      <StudentForm methods={formMethods} loading={loading}
onSubmit={formMethods.handleSubmit(onSubmitHandler)} />

      <Box marginTop={4}>
        <DataTable
          loading={loading}
          rows={lessons}
          columns={lessonsColumn}
          searchableFields={['name']}
          density='compact'
        />
      </Box>
    </Box>
  )
}

export default StudentDetail
```

```
import { gql, useQuery } from '@apollo/client'
import { GridColDef } from '@mui/x-data-grid'
import { useMemo } from 'react'
import DataTable, { primaryColumn, TableLink, textColumn } from '../../DataTable'
import { routes } from '../../util/routes'
import { StudentsQuery } from './__generated__/StudentsQuery'

export const STUDENTS_QUERY = gql`
 query StudentsQuery {
   students {
     id
     firstName
     lastName
     email
     phone
     age
   }
 }
`

const Students = () => {
 const { data, loading } = useQuery<StudentsQuery>(STUDENTS_QUERY)

 const columns: GridColDef[] = useMemo(
   () => [
     primaryColumn({
       field: 'firstName',
       headerName: 'Ad - Soyad',
       renderCell: ({ row, id }) => (
         <TableLink href={{ pathname: routes.student, query: { id } }}
text={`${row.firstName} ${row.lastName}`} />
       ),
     }),
     textColumn({
       field: 'email',
       headerName: 'E-Posta',
     }),
     textColumn({
       field: 'phone',
       headerName: 'Telefon',
     }),
     textColumn({
       width: 75,
       field: 'age',
       headerName: 'Yaş',
     }),
   ],
   []
 )

 return (
```

```
    <DataTable
      loading={!data || loading}
      rows={data?.students || []}
      columns={columns}
      searchableFields={['firstName', 'lastName', 'email', 'phone']}
    />
  )
}

export default Students

import { gql, useMutation } from '@apollo/client'
import { useRouter } from 'next/router'
import { useForm } from 'react-hook-form'
import TeacherForm from '../../components/TeacherForm'
import { TeachersQuery } from '../Teachers/Teachers'
import { TeacherCreateVariables, TeacherCreate } from './__generated__/TeacherCreate'

type TeacherCreateInput = TeacherCreateVariables['teacherCreateInput']

const TEACHER_CREATE = gql`
  mutation TeacherCreate($teacherCreateInput: TeacherCreateInput!) {
    teacherCreate(teacherCreateInput: $teacherCreateInput) {
      id
    }
  }
`

const TeacherCreate = () => {
  const router = useRouter()
  const [teacherCreate, { loading }] = useMutation<TeacherCreate,
TeacherCreateVariables>(TEACHER_CREATE, {
    refetchQueries: [{ query: TeachersQuery }],
    awaitRefetchQueries: true,
  })

  const formMethods = useForm<TeacherCreateInput>({
    defaultValues: {
      firstName: '',
      lastName: '',
      email: '',
      phone: '',
      age: 18,
      address: '',
    },
  })

  const onSubmitHandler = async (data: TeacherCreateInput) => {
    const result = await teacherCreate({
      variables: {
        teacherCreateInput: { ...data, age: Number(data.age) },
```

```
      },
    })
    if (!result.data?.teacherCreate) return
    router.push({ pathname: '/teacher/[id]', query: { id: result.data.teacherCreate.id
} })
  }

  return <TeacherForm methods={formMethods} loading={loading}
onSubmit={formMethods.handleSubmit(onSubmitHandler)} />
}

export default TeacherCreate

import { useMutation } from '@apollo/client'
import { Box } from '@mui/material'
import { GridColDef } from '@mui/x-data-grid'
import { FC, useMemo } from 'react'
import { useForm } from 'react-hook-form'
import TeacherForm from '../../components/TeacherForm'
import DataTable, { primaryColumn, TableLink } from '../../DataTable'
import { TeacherUpdateInput } from '../../globalTypes'
import { routes } from '../../util/routes'
import { TEACHER_DETAIL_QUERY, TEACHER_UPDATE } from './TeacherQuery'
import { TeacherFragment } from './__generated__/TeacherFragment'
import { TeacherUpdate, TeacherUpdateVariables } from './__generated__/TeacherUpdate'

interface TeacherDetailProps {
  teacher: TeacherFragment
}

const TeacherDetail: FC<TeacherDetailProps> = ({ teacher }) => {
  const { email, firstName, lastName, lessons, phone, age, id, address } = teacher

  const [teacherUpdate, { loading }] = useMutation<TeacherUpdate,
TeacherUpdateVariables>(TEACHER_UPDATE, {
    refetchQueries: [{ query: TEACHER_DETAIL_QUERY, variables: { id } }],
  })

  const formMethods = useForm<TeacherUpdateInput>({
    defaultValues: {
      firstName,
      lastName,
      email,
      phone,
      age,
      address,
    },
  })

  const onSubmitHandler = async (data: TeacherUpdateInput) => {
    await teacherUpdate({
```

```
      variables: {
        teacherUpdateId: id,
        teacherUpdateInput: { ...data, age: Number(data.age) },
      },
    })
  }

  const lessonsColumn: GridColDef[] = useMemo(
    () => [
      primaryColumn({
        field: 'id',
        headerName: 'Ders Adı',
        renderCell: ({ row, id }) => <TableLink href={{ pathname: routes.lesson, query:
{ id } }} text={row.name} />,
      }),
    ],
    []
  )

  return (
    <Box>
      <TeacherForm methods={formMethods} loading={loading}
onSubmit={formMethods.handleSubmit(onSubmitHandler)} />

      <Box marginTop={4}>
        <DataTable
          loading={loading}
          rows={lessons}
          columns={lessonsColumn}
          searchableFields={['name']}
          density='compact'
        />
      </Box>
    </Box>
  )
}

export default TeacherDetail

import { gql, useQuery } from '@apollo/client'
import { GridColDef } from '@mui/x-data-grid'
import { useMemo } from 'react'
import DataTable, { primaryColumn, TableLink, textColumn } from '../../DataTable'
import { routes } from '../../util/routes'

export const TeachersQuery = gql`
  query TeachersQuery {
    teachers {
      id
      firstName
      lastName
```

