**Kullanılan Teknolojiler**

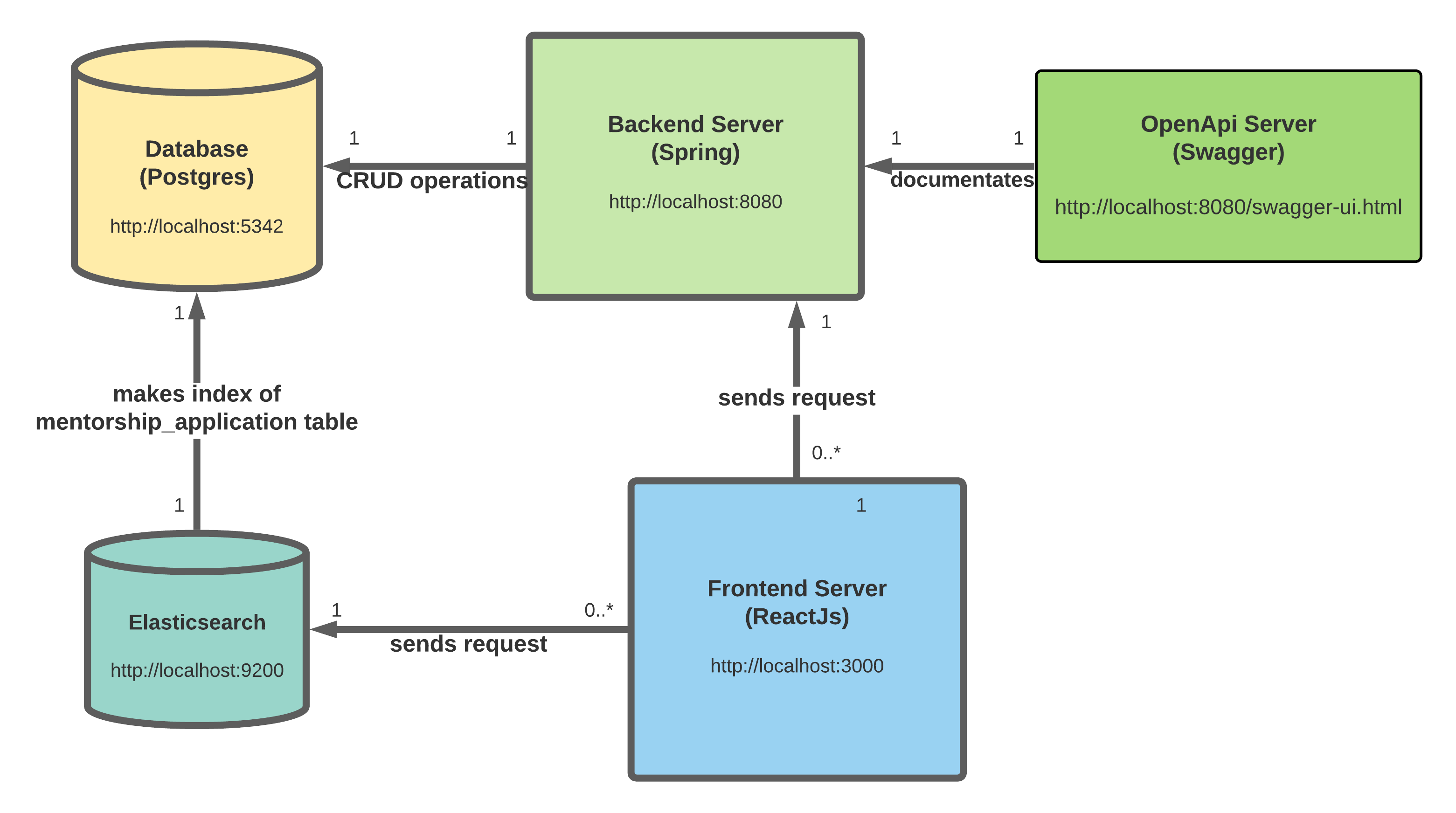
Uygulamada Spring Boot, React-js, PostgreSQL ve Elasticsearch sistemleri kullanıldı.

Spring ve React’ı seçme nedenleri arasında geniş kullanıcı kitlelerinin olması ve dolayısıyla karşılaşılan problemlerin çözümlerinin kolaylıkla bulunması, bu teknolojilere daha önceden aşina olmam ve öğrenme sürecini minimuma indirme isteğim etkili oldu. Özellikle Reactta template kütüphaneleri işimi oldukça kolaylaştırdı.

