Hasta Takip ve Yönetim Sistemi

Alperen Karacan *210201096*
Mehmet Ali Kır *210201033*

—Programlama Laboratuvarı 2 dersini son projesi olarak bize verilen bu projede bizden temel düzeyde hastane bilgi işlem sistemi yapılması istendi. Bu projede amaç veri tabanının aktif kullanımı ile programların veri iletişiminin pekiştirilmesini sağlamaktı.Bunun için de belli başlı isterler belirtilerek proje için bir rota verilmişti. Öncelikle veri tabanında buluncak hasta ve randevu bilgileri gibi temel seviye bilgilerin normalizasyon kurallarına uyularak yapılması ve sistemin tam verimlere verilere erişim sağlayabilmesi isteri var. Ardından nesneye yönelik programlamanın çeşitli kullanımların eklenmesi, https protokollerinin sağlanması ve en önemli kısım olan, veri tabanınyla programın ilişki kurduğu ve bu ilişki ile veritabanında değişiklikler yapılmasında programın aracı kılınmasını sağlayan bir program olması isteriydi. Son özelliği hastaların, randevuların, doktorların ve diğer nesne unsurlarının veri tabanındaki adreslerine ulaşılarak tam anlamıyla bir temel hastane yönetim sistemi kurulmasını amaçlar.

I. Özet

Projemize başladığımız planlama aşamasında hangi dilin bizim için daha iyi olacağına ve hangi local database service in bize en uygun olduğuna karar vermeye çalıştık. Dil kısıtı olmadığından python kullanarak daha rahat kaynak erişimine ve daha kolaylaştırılmış sistemsel yapılara sahip olduk. Bununla beraber proje geliştirme hızımızda fark yarattık ve farklı kaynaklardan daha verimli şekilde yararlanablidik.Python ın kaynak ve community sini kullandığımızı ayrıca belirtmeye bile gerek yok. Aynı zamanda python bize projede belirlenen sınırlar dahilinde özel fonksiyonlar sunarak işimizi kolaylaştırmanın yanında kesinlikle daha verimli hale getirmiş ve geliştirme sürecine ciddi katkı sağlamıştır. Ayrıca veri tabanı tarafında da Postrgre SQL e olan yatkınlığımız, daha doğrusu tecrübemiz, dahilinde en mantıklı kullanım olacağını kararlaştırdık ki bize geliştirme aşamasında o da ,özellikle arayüzde yabancılık çekmememizle, hız ve verimlilik sağladı desek yanlış bir yargı olmaz. Artık projenin geliştirme aşamasına geçelim.

II. Giriş

Planlama aşamasında öncelikle projemizin geliştirme adımlarını belirledik. Böylece belirlene deadline lar ile projemizde düzenli yol kat edebildik. Sırayla veri tabanı ve normalizasyon, ardından back end ve en son da front end ile projemizin geliştirmesini tamamladıK. Her zaman daha iyisi olabilir ancak bu projede elimizden gelen, en iyisiydi.

III. Yöntem

Öncelikle veri tabanında kodda nesne olarak oluşturduğumuz varlıkların verilerini düzenli şekilde belirten ve organize eden bir tablolama oluşturduk. Bunu da dokümanda bize verilen yönergeler dahilinde yaparak projenin doğruluğundan yaparken emin olduk. Özellikleri tablolar halinde verilen bu nesnelerimizin bir birleriyle bağlarını belirtecek olan anahtarlarını tablolarımızı normalizasyona uygun bir şekilde değiştirdikten sonra verdik ve nesnelerin birbiri arasında nesneye yönelik programlama mantığı ile iletişiminin veri tabanı kısmını tamamlamış olduk.
Bu yaptıklarımızı işleme dökerken de SQL dilinin membağına ererek bizi framework kullanmadan verilerle gerek işlemleri yapabilmemizi sağladı ve işimizi kolaylaştırdı. Böylelikle veri tabanının arka kısma bağlanma komutlarını ve arka planda simüle edilen yapının etkilerinin nasıl veri tabanına aktarılacağına karar vermiş olduk. Yani artık nesneye yönelik programlama kısmına gelebildik.