```
      email
      phone
      age
    }
  }
`

const Teachers = () => {
 const { data, loading } = useQuery(TeachersQuery)

 const columns: GridColDef[] = useMemo(
   () => [
     primaryColumn({
       field: 'firstName',
       headerName: 'Ad - Soyad',
       renderCell: ({ row, id }) => (
         <TableLink href={{ pathname: routes.teacher, query: { id } }}
text={`${row.firstName} ${row.lastName}`} />
       ),
     }),
     textColumn({
       field: 'email',
       headerName: 'E-Posta',
     }),
     textColumn({
       field: 'phone',
       headerName: 'Telefon',
     }),
     textColumn({
       width: 75,
       field: 'age',
       headerName: 'Yaş',
     }),
   ],
   []
 )
 return (
   <DataTable
     loading={!data || loading}
     rows={data?.teachers || []}
     columns={columns}
     searchableFields={['firstName', 'lastName', 'email', 'phone']}
   />
 )
}

export default Teachers

import { Box, TextField } from '@mui/material'
import { DataGrid, DataGridProps } from '@mui/x-data-grid'
import { ReactNode, useMemo, useState } from 'react'
```

```
interface DataTableProps extends DataGridProps {
 searchableFields?: string[]
 searchPlaceholder?: string
 headerSlot?: ReactNode
}

const DataTable = ({ searchableFields, headerSlot, searchPlaceholder = 'Ara',
...otherProps }: DataTableProps) => {
 const [searchTerm, setSearchTerm] = useState('')

 const data = useMemo(() => {
   if (!searchTerm || !searchableFields?.length) return otherProps.rows

   const searchTermLower = searchTerm.toLocaleLowerCase('tr-TR')

   return otherProps.rows.filter((row) => {
     return searchableFields.some((field) => {
       const value = row[field]?.toLocaleLowerCase('tr-TR')
       return value && value.includes(searchTermLower)
     })
   })
 }, [otherProps.rows, searchTerm, searchableFields])

 return (
   <div style={{ width: '100%', display: 'flex', flexDirection: 'column' }}>
     <Box display='flex'>
       {headerSlot}
       <TextField
         variant='outlined'
         label={searchPlaceholder}
         placeholder={searchPlaceholder}
         sx={{
           marginLeft: 'auto',
           marginRight: 0,
           marginBottom: otherProps.density === 'compact' ? '0.5rem' : '1rem',
         }}
         size='small'
         value={searchTerm}
         onChange={(e) => setSearchTerm(e.target.value)}
       />
     </Box>
     <DataGrid
       {...otherProps}
       rows={data}
       autoHeight
       hideFooterSelectedRowCount
       sortingOrder={['asc', 'desc']}
       disableSelectionOnClick
       disableColumnMenu
     />
```

```
    </div>
  )
}

export default DataTable
```

## Mobil Uygulama Kaynak Kodları

```
import React, { useCallback, useEffect } from 'react';
import { StatusBar, StyleSheet } from 'react-native';
import { SafeAreaProvider, SafeAreaView } from 'react-native-safe-area-context';
import ApolloProvider from './lib/apollo/apollo';
import { colors } from './lib/theme/globalTheme';
import NativeBaseProvider from './lib/theme/theme';
import { BottomNavigation } from './screens/Navigation';

import dayjs from 'dayjs';
import tr from 'dayjs/locale/tr';
import isBetween from 'dayjs/plugin/isBetween';
import isToday from 'dayjs/plugin/isToday';
import { useAuthStore } from './store';
import Spinner from './components/Spinner';
import Login from './screens/Login';
dayjs.locale(tr);
dayjs.extend(isBetween);
dayjs.extend(isToday);

const Screen = () => {
  const { user, loading, initalize } = useAuthStore();

  const login = useCallback(async () => {
    await initalize();
  }, [initalize]);

  useEffect(() => {
    login();
  }, [login]);

  return (
    <Spinner loading={loading}>
      {user ? <BottomNavigation /> : <Login />}
    </Spinner>
  );
};

export default function App() {
  return (
    <ApolloProvider>
      <NativeBaseProvider>
        <SafeAreaProvider>
```

```
          <SafeAreaView style={styles.SafeAreaView}>
            <StatusBar barStyle={'light-content'} />
            <Screen />
          </SafeAreaView>
        </SafeAreaProvider>
      </NativeBaseProvider>
    </ApolloProvider>
  );
}

const styles = StyleSheet.create({
  SafeAreaView: {
    flex: 1,
    backgroundColor: colors.bg,
  },
});

import { gql, useQuery } from '@apollo/client';
import { NativeStackScreenProps } from '@react-navigation/native-stack';
import {
  Badge,
  Box,
  Center,
  Heading,
  HStack,
  Pressable,
  Text,
  VStack,
} from 'native-base';
import React, { useCallback, useEffect, useMemo, useState } from 'react';
import dayjs from 'dayjs';
import { FlatList, Linking, Platform, StyleSheet } from 'react-native';
import Icon from 'react-native-vector-icons/MaterialIcons';
import { LessonStackParams } from '../Navigation/types';
import {
  LessonDetail as LessonDetailResponse,
  LessonDetailVariables,
  LessonDetail_lesson_schedules_classroom,
} from './__generated__/LessonDetail';
import { useAuthStore } from '../../store';
import Card from '../../components/Card';
import { dateRangeToString } from '../../utils/dateRangeToString';
import { colors } from '../../lib/theme/globalTheme';
import Spinner from '../../components/Spinner';

type Props = NativeStackScreenProps<LessonStackParams, 'LessonDetail'>;

const GET_LESSON_DETAIL = gql`
  query LessonDetail($lessonId: String!) {
    lesson(id: $lessonId) {
      id
      name
```

```
      schedules {
        id
        startDate
        endDate
        attendedList {
          id
        }
        classroom {
          id
          name
          radius
          longitude
          latitude
        }
      }
      teachers {
        id
        email
        firstName
        lastName
      }
    }
  }
`;

const LessonDetail = ({ navigation, route }: Props) => {
  const { user } = useAuthStore();

  const { data, loading } = useQuery<
    LessonDetailResponse,
    LessonDetailVariables
  >(GET_LESSON_DETAIL, {
    variables: {
      lessonId: route.params.lessonId,
    },
  });

  const list = useMemo(() => {
    if (!data?.lesson?.schedules) {
      return [];
    }

    return data.lesson.schedules.map(schedule => {
      return {
        ...schedule,
        isAttended: !!schedule.attendedList.find(
          ({ id }) => id === user?.user.id,
        ),
      };
    });
  }, [data, user]);
```