Uygulamada ana veritabanı yönetim sistemi olarak PostgreSQL kullanıldı. Relational veritabanlarına aşina olmam ve Postgres ile daha önce birçok kez çalışmış olmam sebebi ile bu DBMS’i seçtim. Bir diğer neden ise prosedürleri desteklemesi. Uygulamayı planlarken bazı gereksinimleri prosedürler ile halledebileceğimi düşünmüştüm. Her ne kadar sonradan o gereksinimleri backend kısmında halletsem de Postgres işimi fazlasıyla gördü: kolaylıkla entegre oldu ve sıkıntı çıkarmadı.

Ayrıca uygulamada serbest metin araması (free text search) için Elastichsearch ve Logstash programları kullanıldı. Veritabanının bir tablosuna index oluşturarak ayağa kaldırıldı.

Son olarak uygulamayı dokümente etme ve endpoint test etmede Swagger ile Postman, geliştirme aşamasında ise IntelliJ IDEA IDE’si kullanıldı.

Figür 1: Fiziksel Deployment Diyagramı

**API Kataloğu ve Model Şemaları**

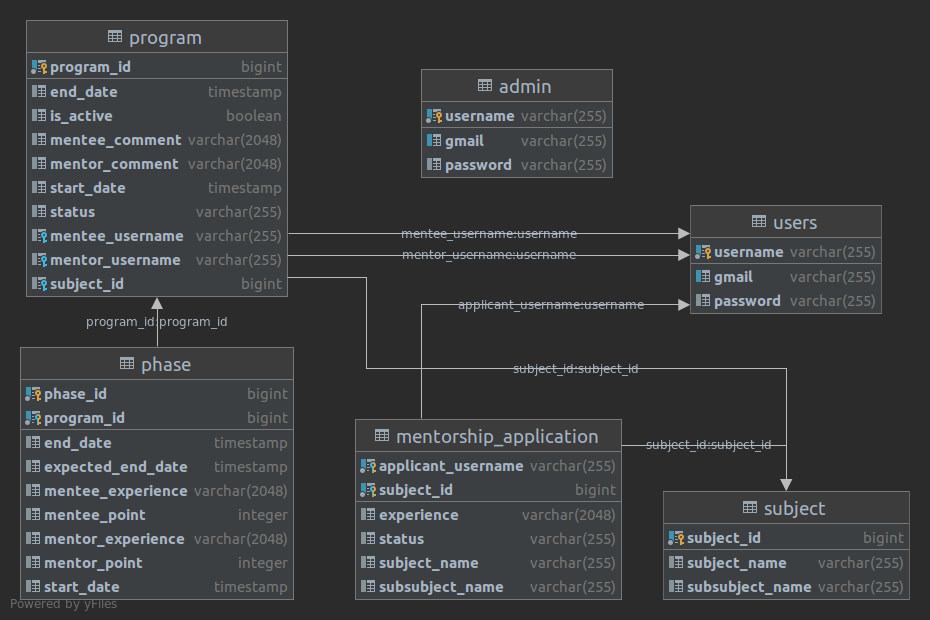
API kataloğu ve Model Şemalarına <http://localhost:8080/swagger-ui.html> sayfasından erişilebilir. Katalog [springfox kütüphanesi](https://github.com/springfox/springfox) kullanılarak otomatik oluşturuldu.

Her bir veritabanı tablosu için backendde bir Entity oluşturuldu. Ayrıca her entity için Data Transfer Object denilen modeller oluşturuldu. Data encapsulation ilkesi kapsamında, entityler modellere maplenerek kullanıcılara hassas bilgiler gösterilmedi.

Spring frameworkü, uygulamanın her ayağa kalkışında bu Entity classlarını baz alarak Obsstaj isimli PostgreSQL veritabanında tabloları oluşturdu/güncelledi.   
Ayrıca bu Entity classlarının fieldlarının üzerinde yazan @ManyToOne, @OneToOne gibi annotationları baz alarak foreign-key kısıtlamarını da veritabanına kodladı.