Öncelikle nesne olarak özelliklendirdiğimiz ve işlevseliğinin sınıflar arası da olduğu, verilerine belirlenmiş SQL kodlarıyla yönlenen ve bunun sonunda da işlemi gerçekleştiren hasta ekle komutumuzun resmini görüyoruz.

```python
def ekle_hastalar(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        hastalar_listesi = json.load(file)

    conn = connect_to_db()
    cur = conn.cursor()

    for hasta in hastalar_listesi:
        ad = hasta['Ad']
        soyad = hasta['Soyad']
        dogum_tarihi = hasta['DogumTarihi']
        cinsiyet = hasta['Cinsiyet']
        telefon_numarasi = hasta['TelefonNumarasi']
        adres = hasta['Adres']

        cur.execute(
            """
            INSERT INTO Hastalar (Ad, Soyad, DogumTarihi, Cinsiyet, TelefonNumarasi, Adres)
            VALUES (%s, %s, %s, %s, %s, %s)
            ON CONFLICT (TelefonNumarasi)
            DO UPDATE SET
                Ad = EXCLUDED.Ad,
                Soyad = EXCLUDED.Soyad,
                DogumTarihi = EXCLUDED.DogumTarihi,
                Cinsiyet = EXCLUDED.Cinsiyet,
                Adres = EXCLUDED.Adres
            """,
            (ad, soyad, dogum_tarihi, cinsiyet, telefon_numarasi, adres)
        )

    conn.commit()
    cur.close()
    conn.close()
    print("Hastalar eklendi veya güncellendi!")
```

Fig. 1. Hasta nesnemizinin metodu ve özellikleriyle eklenmesi

Ardından da yine belirlediğimiz nesnemmizin üzerinde işlem yapacak olan ama görece sadece okumadiyebileceğimiz işlemi ile verinin grüntülenmesini sağlıyoruz. Bu fonksiyonun aynı zamanda doktor ve randevu nesneleri içinde geçerli olup, yapılan değişiklikler ve ekleme çıkarmalarla da bağlı olduğu diğer tabloları etkilediğini söylemek mümkün.

```python
@app.route('/doktor_hastagor/<int:doktor_id>')
def doktor_hastagor(doktor_id):
    conn = connect_to_db()
    cur = conn.cursor()

    cur.execute("""
        SELECT h.HastaID, h.Ad, h.Soyad, h.DogumTarihi, h.Cinsiyet, h.TelefonNumarasi, h.Adres
        FROM Hastalar h
        JOIN Randevular r ON h.HastaID = r.HastaID
        WHERE r.DoktorID = %s
        ORDER BY h.Ad, h.Soyad
    """, (doktor_id,))

    hastalar = cur.fetchall()
    cur.close()
    conn.close()

    return render_template('doktor_hastagor.html', hastalar=hastalar)
```

Fig. 2. Hastamızın doktor ekranında görülebilmesinin sağlanması

Hasta nesnemizin verisiyle rahat bir şekilde işlem yapabildiğimizi ve aynı zamanda nesneye yönelik programlama yapısındaki basamaklı değişimlerin bağlı etkilerinin görülmesine sebep olacak o işlemlerden biri olan güncelleme işlemini yapan kodu görüyoruz. Bu kodun da diğer kodlar gibi randevu ve doktor nesneleri için geçerli olup bu nesnelerin de diğer veri tablolarını isterler ve nesneye yönelik programlama kurallarına göre etkilediğini söylemek mümkün.

```python
@app.route('/hasta_guncelle/<int:hasta_id>', methods=['GET', 'POST'])
def hasta_guncelle(hasta_id):
    conn = connect_to_db()
    cur = conn.cursor()

    if request.method == 'POST':
        telefon_numarasi = request.form['telefon_numarasi']
        adres = request.form['adres']

        cur.execute(
            """
            UPDATE Hastalar
            SET TelefonNumarasi = %s, Adres = %s
            WHERE HastaID = %s
            """,
            (telefon_numarasi, adres, hasta_id)
        )
        conn.commit()

        flash("Hasta başarıyla güncellendi!")
        return redirect(url_for('hastalar'))

    cur.execute("SELECT * FROM Hastalar WHERE HastaID = %s", (hasta_id,))
    hasta = cur.fetchone()
    cur.close()
    conn.close()

    return render_template('hasta_guncelle.html', hasta=hasta)
```