```
const notAttendedCount = useMemo(() => {
  if (!list) {
    return {
      count: 0,
      percentage: '',
    };
  }

  const count = list.filter(({ endDate, isAttended }) => {
    const today = dayjs();
    return dayjs(endDate).isBefore(today) && !isAttended;
  }).length;
  const percentage = Math.round((count / list.length) * 100);
  return {
    count,
    percentage: `${percentage}%`,
  };
}, [list]);

const classooms = useMemo(() => {
  const classroomList: LessonDetail_lesson_schedules_classroom[] = [];

  list.forEach(schedule => {
    if (!classroomList.find(({ id }) => id === schedule.classroom.id)) {
      classroomList.push(schedule.classroom);
    }
  });

  return classroomList;
}, [list]);

console.log(data);

const scheduleStatusBadge = useCallback(
  (startDate: dayjs.Dayjs, endDate: dayjs.Dayjs): keyof typeof styles => {
    const now = dayjs();
    const start = dayjs(startDate);
    const end = dayjs(endDate);

    if (now.isBetween(start, end)) {
      return 'activeBadge';
    }

    if (now.isAfter(end)) {
      return 'pastBadge';
    }

    if (now.isBefore(start)) {
      return 'futureBadge';
    }

    return 'defaultBadge';
```

```
  },
  [],
);

const onClassroomClick = useCallback(
  (classroom: LessonDetail_lesson_schedules_classroom) => {
    const { latitude, longitude } = classroom;

    const scheme = Platform.select({
      ios: 'maps:0,0?q=',
      android: 'geo:0,0?q=',
    });

    const latLng = `${latitude},${longitude}`;
    const label = `Sınıf - ${classroom.name}`;
    const url = Platform.select({
      ios: `${scheme}${label}@${latLng}`,
      android: `${scheme}${latLng}(${label})`,
    });
    if (!url) {
      return;
    }
    Linking.openURL(url);
  },
  [],
);

useEffect(() => {
  navigation.setOptions({ title: route.params.lessonName });
}, [navigation, route.params.lessonName]);

return (
  <Spinner loading={loading}>
    <Box>
      <Center mb={8}>
        <HStack space={8}>
          <Center
            p={4}
            rounded="sm"
            shadow="8"
            backgroundColor={colors.panel}>
            <Text>Toplam ders sayısı</Text>
            <Text fontWeight={'bold'}>{list.length}</Text>
          </Center>

          <Center
            p={4}
            rounded="sm"
            shadow="8"
            backgroundColor={colors.panel}>
            <Text>Toplam devamsızlık</Text>
            <Text fontWeight={'bold'} color="danger.500">
```

```
        {notAttendedCount.count} ({notAttendedCount.percentage})
      </Text>
    </Center>
  </HStack>
</Center>

<Box mb={8}>
  <Heading size="sm">Sınıflar</Heading>
  <HStack space={4} mt={2}>
    {classooms.map(classroom => {
      const { id, name } = classroom;
      return (
        <Pressable key={id} onPress={() => onClassroomClick(classroom)}>
          <Badge>
            <Text>{name}</Text>
          </Badge>
        </Pressable>
      );
    })}
  </HStack>
</Box>

<Box mb={2}>
  <Heading size="sm">Ders Programı Listesi</Heading>
</Box>
<FlatList
  data={list}
  renderItem={({ item }) => {
    const startDate = dayjs(item.startDate);
    const endDate = dayjs(item.endDate);
    const { classroom, isAttended } = item;
    const dateToText = dateRangeToString(startDate, endDate);
    const badge = scheduleStatusBadge(startDate, endDate);

    return (
      <Card marginBottom={2} padding={2}>
        <VStack justifyContent="space-between">
          <Text fontSize="xs" variant="quote">
            {dateToText}
          </Text>
          <Box alignItems="flex-start" mt="2">
            <Badge>{classroom.name}</Badge>
          </Box>
        </VStack>
        <Box
          alignItems="center"
          position="absolute"
          bottom="0"
          right="0">
          {isAttended && (
            <Badge display="flex" flexDirection="row">
              <Text fontSize="8px" mr="4px">
```

```
                      İMZALANDI
                    </Text>
                    <Icon size={12} color="#22c55e" name="check-circle" />
                  </Badge>
                )}
                {!isAttended && badge === 'pastBadge' && (
                  <Badge display="flex" flexDirection="row">
                    <Text fontSize="8px" mr="4px">
                      İMZALANMADI
                    </Text>
                    <Icon size={12} color="#f87171" name="warning" />
                  </Badge>
                )}
              </Box>
            </Card>
          );
        }}
        keyExtractor={item => item.id}
      />
    </Box>
  </Spinner>
);
};

const styles = StyleSheet.create({
 activeBadge: {
   backgroundColor: '#22c55e',
   shadowColor: '#4ade80',
   shadowOffset: {
     width: 0,
     height: 0,
   },
   shadowOpacity: 0.8,
   shadowRadius: 3,
   elevation: 1,
 },
 pastBadge: {
   backgroundColor: '#f87171',
   shadowColor: '#ef4444',
   shadowOffset: {
     width: 0,
     height: 0,
   },
   shadowOpacity: 0.8,
   shadowRadius: 3,
   elevation: 1,
 },
 futureBadge: {
   backgroundColor: '#0ea5e9',
   shadowColor: '#38bdf8',
   shadowOffset: {
     width: 0,
```

```
    height: 0,
  },
  shadowOpacity: 0.8,
  shadowRadius: 3,
  elevation: 1,
 },

 defaultBadge: {},
});

export default LessonDetail;

import React from 'react';
import { gql, useQuery } from '@apollo/client';
import { Divider, Pressable, ScrollView, VStack, Text } from 'native-base';
import Card from '../../components/Card';
import Spinner from '../../components/Spinner';
import { GetLessons } from './__generated__/GetLessons';
import { LessonStackParams } from '../Navigation/types';
import { NativeStackScreenProps } from '@react-navigation/native-stack';

type Props = NativeStackScreenProps<LessonStackParams, 'LessonList'>;

const GET_LESSONS = gql`
 query GetLessons {
   clientLessons {
     id
     name
     teachers {
       firstName
       lastName
     }
   }
 }
`;

const LessonList = ({ navigation }: Props) => {
 const { data, loading } = useQuery<GetLessons>(GET_LESSONS);

 const lessons = data?.clientLessons || [];

 return (
   <Spinner loading={loading}>
     <ScrollView>
       <VStack space={4}>
         {lessons.map((lesson, i) => {
           return (
             <Pressable
               key={i}
               w={'100%'}
               onPress={() =>
                 navigation.navigate('LessonDetail', {
```