**Veritabanı**

Obsstaj adlı veritabanı oluşturularak admin, mentorship\_application, phase, program, subject, users isimli tablolar oluşturuldu ve bu tabloların ilişkileri kodlandı.

Figür *2*: Veritabanı tabloları ve ilişkileri

Elastichsearch arama motoru, Obsstaj isimli veritabanının “mentorship\_application” isimli tablosuna index oluşturarak, mentor araması yapılmasını sağlandı. Arama motoru [http://localhost:9200](http://localhost:9200/) portunda ayağa kaldırıldı. Config ayarları doküman klasöründe bulunabilir.

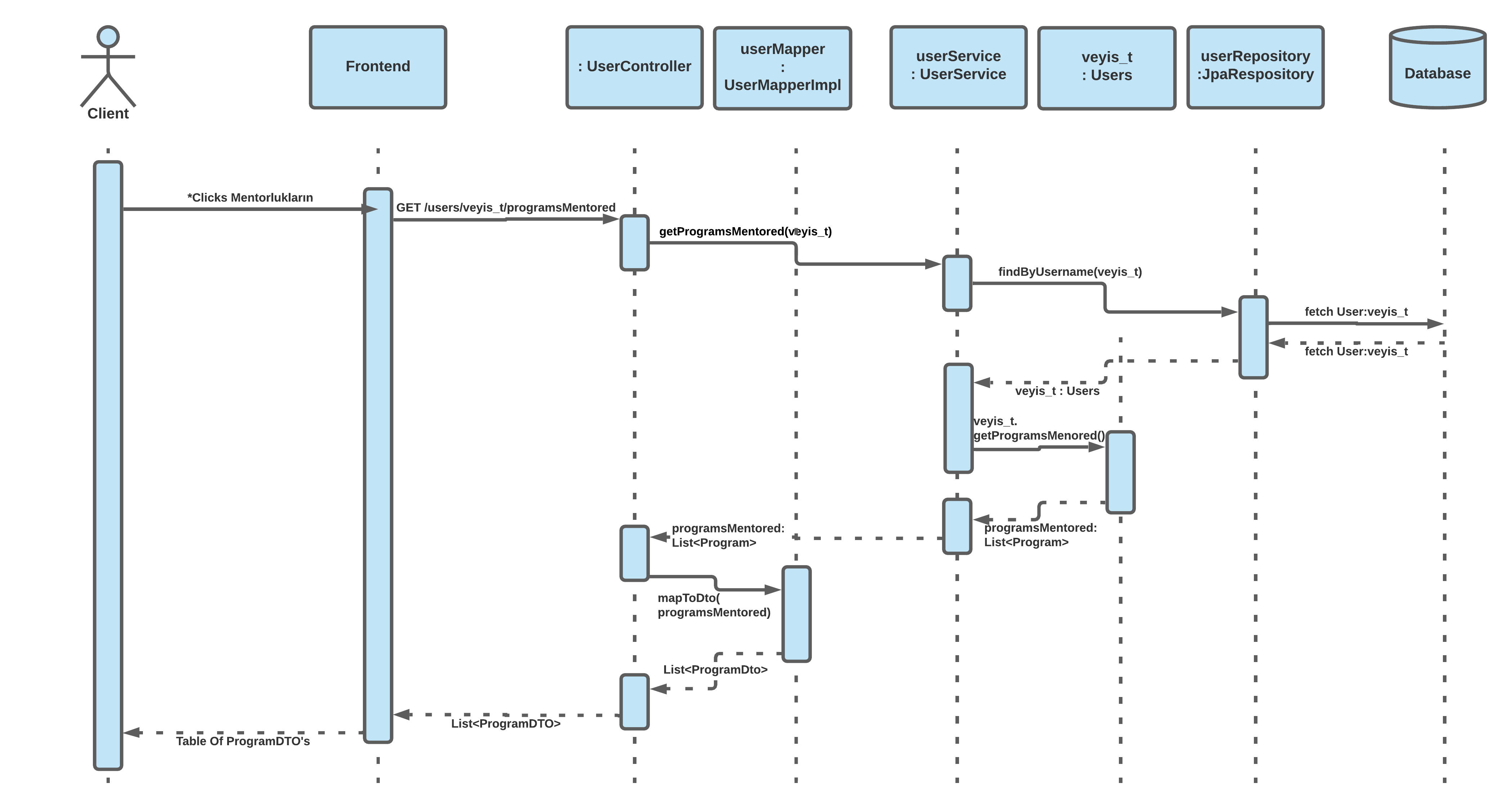
**Veri Erişim Mekanizması**

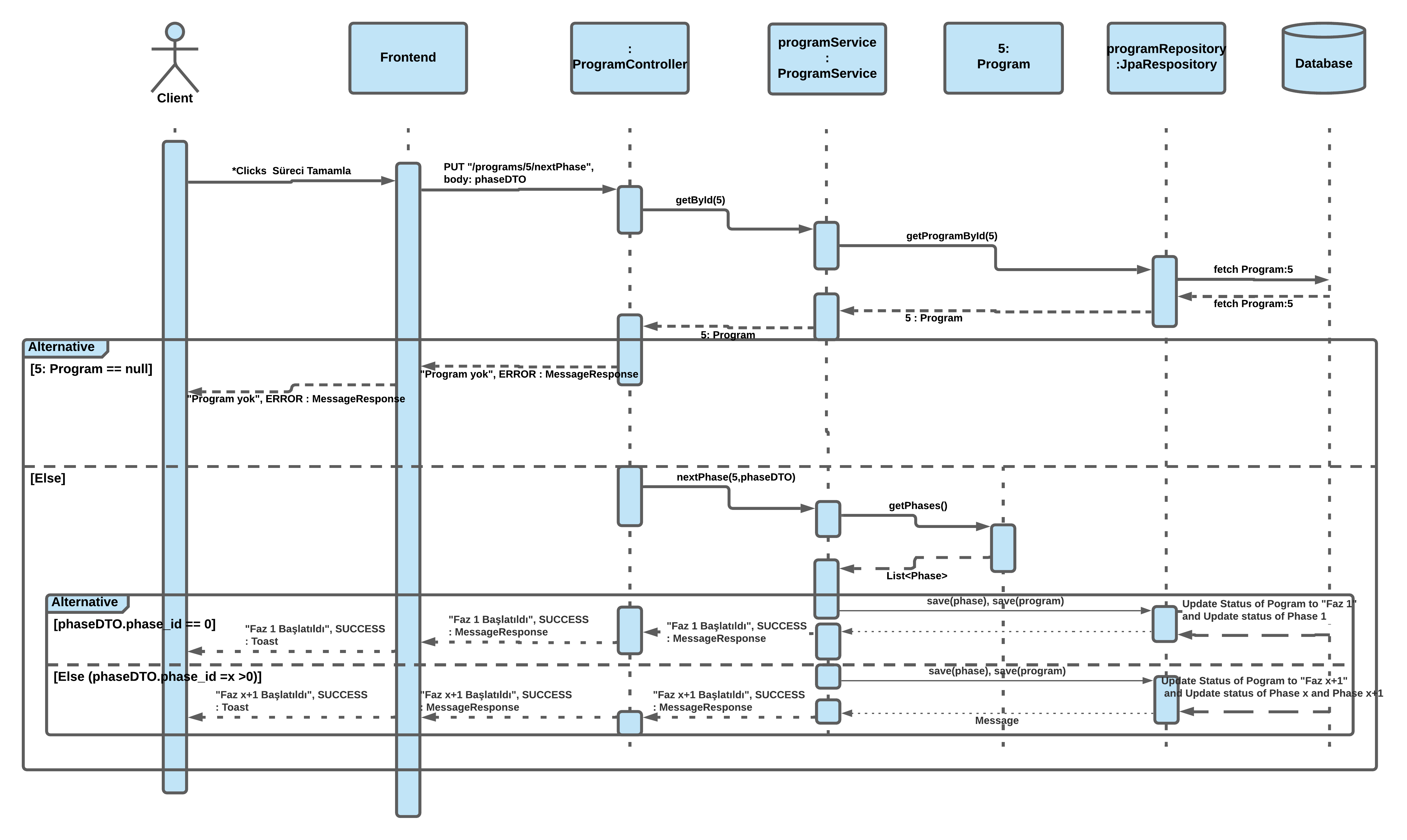
Veritabanındaki verilere erişmek için Spring Data JPA modülü kullanıldı. Her bir model için ayrı JPA Repository interface’i yazıldı. Bu interfacelerde veritabanı ile iletişim SQL queryleri Spring Data JPA’nın yardımı ve native query yazılarak sağlandı.