Fig. 3. Eklenmiş hasta nesnesinin spesifik özelliklerinin veir tabanında güncellenmesini sağlayan kod

Akabinde başka bir veri tabanı nesne özelliği değişimi olan hasta silme işlemini gerçekleştiren kodu görüntüleyebiliyoruz. Bir hastayı temsil eden bir nesneyi silmek, genellikle karmaşık bir işlemdir. Bunun nedeni, silinen nesnenin diğer nesnelerle ilişkili olabilmesidir. Örneğin, bir hasta nesnesi, doktor nesneleri ve randevu nesneleri ile ilişkilendirilebilir. Bir hasta nesnesini silmeden önce, bu ilişkilerin nasıl ele alınacağı belirlenmelidir. Ayrıca diğer veri tabanı işlmelerinin de bu işlemlerle karışmamasının sağlayıcısı aslında yine nesneye yönelik programlama denebilir. Çünkü yapılacak işlemlerin parçalanması ve bu parçalanmalar ile görevlendirmenin düzgün olması ve böylece daha verimli hızlı ve kullanışlı bir sistemin oluşmasını sağlar. Bu da bizim ne kadar bu projede düşünmesek de görece büyük projelerdeki hızların ve verimliliklerin boyutsal farklarında yapacağı gözardı edilemez etkiyle önemini belli edecektir.

Geldiğimiz kısımda hastane sisteminden çok

```python
@app.route('/hasta_sil/<int:hasta_id>', methods=['POST'])
def hasta_sil(hasta_id):
    conn = connect_to_db()
    cur = conn.cursor()

    cur.execute("DELETE FROM Hastalar WHERE HastaID = %s", (hasta_id,))
    conn.commit()

    flash("Hasta ve ilgili tüm randevuları başarıyla silindi!")

    cur.close()
    conn.close()

    return redirect(url_for('hastalar'))
```

Fig. 4. Daha önce eklenmiş hasta nesnesinin silinmesini sağlayan kod bloğu

genel sürekliliklerin ve belli süreçlerin topluluklarca organizasyonunu sağlayan kayıt ve giriş sistemlerine gelelim. Göreceğimiz kod, hastaların kayıtlarını yapmasını sağlıyor. Bu da yönetilebilecek temel bir kayıt sistemi sağlıyor. Kayıt olma kodu SQL kodlarıyla beraber görülebilir.

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    conn = connect_to_db()
    cur = conn.cursor()

    cur.execute("SELECT DISTINCT UzmanlikAlani FROM Doktorlar ORDER BY UzmanlikAlani")
    uzmanlik_alanlari = [row[0] for row in cur.fetchall()]

    cur.execute("SELECT DISTINCT CalistigiHastane FROM Doktorlar ORDER BY CalistigiHastane")
    hastaneler = [row[0] for row in cur.fetchall()]

    if request.method == 'POST':
        kullanici_turu = request.form['kullanici_turu']
        ad = request.form['ad']
        soyad = request.form['soyad']


        if kullanici_turu == 'Hasta':
            dogum_tarihi = request.form['dogum_tarihi']
            cinsiyet = request.form['cinsiyet']
            telefon_numarasi = request.form['telefon_numarasi']
            adres = request.form['adres']