```
                  lessonId: lesson.id,
                  lessonName: lesson.name,
                })
              }>
              <Card>
                <Text>{lesson?.name}</Text>
                <Divider my="2" />
                <Text>
                  {lesson?.teachers
                    ?.map(
                      teacher => `${teacher?.firstName} ${teacher?.lastName}`,
                    )
                    .join(', ')}
                </Text>
              </Card>
            </Pressable>
          );
        })}
      </VStack>
    </ScrollView>
  </Spinner>
 );
};

export default LessonList;

import React, { useCallback, useState } from 'react';
import {
 Box,
 Button,
 Center,
 Heading,
 Input,
 Stack,
 useToast,
} from 'native-base';
import Icon from 'react-native-vector-icons/MaterialIcons';
import { gql, useMutation } from '@apollo/client';
import {
 StudentLogin,
 StudentLoginVariables,
} from './__generated__/StudentLogin';
import { colors } from '../../lib/theme/globalTheme';
import { useAuthStore } from '../../store';
import LottieView from 'lottie-react-native';
import { StyleSheet } from 'react-native';

const LOGIN = gql`
 mutation StudentLogin($input: AuthInput!) {
   studentLogin(input: $input) {
     accessToken
     me {
```

```
        id
        firstName
        lastName
        email
        age
        email
        studentId
        phone
        address
      }
    }
  }
`;

const Login = () => {
  const toast = useToast();
  const [show, setShow] = useState(false);
  const [loginInput, setLoginInput] = useState({
    email: '',
    password: '',
  });
  const { setUser } = useAuthStore();
  const [login, { loading, reset }] = useMutation<
    StudentLogin,
    StudentLoginVariables
  >(LOGIN);

  const handleLogin = useCallback(async () => {
    const { email, password } = loginInput;
    try {
      const response = await login({
        variables: {
          input: {
            email,
            password,
          },
        },
      });

      const data = response.data?.studentLogin;
      if (!data?.accessToken || !data?.me) {
        return;
      }
      const { accessToken, me: user } = data;
      toast.show({
        title: `${user.firstName}, Hoşgeldin.`,
        status: 'success',
      });
      setUser({ accessToken, user });
    } catch (err: any) {
      toast.show({
        title: 'E-posta veya parola yanlış',
```

```
      status: 'error',
    });
    reset();
  }
}, [login, loginInput, reset, setUser, toast]);

return (
  <Box height="100%">
    <Center mb={2}>
      <LottieView
        source={require('./logo.json')}
        autoPlay
        loop
        speed={2}
        style={styles.logo}
      />
      <Heading fontSize={'3xl'}>Buradaa</Heading>
    </Center>
    <Stack space={4} w="100%" alignItems="center">
      <Input
        value={loginInput.email}
        onChangeText={text => setLoginInput({ ...loginInput, email: text })}
        size="2xl"
        w={{
          base: '75%',
          md: '25%',
        }}
        InputLeftElement={
          <Box ml="8px">
            <Icon size={18} color={colors.quote} name="person" />
          </Box>
        }
        placeholder="E-Posta"
      />
      <Input
        value={loginInput.password}
        onChangeText={text =>
          setLoginInput({ ...loginInput, password: text })
        }
        size="2xl"
        w={{
          base: '75%',
          md: '25%',
        }}
        type={show ? 'text' : 'password'}
        InputLeftElement={
          <Box ml="8px">
            <Icon size={18} color={colors.quote} name="lock" />
          </Box>
        }
        InputRightElement={
          <Box mr="8px">
```

```
              <Icon
                size={18}
                color={colors.quote}
                name={show ? 'visibility' : 'visibility-off'}
                onPress={() => setShow(!show)}
              />
            </Box>
          }
          placeholder="Parola"
        />

        <Button
          w={{
            base: '75%',
            md: '25%',
          }}
          isLoading={loading}
          onPress={handleLogin}>
          Giriş
        </Button>
      </Stack>
    </Box>
  );
};

const styles = StyleSheet.create({
  logo: {
    width: 200,
    height: 300,
  },
});

export default Login;

import * as React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import Icon from 'react-native-vector-icons/MaterialCommunityIcons';
import FontAwesome from 'react-native-vector-icons/FontAwesome';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { colors, globalStyles } from '../../lib/theme/globalTheme';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import LessonDetail from '../LessonDetail';
import LessonList from '../LessonList';
import { LessonStackParams } from './types';
import UpcomingEventsList from '../UpcomingEventsList';
import Profile from '../Profile';

const Tab = createBottomTabNavigator();
const LessonStack = createNativeStackNavigator<LessonStackParams>();

function LessonStackScreen() {
  return (
```

```
    <LessonStack.Navigator
      screenOptions={{
        headerTitleStyle: { color: colors.title },
        headerStyle: { backgroundColor: colors.bg },
        contentStyle: { backgroundColor: colors.bg },
      }}>
      <LessonStack.Screen
        options={{ title: 'DERSLER' }}
        name={'LessonList'}
        component={LessonList}
      />
      <LessonStack.Screen
        options={{ title: '' }}
        name={'LessonDetail'}
        component={LessonDetail}
      />
    </LessonStack.Navigator>
  );
}

const getColor = (focused: boolean) => {
  return focused ? '#e2e8f0' : '#374151';
};

const BottomNavigation = () => {
  return (
    <NavigationContainer>
      <Tab.Navigator
        safeAreaInsets={{ bottom: 0, left: 0, right: 0, top: 0 }}
        sceneContainerStyle={globalStyles.sceneContainerStyle}
        initialRouteName={'UpcomingEvents'}
        screenOptions={{
          headerShown: false,
          tabBarStyle: {
            backgroundColor: colors.bg,
            display: 'flex',
            height: 48,
            justifyContent: 'center',
            alignItems: 'center',
          },
          tabBarLabel: () => null,
        }}>
        <Tab.Screen
          name={'UpcomingEvents'}
          component={UpcomingEventsList}
          options={{
            tabBarIcon: ({ focused }) => (
              <Icon name="calendar-clock" size={24} color={getColor(focused)} />
            ),
          }}
        />
        <Tab.Screen
```