**Veri Akışı**

Frontendde kullanıcıdan alınan veriler, axios kütüphanesi ile backendin ilgili endpointine iletilir. Güvenlik filtresinden geçen istekler backendin “controller” classında karşılanır. Controller classları model halinde gelen veriyi entityye mapleyerek ilgili service classına gönderir. Bu nedenle controller classına köprü denebilir. Service classları asıl işin yapıldığı yerdir. Service classı hata bulmadığı sürece “repository” classları aracılığı ile gelen veriyi veritabanına kaydeder/günceller, veritabanından ilgili veriyi çeker/siler.

Örnek sekans diyagramları:

 Figür 3: Mentorlukların listelenmesi sekans diyagramı.   
 Not: resmin orjinali dokumanlar klasöründe bulunabilir.

 Figür 4: Fazı tamamla sekans diyagramı  
 Not: Resmin orijinali dokumanlar dosyasında da bulunabilir.

**Güvenlik**

**Kimlik Doğrulaması**

Uygulamaki bütün endpointler (/login/\* hariç) sisteme giriş yapmayan kullanıcılara gizlendi. Kimlik doğrulama içinse LDAP ve Google Login sistemleri kullanıldı.

LDAP kimlik doğrulaması sırasında kullanıcı verileri “ldif” dosyasında tutulur ve yeni kullanıcı kaydedilmez. Bu kullanıcılar ayrıca PostgreSQL veritabanında da tutulur.

Google ile loginde ise kullanıcılar ilk defa giriş yapıyolar ise veritabanına kaydedilirler. Gmail adreslerinin “@gmail.com” kısmı atılıp şifreleri yapılır. Eğer kullanıcı daha önce giriş yapmış ise sistem onu hatırlar. Google Login ayarları backendde değil frontendde yapılmıştır. [react-google-login](https://www.npmjs.com/package/react-google-login) kütüphanesi ile kulanıcılar frontendde doğrulanmış ve verileri backende aktarılmıştır. Bu işlemin backendde yapılmamasının nedeni üstüste alınan hatalar ve zamanın darlığıdır.

Kimlik doğrulaması ayarları config/security paketinde yapılmıştır.

**Yetkilendirme ve Oturum Yönetimi**

Yetkilendirme sistemi olarak BasicAuth kullanılmıştır. Giriş yapan kullanıcılara BasicAuth tokenı verilir, ve her requestte bu tokenı headera koymaları beklenir. Bu sistemde oturum kavramı yoktur. Frontend kullanan kullanıcılar için login sonrası token Cookie’de tutulur ve çıkış yapılması sonucu silinir. Ayrıca her endpointte yetki sınırlandırması yapılmıştır. Adminlerin ve kullanıcıların erişebileceği endpointler sınırlıdır.

**Gereksinimler ve Karşılanma Durumları**

\* **1.** Uygulamayı 2 tip kullanıcı kullanabilir: Admin ve Kullanıcı

\*\*\* Users ve Admin Entity classları bu gereksinimi karşılar. Adminler ve kullancılar giriş yaparak yetki sınırları içerisinde uygulama ile etkileşebilirler.

\* **2.1.** Kullanıcı ve adminler sisteme LDAP Authentication ile login olmalıdır.

\*\*\* Program başlatılmadan önce ldif dosyasında tanımlanan kullanıcı ve adminler username ve password bilgilerini sağlayarak sisteme login olabilirler.

\* **2.2.** Sistemde kullanıcıların LDAP haricinde Google hesabı ile login olmaları da sağlanmalıdır.

\*\*\* Google ile giriş yapanların ise databasede recordları oluşturulur ve oradan authenticate edilir. Böylece Google ile giriş yapan birisi tekrar girdiğinde hatırlanabilir.

Google Authentication kısmı backend yerine frontendde yapılmıştır. Backendde yapmaya çalışırken sürekli hata almam ve zamanın darlığı beni bu işlemi frontendde yapmaya itti. Sonuç olarak gereksinim karşılanmış olup, kullanıcı ve adminler rahatlıkla gmailleri ile sisteme giriş yapabilirler.

\* **3.1.** Admin’ler login olduğunda dashboard sayfasında mentorluk başvuruları listelenmelidir. Admin herhangi bir başvuruya tıklayıp detayı görebilir ve mentorluk başvurusunu onaylayabilir, reddedebilir.