            cur.execute(
                """
                INSERT INTO Hastalar (Ad, Soyad, DogumTarihi, Cinsiyet, TelefonNumarasi, Adres)
                VALUES (%s, %s, %s, %s, %s, %s)
                """,
                (ad, soyad, dogum_tarihi, cinsiyet, telefon_numarasi, adres)
            )
            conn.commit()
            flash("Hasta kaydı başarılı!", "success")

        elif kullanici_turu == 'Doktor':
            uzmanlik_alani = request.form['uzmanlik_alani']
            calistigi_hastane = request.form['calistigi_hastane']

            cur.execute(
                """
                INSERT INTO Doktorlar (Ad, Soyad, UzmanlikAlani, CalistigiHastane)
                VALUES (%s, %s, %s, %s)
                """,
                (ad, soyad, uzmanlik_alani, calistigi_hastane)
            )
            conn.commit()
            flash("Doktor kaydı başarılı!", "success")

        return redirect(url_for('login'))

    cur.close()
    conn.close()

    return render_template('register.html', uzmanlik_alanlari=uzmanlik_alanlari, hastaneler=hastaneler)
```

Fig. 5. Alınan bilgilerin nesneleştirilip veri tabanına kaydeden kod

Bu işlevin yanı sıra kayıt özelliği olmasının bir diğer yanı olan giriş sistemi de mevcut. Bu iki sistem de hem doktor hem hasta için geçerli olup dinamik bir erişim ile karşılıklı tarafların yeterli bilgiye yeterli yetki ve güvenlikle ulaşmasını sağlıyor. Kodu yazının devamında görülebilir.

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        ad = request.form['ad']
        soyad = request.form['soyad']
        kullanici_turu = request.form['kullanici_turu']

        conn = connect_to_db()
        cur = conn.cursor()

        if kullanici_turu == 'Hasta':
            cur.execute("SELECT * FROM Hastalar WHERE Ad = %s AND Soyad = %s", (ad, soyad))
            user = cur.fetchone()
        elif kullanici_turu == 'Doktor':
            cur.execute("SELECT * FROM Doktorlar WHERE Ad = %s AND Soyad = %s", (ad, soyad))
            user = cur.fetchone()

        cur.close()
        conn.close()

        if user:
            session['loggedin'] = True
            session['user_id'] = user[0]
            session['user_type'] = kullanici_turu
            flash("Giriş başarılı!")
            return redirect(url_for('index'))
        else:
            flash("Geçersiz kullanıcı adı veya soyadı!")

    return render_template('login.html')
```

Fig. 6. Kaydolan ve verileri nesneleştirilen kimsenin giriş yapmasını sağlayan kod

Kodumuzun devamında aslında aşama olarak da devam ederek kayıt olmuş ve giriş yapmış kullanıcıların gerekli bilgilerini nasıl görüntüleyebileceğinin örnek bir kodunu görmüş oluyoruz. Gördüğümüz kodda doktor kullanıcı kendisine randevusu olan hastaları görebiliyor. Aynısını hastaların randevularını görüntüleyebildiklerini de söyleyebiliriz. Kodu yazının devamındadır.

```python
@app.route('/doktor_hastagor/<int:doktor_id>')
def doktor_hastagor(doktor_id):
    conn = connect_to_db()
    cur = conn.cursor()

    cur.execute("""
        SELECT h.HastaID, h.Ad, h.Soyad, h.DogumTarihi, h.Cinsiyet, h.TelefonNumarasi, h.Adres
        FROM Hastalar h
        JOIN Randevular r ON h.HastaID = r.HastaID
        WHERE r.DoktorID = %s
        ORDER BY h.Ad, h.Soyad
    """, (doktor_id,))

    hastalar = cur.fetchall()
    cur.close()
    conn.close()

    return render_template('doktor_hastagor.html', hastalar=hastalar)
```