```
          name={'LessonStack'}
          component={LessonStackScreen}
          options={{
            tabBarIcon: ({ focused }) => (
              <Icon name="book" size={24} color={getColor(focused)} />
            ),
          }}
        />
        <Tab.Screen
          name={'Profile'}
          component={Profile}
          options={{
            tabBarIcon: ({ focused }) => (
              <FontAwesome name="user" size={24} color={getColor(focused)} />
            ),
          }}
        />
      </Tab.Navigator>
    </NavigationContainer>
  );
};

export default BottomNavigation;

import * as React from 'react';
import MaterialIcons from 'react-native-vector-icons/MaterialIcons';
import {
  Box,
  Button,
  Center,
  Divider,
  FormControl,
  Heading,
  HStack,
  Input,
  ScrollView,
  Text,
  VStack,
} from 'native-base';
import { useAuthStore } from '../../store';
import { StyleSheet } from 'react-native';
import { colors } from '../../lib/theme/globalTheme';

interface FormInputProps {
  label: string;
  value: string;
}

const FormInput = ({ label, value }: FormInputProps) => {
  return (
    <Box alignItems="center">
      <FormControl>
```

```
      <FormControl.Label style={styles.label}>
        <Text color={colors.subtitle} fontSize="xs">
          {label}
        </Text>
      </FormControl.Label>
      <Input size="lg" isReadOnly placeholder="" value={value} />
    </FormControl>
  </Box>
 );
};

const Profile = () => {
 const { user: data, deleteUser } = useAuthStore();

 if (!data) {
   return null;
 }

 const { firstName, lastName, phone, address, age, studentId, email } =
   data.user;

 return (
   <ScrollView>
     <VStack space={4}>
       <FormInput label="Ad" value={firstName} />
       <FormInput label="Soyad" value={lastName} />
       <FormInput label="E-Posta" value={email} />
       <FormInput label="Öğrenci numarası" value={studentId} />
       <FormInput label="Yaş" value={age.toString()} />
       <FormInput label="Adres" value={address} />
       <FormInput label="Telefon" value={phone} />
       <Button>Kaydet</Button>
     </VStack>
     <Divider mt={4} mb={4} backgroundColor="blueGray.800" />
     <HStack alignItems="center">
       <Input size="lg" placeholder="Şifre" flex={1} />
       <Box ml={4}>
         <Button>Değiştir</Button>
       </Box>
     </HStack>

     <Divider mt={4} mb={4} backgroundColor="blueGray.800" />
     <Button onPress={deleteUser} colorScheme="secondary">
       Çıkış yap
     </Button>
   </ScrollView>
 );
};

const styles = StyleSheet.create({
 label: {
   marginBottom: 2,
```

```
 },
});

export default Profile;

import React, { useMemo } from 'react';
import { useQuery } from '@apollo/client';
import dayjs from 'dayjs';

import Spinner from '../../components/Spinner';
import TodayList from './TodayList';

import { createMaterialTopTabNavigator } from '@react-navigation/material-top-tabs';
import WeekList from './WeekList';
import { colors } from '../../lib/theme/globalTheme';
import { UpcomingEvents } from './__generated__/UpcomingEvents';
import { StyleSheet } from 'react-native';
import { UPCOMING_EVENTS } from './ScheduleQueries';

const Tab = createMaterialTopTabNavigator();

const UpcomingEventsList = () => {
 const { data, loading, refetch } = useQuery<UpcomingEvents>(UPCOMING_EVENTS);

 const schedules = data?.currentWeekSchedule || [];

 const todayList = useMemo(() => {
   if (!data?.currentWeekSchedule) {
     return [];
   }

   return data?.currentWeekSchedule.filter(({ startDate }) =>
     dayjs(startDate).isToday(),
   );
 }, [data?.currentWeekSchedule]);

 return (
   <Spinner loading={loading}>
     <Tab.Navigator
       sceneContainerStyle={styles.sceneContainerStyle}
       screenOptions={{
         tabBarLabelStyle: {
           fontSize: 12,
           color: colors.subtitle,
         },
         tabBarStyle: { backgroundColor: colors.bg },
       }}>
       <Tab.Screen
         name="Yaklaşan Dersler"
         children={() => (
           <TodayList
             list={todayList}
```

```
              refreshing={loading}
              onRefresh={refetch}
            />
          )}
        />
        <Tab.Screen
          name="Haftalık Program"
          children={() => <WeekList list={schedules} />}
        />
      </Tab.Navigator>
    </Spinner>
  );
};

export const styles = StyleSheet.create({
  sceneContainerStyle: {
    backgroundColor: colors.bg,
    paddingHorizontal: 4,
    paddingVertical: 16,
  },
});

export default UpcomingEventsList;

import React, { FC, useCallback, useState } from 'react';
import { ScrollView, Text, VStack, useToast } from 'native-base';
import dayjs from 'dayjs';
import { UpcomingEvents_currentWeekSchedule as Schedule } from
'./__generated__/UpcomingEvents';
import ScheduleSign from './ScheduleSign';
import ScheduleCard from './ScheduleCard';
import { RefreshControl } from 'react-native';

interface TodayListProps {
  list: Schedule[];
  onRefresh: () => void;
  refreshing: boolean;
}

const TodayList: FC<TodayListProps> = ({ list, refreshing, onRefresh }) => {
  const toast = useToast();
  const [selectedSchedule, setSelectedSchedule] = useState<Schedule | null>(
    null,
  );

  const handleSchedulePress = useCallback(
    (schedule: Schedule) => {
      const now = dayjs();
      const start = dayjs(schedule.startDate);
      const end = dayjs(schedule.endDate);

      if (schedule.isAttended) {
```

```
      toast.show({
        title: 'Yoklama zaten imzalandı',
        status: 'info',
      });
      return;
    }

    if (now.isAfter(end)) {
      toast.show({
        title: 'Ders açık değil',
        status: 'warning',
      });
      return;
    }

    if (now.isBefore(start)) {
      toast.show({
        title: 'Ders daha başlamadı',
        status: 'info',
      });
      return;
    }

    setSelectedSchedule(schedule);
  },
  [toast],
);

if (!list.length) {
  return <Text>Bugün dersin yok</Text>;
}

return (
  <>
    <ScrollView
      refreshControl={
        <RefreshControl refreshing={refreshing} onRefresh={onRefresh} />
      }>
      <VStack space={4}>
        {list.map(schedule => {
          return (
            <ScheduleCard
              key={schedule.id}
              schedule={schedule}
              onPress={handleSchedulePress}
            />
          );
        })}
      </VStack>
    </ScrollView>
    {!!selectedSchedule && (
      <ScheduleSign
```