\*\*\* Adminler login olduklarında karşılarına 2 sekmeli bir sayfa çıkar. İlk sekmede mentorluk başvurularını görebilirler. Adminler burada tablo halinde başvuruları görür. Her bir başvuru için başvuran ismi, konu ve tecrübeler kısmı görünür. Adminler başvuruyu onaylayıp reddedebilir.

\* **3.2.** Kullanıcılar login olduğunda dashboard sayfasında mentor ve mentee oldukları süreçler listelenecektir. Mentor listesinde kime hangi konuda mentorluk yaptıkları, sürecin durumu belirtilecek ve tıklandığında mentorluk detay ekranı açılacaktır. Mentee listesinde de kimden mentorluk alındığı, mentorluk konusu ve durumu belirtilecek ve tıklandığında mentorluk detay ekranı açılacaktır.

\*\*\* Kullanıcılar login olduklarında karşılarına 4 sekmeli bir sayfa çıkar: Mentorlukların, Menteeliklerin, Mentorluğa başvur, Menteeliğe Başvur.

Mentorlukların sayfasında da 2 sekme vardır: aktif mentorluklar ve bitmiş mentorluklar. Bu sekmelerde tablo halinde mentorlukları görünür. Her bir mentorluk için mentee adı, konu, durum(başlamadı, faz x, bitti) ve tarih bilgileri görünür. Tıklandığında programın detay ekranı açılır.   
Aynı tablo menteeliklerin sekmesi için de geçerlidir.

\* **4.** Mentorluk başvurusu yapmak isteyenler için bir link sağlanmalıdır. İlgili linke tıklandığında bir başvuru formu açılmalıdır. Bu formda kullanıcılar mentorluk yapmak istedikleri konuyu seçmelidirler. İlgili konu seçildikten sonra o konuya ait alt konular listelenmelidir. Kullanıcı ilgili alt konulardan bir veya daha fazlasını seçebilmelidir. İlgili seçim sonrası kullanıcı, yapacağı mentorluk ile ilgili tecrübelerinden bahsedebileceği bir açıklama alanını doldurmalıdır. İlgili alanları doldurduktan sonra “Başvur” butonu ile başvurusunu tamamlamalıdır.

\*\*\* Kullanıcılar login olduklarında karşılarına 4 sekmeli bir sayfa çıkar: Mentorlukların, Menteeliklerin, Mentorluğa başvur, Menteeliğe Başvur. Mentorluğa başvur sayfasında da 2 sekme vardır: geçmiş başvurular ve yeni başvuru. “Yeni başvuru” sekmesinde kullanıcı tüm konuları görebilir. Bunlardan kendine uygun olanı seçip tecrübelerini yazarak başvurusunu admin onayına gönderebilir. Admin başvuruyu onayladıktan sonra menteeler de bu başvuruyu görebilir ve programa kaydolabilir. Ayrıca kullanıcılar “geçmiş başvurular” sekmesinden henüz onaylanmamaış başvurularını silebilirler.

Önce konu seçimi daha sonra alt konu seçimi yerine tek tek konu-alt konu ikilisi seçimi tercih edilmiştir. Bunun nedeni kullanıcının her bir alt konu için özel ve ayrı bir biçimde tecrübelerini ve motivasyonunu dile getirmesini sağlamaktır.

\* **5.1.** Mentee’ler istedikleri konu ile ilgili mentorluk yapan kişileri listeleyebilir ve bir kişiyi mentor olarak seçebilir.

\*\*\* Kullanıcılar login olduklarında karşılarına 4 sekmeli bir sayfa çıkar: Mentorlukların, Menteeliklerin, Mentorluğa başvur, Menteeliğe Başvur. Menteeliğe başvur sayfasında tüm mentee bekleyen programlar tablo halinde görünür. Burada onaylanan mentorluk başvuruları ve henüz dolmayan, yani 1 menteesi olan programlar, listelenir. Kullanıcılar burada mentor adı, konu ve tecrübeleri görüp başvurabilir.

\* **5.2.** Bir “Mentor Arama” linki olmalıdır. Kullanıcı ilgili linke tıkladığında bir sayfa açılmalıdır.