Fig. 7. Doktorun hastalarını görüntüleyebileceği kod

```python
@app.route('/hasta_randevularim')
def hasta_randevularim():
    if 'ad' in session and session['kullanici_turu'] == 'Hasta':
        ad = session['ad']
        soyad = session['soyad']

        conn = connect_to_db()
        cur = conn.cursor()
        cur.execute("""
            SELECT r.RandevuTarihi, r.RandevuSaati, d.Ad, d.Soyad
            FROM Randevular r
            JOIN Doktorlar d ON r.DoktorID = d.DoktorID
            JOIN Hastalar h ON r.HastaID = h.HastaID
            WHERE h.Ad = %s AND h.Soyad = %s
            ORDER BY r.RandevuTarihi, r.RandevuSaati
        """, (ad, soyad))
        randevular = cur.fetchall()
        cur.close()
        conn.close()

        return render_template('hasta_randevularim.html', randevular=randevular)
    else:
        flash("Bu sayfaya erişim izniniz yok!")
        return redirect(url_for('index'))
```

Fig. 8. Hastanın randevularını görüntüleybilmesini sağlayan kod

Tüm bu nesneleştirerek kaydedip ilişkilendirme ile yönetme sistemimizi ifade ediyor. Böylece elimizdeki projeyi teslim etmiş ve temel seviye bir yönetim sistemi kurmuş oluyoruz. Yapıla üyelik işlemleri görüntülemeler ve ön yüz çalışmları tamamen isterlere binaen yapılmış olup bazı rütuşlarla fark ortaya konulmuşutur.