```
          schedule={selectedSchedule}
          onClose={() => setSelectedSchedule(null)}
        />
      )}
    </>
  );
};

export default TodayList;

import { Box } from 'native-base';
import React, { FC, useMemo } from 'react';
import { Dimensions } from 'react-native';
import Calendar from 'react-native-big-calendar';
import { UpcomingEvents_currentWeekSchedule } from './__generated__/UpcomingEvents';

const heigth = Dimensions.get('window').height;

const darkTheme = {
  palette: {
    primary: {
      main: '#6185d0',
      contrastText: '#000',
    },
    gray: {
      '100': '#333',
      '200': '#666',
      '300': '#888',
      '500': '#aaa',
      '800': '#ccc',
    },
  },
};

interface WeekListProps {
  list: UpcomingEvents_currentWeekSchedule[];
}

const WeekList: FC<WeekListProps> = ({ list }) => {
  const events = useMemo(
    () =>
      list.map(event => ({
        title: `${event.classroom.name} - ${event.lesson.name}`,
        start: new Date(event.startDate),
        end: new Date(event.endDate),
        id: event.id,
      })),
    [list],
  );

  return (
    <Box bg="blueGray.800" py={8} rounded={4} shadow={8}>
```

```
      <Calendar
        locale="tr"
        events={events}
        height={heigth - 250}
        theme={darkTheme}
        mode="custom"
        weekStartsOn={1}
        weekEndsOn={5}
      />
    </Box>
  );
};

export default WeekList;

import create from 'zustand';

import AsyncStorage from '@react-native-async-storage/async-storage';

export const USER_STORAGE_KEY = '@user';

export interface User {
  id: string;
  firstName: string;
  lastName: string;
  email: string;
  age: number;
  studentId: string;
  phone: string;
  address: string;
}

export interface AuthUser {
  accessToken: string;
  user: User;
}

export interface AuthStore {
  user: AuthUser | null;
  loading: boolean;
  initalize: () => Promise<AuthUser | null>;
  setUser: (authUser: AuthUser) => void;
  deleteUser: () => void;
}

export const useAuthStore = create<AuthStore>(set => {
  return {
    loading: true,
    user: null,
    initalize: async () => {
      const user = await AsyncStorage.getItem(USER_STORAGE_KEY);
      const parsedUser: AuthUser | null = user ? JSON.parse(user) : null;
```

```
      set({
        loading: false,
        user: parsedUser,
      });

      return parsedUser;
    },
    setUser: async (authUser: AuthUser) => {
      await AsyncStorage.setItem(USER_STORAGE_KEY, JSON.stringify(authUser));
      set(() => ({ user: authUser }));
    },
    deleteUser: async () => {
      await AsyncStorage.removeItem(USER_STORAGE_KEY);
      set(() => ({ user: null }));
    },
  };
});
```

## 4.3 Buradaa Web Servis Kaynak Kodları

```
import { Args, Mutation, Resolver } from '@nestjs/graphql';
import { AuthResponseStudent, AuthResponseTeacher } from './auth.response';
import { AuthInput } from './auth.input';
import { Inject } from '@nestjs/common';
import { PrismaService } from '../prisma.service';
import { JwtHelperService } from '../security/services/jwt-helper.service';
import bcrypt from 'bcrypt';

@Resolver()
export class AuthResolver {
 constructor(@Inject(PrismaService) private prismaService: PrismaService, private jwtHelper:
JwtHelperService) {}

 @Mutation(() => AuthResponseStudent)
 async studentLogin(@Args('input') input: AuthInput): Promise<AuthResponseStudent> {
   const { email, password } = input;

   const foundUser = await this.prismaService.student.findUnique({
     where: {
       email,
     },
     rejectOnNotFound: true,
   });

   const compare = bcrypt.compareSync(password, foundUser.password);

   if (!compare) {
     throw new Error('Invalid credentials');
   }

   return {
```

```
      accessToken: this.jwtHelper.sign({ id: foundUser.id, email: foundUser.email, role: foundUser.role
}),
      me: foundUser,
    };
  }

  @Mutation(() => AuthResponseTeacher)
  async teacherLogin(@Args('input') input: AuthInput): Promise<AuthResponseTeacher> {
    const { email, password } = input;

    const foundUser = await this.prismaService.teacher.findUnique({
      where: {
        email,
      },
      rejectOnNotFound: true,
    });

    const compare = bcrypt.compareSync(password, foundUser.password);

    if (!compare) {
      throw new Error('Invalid credentials');
    }

    return {
      accessToken: this.jwtHelper.sign({ id: foundUser.id, email: foundUser.email, role: foundUser.role
}),
      me: foundUser,
    };
  }
}

import { Resolver, Query, Mutation, Args } from '@nestjs/graphql';
import { Classroom } from 'src/clasroom/clasroom.model';
import { ClassroomCreateInput } from './dto/classroom-create.input';
import { ClassroomUpdateInput } from './dto/classroom-update.input';
import { PrismaService } from 'src/prisma.service';
import { Inject } from '@nestjs/common';

@Resolver(() => Classroom)
export class ClassroomResolver {
  constructor(@Inject(PrismaService) private prismaService: PrismaService) {}

  @Query(() => [Classroom])
  classrooms() {
    return this.prismaService.classroom.findMany();
  }

  @Query(() => Classroom, { nullable: true })
  classroom(@Args('id') id: string) {
    return this.prismaService.classroom.findUnique({ where: { id } });
  }

  @Mutation(() => Classroom, { nullable: true })
  classroomCreate(@Args('classroomCreateInput') classroomCreateInput: ClassroomCreateInput) {
    const { latitude, longitude, name, radius } = classroomCreateInput;
    return this.prismaService.classroom.create({
      data: {
        radius,
        latitude,
```

```
      longitude,
      name,
    },
  });
}

@Mutation(() => Classroom, { nullable: true })
classroomUpdate(@Args('id') id: string, @Args('classroomUpdateInput') classroomUpdateInput:
ClassroomUpdateInput) {
  const { latitude, longitude, name, radius } = classroomUpdateInput;
  return this.prismaService.classroom.update({
    where: { id },
    data: {
      radius,
      latitude,
      longitude,
      name,
    },
  });
}