\*\*\* Sayfadaki tablonun üstünde mentor arama butonu vardır. Butona basıldığında bir diyalog açılır.

\* **5.3.** Kullanıcı açılan sayfada bir konu ve o konuya ait alt konuları seçerek bir arama yapabilir.

\*\*\* Açılan diyalogda önerilen tüm konular listelenir ve istenilenler seçilebilir.

\* **5.4.** Herhangi bir konu seçmeden de bir free text araması yapabilir. Bu durumda mentorlerin başvuru sırasında yazdıkları açıklama alanı dikkate alınır.

\*\*\* Açılan diyalogda ayrıca metin girdisi kısmı vardır. Kullanıcılar buraya metin yazarak tecrübe kısmında arama yapabilir.

\*\*\* Ek olarak kullanıcılar 5.3 ve 5.4 te sağlanan fonksiyonların kombinasyonunu yani hem konu seçip hem serbest metina araması yapabilir.

\* **5.5.** Arama sonucu listelenir ve listedeki bilgilerin yanında bulunan “Mentor Seç” butonu ile bir mentor seçimi yapılır. Bir mentor, birden fazla konuda mentorluk yapıyorsa, hangi konu ile ilgili mentorluk alınacağı da belirtilmelidir.

\*\*\* Arama sonucunda mentor yerine direkt başvurulabilecek programlar listelenir. Bunun nedeni Elasticsearch üzerinde birden fazla tabloyu joinleyememedir. Ayrıca halihazırda başvurulabilinecek programları listelenerek kullanıcı ekstra işten kurtarılmıştır.

\* **5.6.** Bir kullanıcı aynı ana konu altında sadece bir mentor ile çalışabilir.

\*\*\* Bir mentee aynı anda bir ana konu için bir programa kaydolduğunda o ana konuda başka mentor ile çalışamaz. Bu programlar mentora listelenmez. Serbest metin araması sonucu listelense daha sistem hata verir ve menteeyi kabul etmez.

\* **5.7.** Bir mentor aynı anda bir konu için sadece 2 mentee ile çalışabilir.

\*\*\* Bir mentor bir konu-altkonu ikilisinde 2 mentee ile çalışmaya başladığı anda başvurusu görünmez olur. Başka menteeler görüntüleyemez, sayfanın yenilenmediği durumlarda halen görünür ise de başvurusu onaylanmaz.

\* **6.1.** Bu sayfada mentor, mentee bilgileri, başlangıç tarihi ve durum bilgisi yer alacaktır. Durum bilgisinde; henüz bir faz oluşturulmamışsa başlamadı, fazlar oluşturulduysa hangi fazda olduğu, bütün fazlar tamamlandıysa tamamlandı bilgisi yer almalıdır.

\*\*\* Sayfanın en üstünde programın genel bilgileri: mentor ismi, mentee ismi, başlangıç tarihi (başladıysa), bitiş tarihi(varsa) ve durum bilgisi yer almaktadır. Durum bilgisinde; henüz bir faz oluşturulmamışsa “Başlamadı”, fazlar oluşturuldu ve başlandıysa hangi fazda olduğu (“Faz x”), bütün fazlar tamamlandıysa “Bitti” bilgisi yer almaktadır.

\* **6.2.** Sayfada 6.1deki bilgiler dışında kaydedilen fazlar listelenmeli, fazların durumları ve eğer tamamlanmışsa mentor ve mentee’lerin değerlendirmeleri yer almalıdır.

\*\*\* Sayfanın en üstünde yer alan program genel bilgilerinin hemen altında “fazlar” kısmı yer almaktadır. Tamamlanan fazlar için mentor ve menteenin puan ve değerlendirmeleri görünür.

\* **6.3.** Eğer henüz faz oluşturulmamışsa “Süreç Planla” butonu olmalı ve Süreç Planlama sayfasına geçilmelidir. Fazlar oluşturulduktan sonra yine bu ekranda “Süreci Başlat” butonu olmalıdır. İlgili butona basıldığında ilk faz aktif hale gelmelidir.

\*\*\* Henüz hiç faz yok ise “Süreç Planla” butonu vardır. Bu butona basıldığında kullanıcıdan bir sayı seçmesi istenir. 1-10 arasından bir sayı seçilmesi sonucu bu sayı kadar faz eklenir. Sonrasında en aşağıda “Süreci Başlat” butonu vardır. Bu butona basıldığında ilk faz aktif olur.