IV. Deneysel Sonuçlar

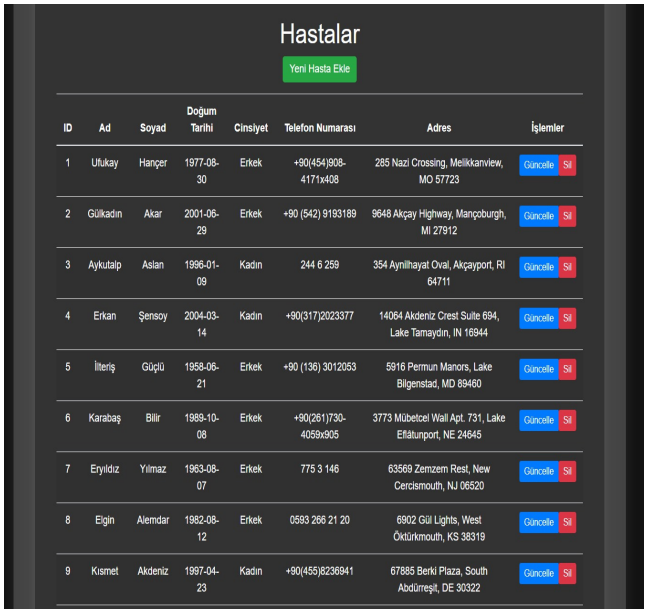

Fig. 9. Tüm hastaların görüntülendiği ekran

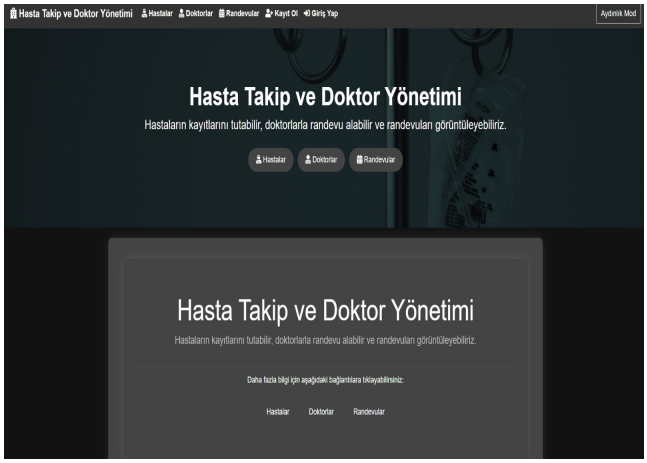

Fig. 10. Hastane Yönetim Sistemi sitemizin ana sayfası

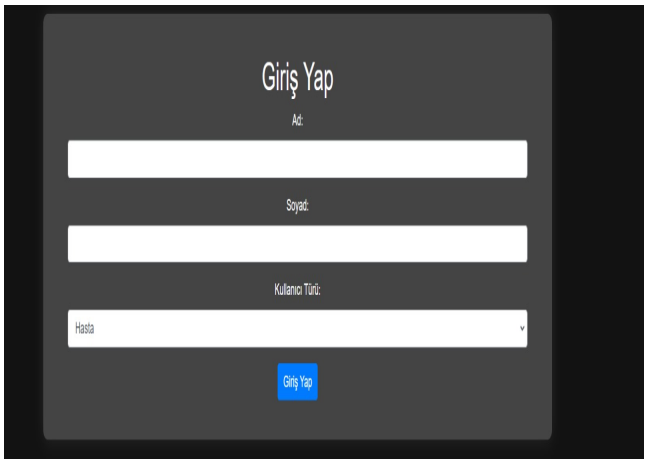

Fig. 11. Kayıtlı kullanıcıların giriş ekranı

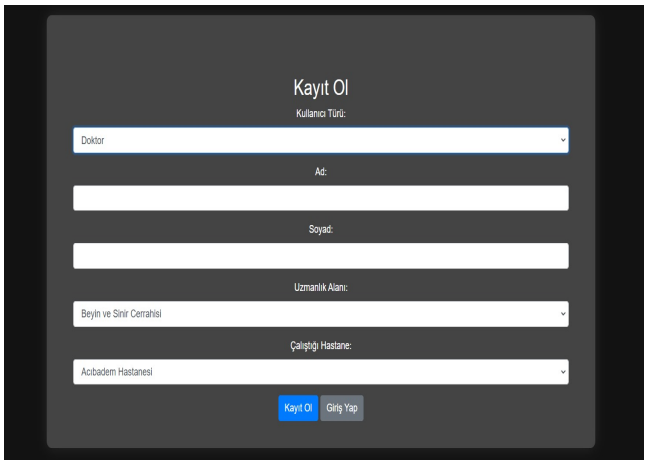

Fig. 12. Kayıt olmamış kullanıcıların kayıt olması için olan ekran

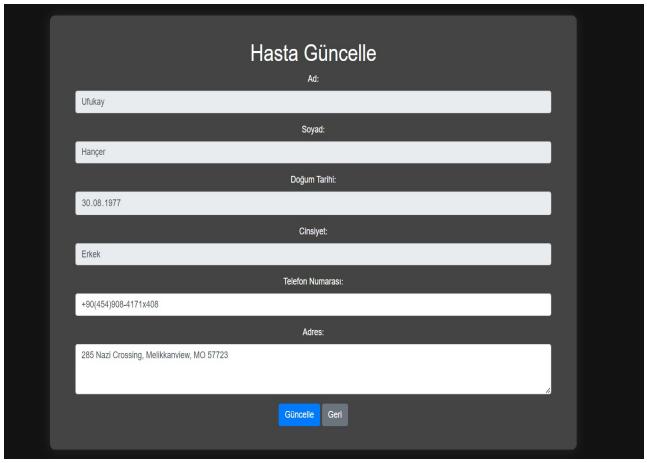

Fig. 13. Hasta güncellemesinin yapılabildiği ekran

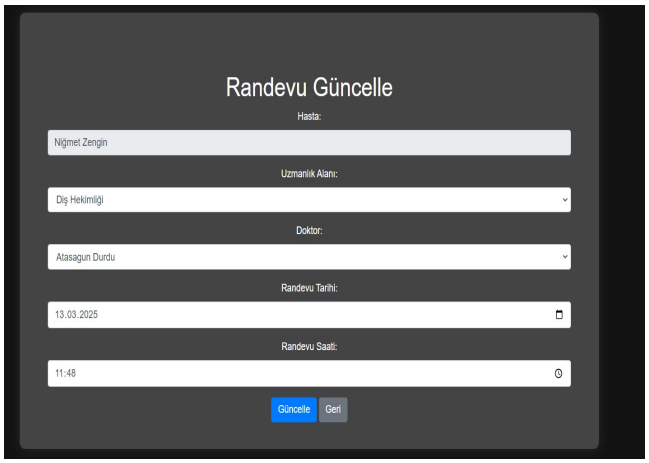

Fig. 14. Mevcut randevuların güncellenemsi için olan ekran

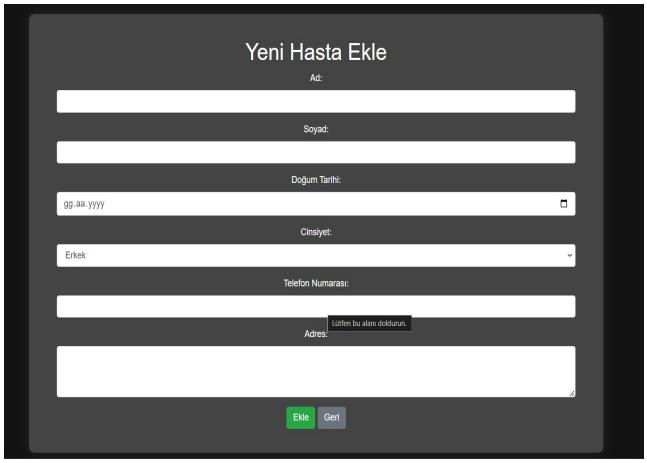