@Mutation(() => Classroom)
classroomDelete(@Args('id') id: string) {
  return this.prismaService.classroom.delete({ where: { id } });
}
}
import { Resolver, Query, Mutation, Args, ResolveField, Parent } from '@nestjs/graphql';
import { Inject, UseGuards } from '@nestjs/common';
import { LessonUpdateInput } from './dto/lesson-update.input';
import { LessonCreateInput } from './dto/lesson-create.input';
import { PrismaService } from 'src/prisma.service';
import { LessonSchedule } from 'src/lessonSchedule/lessonSchedule.model';
import { Student } from 'src/student/student.model';
import { Lesson } from './lesson.model';

@Resolver(() => Lesson)
export class LessonResolver {
constructor(@Inject(PrismaService) private prismaService: PrismaService) {}

@Query(() => [Lesson])
lessons() {
  return this.prismaService.lesson.findMany();
}

@Query(() => Lesson, { nullable: true })
lesson(@Args('id') id: string) {
  const lesson = this.prismaService.lesson.findUnique({ where: { id } });
  return lesson;
}

@Mutation(() => Lesson, { nullable: true })
lessonCreate(@Args('lessonCreateInput') lessonCreateInput: LessonCreateInput) {
  const { name, year, studentIds, teacherIds } = lessonCreateInput;

  return this.prismaService.lesson.create({
    data: {
      name,
      year,
      students: {
```

```
          connect: studentIds.map((id) => ({ id })),
        },
        teachers: {
          connect: teacherIds.map((id) => ({ id })),
        },
      },
    });
  }

  @Mutation(() => Lesson, { nullable: true })
  lessonUpdate(@Args('id') id: string, @Args('lessonUpdateInput') lessonUpdateInput: LessonUpdateInput) {
    const { name, year, studentIds, teacherIds } = lessonUpdateInput;

    return this.prismaService.lesson.update({
      where: { id },
      data: {
        name,
        year,
        students: {
          set: studentIds.map((id) => ({ id })),
        },
        teachers: {
          set: teacherIds.map((id) => ({ id })),
        },
      },
    });
  }

  @Mutation(() => Lesson)
  lessonDelete(@Args('id') id: string) {
    return this.prismaService.lesson.delete({ where: { id } });
  }

  @ResolveField('schedules', () => [LessonSchedule], { nullable: true })
  async schedules(@Parent() parent: Lesson) {
    const schedules = await await this.prismaService.lessonSchedule.findMany({
      where: { lessonId: parent.id },
      orderBy: { startDate: 'asc' },
    });
    return schedules;
  }

  @ResolveField('studentCount', () => Number, { nullable: true })
  async studentCount(@Parent() parent: Lesson) {
    const count = await this.prismaService.student.count({ where: { lessons: { some: { id: parent.id } } }
} });
    return count;
  }

  @ResolveField('students', () => [Student])
  async students(@Parent() parent: Lesson) {
    const students = await this.prismaService.student.findMany({ where: { lessons: { some: { id:
parent.id } } } });
    return students;
  }

  @ResolveField('teachers', () => [Student])
  async teachers(@Parent() parent: Lesson) {
```

```
    const teachers = await this.prismaService.teacher.findMany({ where: { lessons: { some: { id:
parent.id } } } });
    return teachers;
  }
}

import { Resolver, Query, Mutation, Args, Parent, ResolveField } from '@nestjs/graphql';
import { Inject } from '@nestjs/common';
import { PrismaService } from 'src/prisma.service';
import { LessonScheduleCreateInput, LessonScheduleUpdateInput } from './dto/lesson-schedule.input';
import { Classroom } from 'src/clasroom/clasroom.model';
import { LessonSchedule, LessonScheduleCreateMany } from './lessonSchedule.model';
import { Student } from 'src/student/student.model';
import { Lesson } from 'src/lesson/lesson.model';
import { Prisma } from '@prisma/client';

@Resolver(() => LessonSchedule)
export class LessonScheduleResolver {
  constructor(@Inject(PrismaService) private prismaService: PrismaService) {}

  @Query(() => [LessonSchedule])
  lessonSchedules() {
    return this.prismaService.lessonSchedule.findMany();
  }

  @Mutation(() => LessonSchedule)
  lessonScheduleCreate(@Args('lessonScheduleCreateInput') lessonScheduleCreateInput:
LessonScheduleCreateInput) {
    const { startDate, endDate, lessonId, classroomId } = lessonScheduleCreateInput;

    return this.prismaService.lessonSchedule.create({
      data: {
        startDate,
        endDate,
        lesson: {
          connect: {
            id: lessonId,
          },
        },
        classroom: {
          connect: {
            id: classroomId,
          },
        },
      },
    });
  }

  @Mutation(() => LessonScheduleCreateMany)
  async lessonScheduleCreateMany(
    @Args({ name: 'lessonScheduleCreateManyInput', type: () => [LessonScheduleCreateInput] })
    lessonScheduleCreateManyInput: LessonScheduleCreateInput[],
  ) {
    const data: Prisma.Enumerable<Prisma.LessonScheduleCreateManyInput> =
lessonScheduleCreateManyInput.map(
      (lessonSchedule) => {
        const { startDate, endDate, lessonId, classroomId } = lessonSchedule;

        return {
```

```
        startDate,
        endDate,
        classroomId,
        lessonId,
      };
    },
  );

  const result = await this.prismaService.lessonSchedule.createMany({
    data: data,
  });

  return { count: result.count };
}

@Mutation(() => LessonSchedule, { nullable: true })
lessonScheduleUpdate(
  @Args('id') id: string,
  @Args('lessonScheduleUpdateInput') lessonScheduleUpdateInput: LessonScheduleUpdateInput,
) {
  const { startDate, endDate, classroomId } = lessonScheduleUpdateInput;

  return this.prismaService.lessonSchedule.update({
    where: { id },
    data: {
      startDate,
      endDate,
      classroom: {
        connect: {
          id: classroomId,
        },
      },
    },
  });
}

@Mutation(() => LessonSchedule)
lessonScheduleDelete(@Args('id') id: string) {
  return this.prismaService.lessonSchedule.delete({ where: { id } });
}

@ResolveField('lesson', () => Lesson)
async lesson(@Parent() parent: LessonSchedule) {
  const lesson = await this.prismaService.lesson.findUnique({
    where: { id: parent.lessonId },
  });
  return lesson;
}

@ResolveField('classroom', () => Classroom)
async classroom(@Parent() parent: LessonSchedule) {
  const classroom = await this.prismaService.classroom.findUnique({
    where: { id: parent.classroomId },
  });
  return classroom;
}