\* **6.4.** Aktif olan fazın yanında “Fazı Tamamla” butonu yer almalıdır. İlgili butona tıklandığında değerlendirme ekranı açılmalı ve faz ile ilgili hem text hem de 1-5 arası puan verilebilecek alanlar bulunmalıdır.

Kullanıcı ilgili alanı doldurduktan sonra “Tamamla” butonuna bastığında ilgili faz kapanıp bir sonraki faza geçilmelidir. Eğer bu faz son faz ise mentorluk süreci tamamlanmalıdır.

\*\*\* Aktif fazın yanında “Fazı Tamamla” butonu vardır. Bu butona tıklandığında değerlendirme diyaloğu açılır. Faz ile ilgili 1-5 arası puan ve metin değerlenirme boşluğu vardır. Kullanıcı ilgili alanları doldurduktan sonra “gönder” butonuna bastığında faz tamamlanıp sonraki faza geçilir. Eğer bitirilen faz son faz ise programın durumu “Bitti” olur ve program değerlendirme butonu altif hale gelir.

Bu butona basılması sonucu genel program hakkında değerlendirmeler alınır ve en yukarda genel program bilgileri sekmesinde gösterilir.

\* **6.5.** Faz kapandıktan sonra fazı kapatmayan kişi için de fazın yanında “Fazı Değerlendir” butonu olmalı ve faz tamamlanırken açılan ekran açılıp ilgili kişi de değerlendirmesini tamamlayabilmelidir.

\*\*\* Kapanan bir faz için değerlendirmeyen kişinin sayfasında “Fazı Değerlendir” butonu aktif olur. Faz tamamlanırken açılan ekran açılır ve ilgili kişi de değerlendirmesini tamamlayabilir.

\* **7.1.** Süreç planlama sayfasında kullanıcı mentorluk sürecine fazlar ekleyebilecektir. Bunun için ilgili ekranda “Faz Adı” ve “Bitiş Zamanı” alanları olmalı ve bunları tamamladıktan sonra “Ekle” butonu ile altta bulunan listeye eklemeli ve faz tanımlama işlemi tamamlandığında en altta bulunan “Süreci

Planla” butonu ile oluşturulan fazlar kaydedilmeli ve Süreç Detay ekranına geri dönülmelidir.

\*\*\* 6.3. maddesinde belirtilen 1-10 arasında sayı seçilip süreç planla butonuna basılması ile fazların ilk oluşturulmasından sonra “tahmini bitiş tarihi” alanı doldurulabilir. Böylece fazların planlanması da tamamlanmış olur. Tarihler belirlenirken bir fazın önceki fazdan ileri tarihli olması ve sonraki fazlardan geri tarihli olması şartı koşulur.

Faz adına gerek duyulmamıştır. Onun yerine 1’den başlayarak ardışık numaralar verilmiştir. Kullanım kolaylığı açısından bu yönteme karar verilmiştir. Ayrıca her fazı tek tek oluşturup ekle butonuna basmaktansa adedini seçip sistemin hepsini otomatik oluşturması kullanıcı açısından daha rahat bir deneyim sunmaktadır.

\* **8.** Admin’lerin mentorluk için gerekli konu ve alt konuları tanımlayıp düzenleme yapabilecekleri ve silebilecekleri bir sayfa geliştirilmelidir.

\*\*\* Adminler login olduklarında karşılarına 2 sekmeli bir sayfa çıkar. İkinci sekmede konuları görebilirler. Tablo halinde görülen konulardan adminler istediğini silebilir. Ayrıca tablonun üstünde bulunan “Ekle” butonuna basıp açılan diyalogtan konu ve alt konuları girip konuyu sisteme kaydedebilir.

\* **9.** Sistemdeki aktif bir fazın bitişine 1 saat kala ilgili mentor ve mentee’ye mail ile hatırlatma yapılmalıdır.

\*\*\* 7.1’de istenen tahmini bitiş süresinden 1 saat önceye hem mentor hem de menteenin sistemde kayıtlı email adreslerine program ve faz bilgilerini içeren hatırlatma maili atılır. Bu esnada ThreadPoolTaskScheduler, CronTrigger ve javax.mail modülleri kullanılır.