Fig. 15. Hasta eklenebilmesini sağlayan yönetici ekranı

V. Sonuç

Bu projeyle gerçek hayatta bölümümüzden en çok bulunan alanlardan biri olan bilgi işlem konusu üzerine,nesneye yönelik programlamanın kolaylaştırıcılığı ve düzenliliği üzerine ve son olarak da normalize edilmiş bir veri tabanındaki veri miktarının önemi olmadan ne kadar düzenliyse o kadar verimli ve o kadar kıymetli bir sistem olacağını öğrenmiş olduk.

VI. Deneysel Sonuçlar

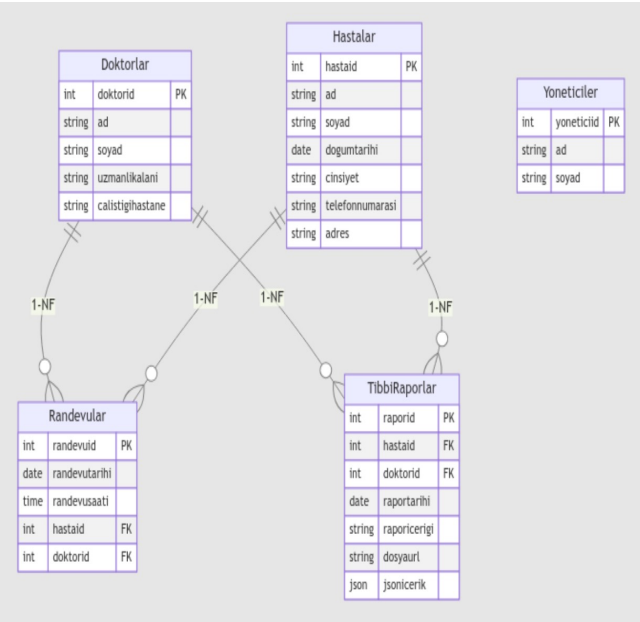

Fig. 16. Veri tabanınadki nesneleri ve aralarındaki ilişkileri ifade eden entity diagram

VII. Kaynakça

OpenAI-ChatGPT 3.5
Çeşitli Youtube videoları
Stackoverflow'da muhtelif konu başlıkları