@ResolveField('attendedList', () => [Student])
async attendedList(@Parent() parent: LessonSchedule) {
```

```
    const students = await this.prismaService.student.findMany({
      where: { lessonSchedule: { some: { id: parent.id } } },
    });
    return students;
  }
}

import { Resolver, Query, Mutation, Args, ResolveField, Parent } from '@nestjs/graphql';
import { StudentCreateInput } from './dto/student-create.input';
import { StudentUpdateInput } from './dto/student-update.input';
import { PrismaService } from 'src/prisma.service';
import { Inject } from '@nestjs/common';
import { Lesson } from 'src/lesson/lesson.model';
import { Student } from './student.model';
import bcrypt from 'bcrypt';

@Resolver(() => Student)
export class StudentResolver {
  constructor(@Inject(PrismaService) private prismaService: PrismaService) {}

  @Query(() => [Student])
  students() {
    return this.prismaService.student.findMany();
  }

  @Query(() => Student, { nullable: true })
  student(@Args('id') id: string) {
    return this.prismaService.student.findUnique({ where: { id } });
  }

  @Mutation(() => Student, { nullable: true })
  studentCreate(@Args('studentCreateInput') studentCreateInput: StudentCreateInput) {
    const { firstName, lastName, email, age, address, phone, studentId, password } = studentCreateInput;
    const hashedPassword = bcrypt.hashSync(password, 10);

    return this.prismaService.student.create({
      data: {
        password: hashedPassword,
        studentId,
        firstName,
        lastName,
        email,
        age,
        address,
        phone,
      },
    });
  }

  @Mutation(() => Student, { nullable: true })
  studentUpdate(@Args('id') id: string, @Args('studentUpdateInput') studentUpdateInput:
StudentUpdateInput) {
    const { firstName, lastName, email, age, address, phone, studentId } = studentUpdateInput;
    return this.prismaService.student.update({
      where: { id },
      data: {
        studentId,
        firstName,
        lastName,
```

```
      email,
      age,
      address,
      phone,
    },
  });
 }

 @Mutation(() => Student)
 studentDelete(@Args('id') id: string) {
   return this.prismaService.student.delete({ where: { id } });
 }

 @ResolveField('lessons', () => [Lesson])
 async lessons(@Parent() parent: Student) {
   const lessons = await this.prismaService.lesson.findMany({
     where: { students: { some: { id: parent.id } } },
   });

   return lessons;
 }
}

import { Resolver, Query, Mutation, Args, ResolveField, Parent } from '@nestjs/graphql';
import { TeacherCreateInput } from './dto/teacher-create.input';
import { TeacherUpdateInput } from './dto/teacher-update.input';
import { PrismaService } from 'src/prisma.service';
import { Inject } from '@nestjs/common';
import { Lesson } from 'src/lesson/lesson.model';
import { Teacher } from './teacher.model';
import bcrypt from 'bcrypt';

@Resolver(() => Teacher)
export class TeacherResolver {
 constructor(@Inject(PrismaService) private prismaService: PrismaService) {}

 @Query(() => [Teacher])
 teachers() {
   return this.prismaService.teacher.findMany();
 }

 @Query(() => Teacher, { nullable: true })
 teacher(@Args('id') id: string) {
   return this.prismaService.teacher.findUnique({ where: { id } });
 }

 @Mutation(() => Teacher, { nullable: true })
 teacherCreate(@Args('teacherCreateInput') teacherCreateInput: TeacherCreateInput) {
   const { firstName, lastName, email, age, address, phone, password } = teacherCreateInput;

   const hashedPassword = bcrypt.hashSync(password, 10);

   return this.prismaService.teacher.create({
     data: {
       password: hashedPassword,
       firstName,
       lastName,
       email,
       age,
```

```
        address,
        phone,
      },
    });
  }

  @Mutation(() => Teacher, { nullable: true })
  teacherUpdate(@Args('id') id: string, @Args('teacherUpdateInput') teacherUpdateInput:
TeacherUpdateInput) {
    const { firstName, lastName, email, age, address, phone } = teacherUpdateInput;
    return this.prismaService.teacher.update({
      where: { id },
      data: {
        firstName,
        lastName,
        email,
        age,
        address,
        phone,
      },
    });
  }

  @Mutation(() => Teacher)
  teacherDelete(@Args('id') id: string) {
    return this.prismaService.teacher.delete({ where: { id } });
  }

  @ResolveField('lessons', () => [Lesson])
  async lessons(@Parent() parent: Teacher) {
    const lessons = await this.prismaService.lesson.findMany({
      where: { teachers: { some: { id: parent.id } } },
    });

    return lessons;
  }
}

import { join } from 'path';
import { Module } from '@nestjs/common';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloServerPluginLandingPageLocalDefault } from 'apollo-server-core';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { LessonModule } from './lesson/lesson.module';
import { StudentModule } from './student/student.module';
import { ClassroomModule } from './clasroom/clasroom.module';
import { LessonScheduleModule } from './lessonSchedule/lessonSchedule.module';
import { TeacherModule } from './teacher/teacher.module';
import { LessonResolver } from './lesson/lesson.resolver';
import { ClassroomResolver } from './clasroom/classroom.resolver';
import { LessonScheduleResolver } from './lessonSchedule/lessonSchedule.resolver';
import { StudentResolver } from './student/student.resolver';
import { TeacherResolver } from './teacher/teacher.resolver';
import { ClientModule } from './client/client.module';
import { SecurityModule } from './security/security.module';
import { ClientResolver } from './client/client.resolver';
import { AuthModule } from './auth/auth.module';

@Module({
```

```
  imports: [
    LessonModule,
    StudentModule,
    TeacherModule,
    ClassroomModule,
    LessonScheduleModule,
    ClientModule,
    AuthModule,
    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      buildSchemaOptions: { dateScalarMode: 'timestamp' },
      sortSchema: true,
      autoSchemaFile: join(process.cwd(), 'src/schema.gql'),
      introspection: true,
      playground: false,
      plugins: [ApolloServerPluginLandingPageLocalDefault()],
      installSubscriptionHandlers: true,
      context: ({ req }) => {
        return { req };
      },
      cors: {
        credentials: true,
        origin: true,
      },
    }),
    SecurityModule,
  ],
  providers: [
    LessonResolver,
    StudentResolver,
    TeacherResolver,
    ClassroomResolver,
    LessonScheduleResolver,
    ClientResolver,
  ],
})
export class AppModule {}
```