

ANKARA ÜNİVERSİTESİ  
MÜHENDİSLİK FAKÜLTESİ  
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



Ağ Tabanlı Teknolojiler Ve Uygulamaları  
RAPORU.

**Gelirim Yönetme Paneli “Admin User”**

**Qutaiba ALASHQAR, 20290036.**

25/09/2023 – 30/12/2023

## ÖZET

Bu projede, "Gelirim Yönetme Paneli" adlı kapsamlı bir web uygulaması geliştirdim. Proje, başlangıçtan itibaren tasarım, frontend, veritabanı ve backend aşamalarını içermekte olup, kullanıcı dostu bir arayüzün yanı sıra güçlü bir işlevselliği hedeflemiştir.

### **Tasarım Aşaması:**

Proje, Figma aracılığıyla detaylı bir tasarım süreci ile başlamıştı. Bu süreç, modern ve etkileşimli bir kullanıcı deneyimi sağlamak üzere tasarlanmıştır.

### **Frontend Geliştirme:**

HTML, CSS ve JavaScript kullanılarak, tasarım frontend kısmına dönüştürülmüştür. Visual Studio IDE'nin sağladığı etkili kod yazma ortamı, geliştirme sürecini hızlandırmış ve kullanıcı arayüzünü görsel olarak çekici hale getirmiştir.

### **Veritabanı Oluşturma:**

SQL Server kullanılarak, projenin veritabanı altyapısı oluşturulmuştum. Veritabanı, projenin gereksinimlerine uygun olarak tasarlanmış ve ilişkilendirilmiş tablolar aracılığıyla veri depolama işlevselliği sağlanmıştır.

### **Backend Geliştirme:**

ASP.NET Core Razor 6.0 kullanılarak, backend tarafı güçlü bir şekilde inşa edilmiştir. Visual Studio IDE, gelişmiş kod düzenleme ve hata ayıklama özellikleri ile projenin güvenilir bir şekilde geliştirilmesine katkıda bulunmuştur. API'lar, kullanıcıların veritabanı ile etkileşimde bulunmasını kolaylaştıracak şekilde tasarlanmıştır.

Bu projenin amacı, modern teknolojilerin etkin kullanımı ve Visual Studio IDE'nin avantajlarıyla bir araya gelerek, gelir yönetimini hedefleyen başarılı bir web uygulamasını ortaya koymaktır. Detaylı rapor, projenin her aşamasındaki kararları ve uygulamaları detaylandırarak, geliştirilen çözümün etkili bir şekilde nasıl çalıştığını gösterecektir.

## İÇERDEKİLER

ÖZET.....	1
İÇERDEKİLER.....	3
1.1. GİRİŞ.....	4
1.2. Program Yapısı.....	7
1.3. Bilgilerin Güncellemsi.....	9
1.4. Sonuç.....	10
2.1. Database.....	10
3.1. Back-End.....	12
3.2. Veritabanı Etkileşimle.....	14
3.3. Güvenlik Zorlukları ve Çözüm Önerileri.....	16
4.1. Genel Sonuç.....	16
Kaynaklar.....	19

<https://github.com/QutaibaAlashqar/Incom-Management-Panel>

## 1. GİRİŞ

Gelirim Yönetme Paneli projesi, modern teknolojilerle entegre edilen, gelir yönetimi odaklı bir web uygulamasını hayata geçirmeyi amaçlamaktadır. Bu projenin geliştirilmesinde özellikle öne çıkan teknoloji, ASP.NET Core Razor 6.0'dır. Bu bölümde, ASP.NET Core Razor 6.0'nın projedeki rolünü, özelliklerini ve sağladığı avantajları detaylı bir şekilde inceleyeceğiz.

### Tasarımın Esintisi

Projenin temeli, Figma tasarım süreciyle atılmıştır. Kullanıcı deneyimini en üst düzeye çıkarmak ve modern bir görünüm elde etmek amacıyla detaylı tasarımlar gerçekleştirilmiştir. Figma, projenin görsel kimliğini oluşturmak için başlangıç noktası olmuş, ve bu tasarımların hemen hemen aynısı, bazı küçük ayarlamalarla, projenin final haline taşınmıştır.

### ASP.NET Core Razor 6.0 Nedir

ASP.NET Core Razor 6.0, modern web uygulamaları oluşturmak için kullanılan bir framework'tir. Bu framework, özellikle C# dilini kullanan geliştiricilere, dinamik web sayfaları ve uygulama içi C# kodu kullanma yeteneği sağlar. Razor, HTML içinde C# kodu yazmanın yanı sıra, veritabanı bağlantıları ve diğer server-side işlemleri kolaylaştırarak geliştirme sürecini optimize eder.

### Razor Sayfaları ve Özellikleri

ASP.NET Core Razor 6.0, Razor sayfaları aracılığıyla sayfa bazlı uygulama geliştirmeyi mümkün kılar. Razor sayfaları, HTML içinde C# kodu yazma kabiliyeti ile dinamik ve etkileşimli web sayfaları oluşturmayı sağlar. Bu sayfalar, frontend ve backend kodunun bütünleşik bir şekilde çalışmasını sağlayarak, geliştirme sürecini daha modüler hale getirir.

### Önemli Değişiklikler ve Figma Görselleri

Tasarladığınız Figma görüntülerinin projeye büyük bir uyum içinde olduğunu fark etmek, projenin tutarlılığını ve estetiğini korumanın bir başarısıdır. Projenizin Figma tasarımını geliştirilmiş bir versiyon olarak görmek, kullanıcılara tasarım vizyonunuzun doğru bir şekilde yansıtıldığını gösterir. Bu bağlamda, projenin detaylı inceleme aşamasında, Figma tasarımı ile proje ekran görüntülerini karşılaştırmak için kullanılacak olan görselleri eklemek, okuyuculara projenin evrimini gösterirken görsel bir zenginlik katacaktır.

## **Güçlü Backend Altyapısı**

ASP.NET Core Razor 6.0, sadece güçlü bir frontend geliştirme aracı değil, aynı zamanda güçlü bir backend altyapısı sunar. MVC (Model-View-Controller) mimarisi ile uyumlu çalışarak, web uygulamasının iş mantığını etkili bir şekilde yönetir. Bu, veritabanı işlemleri, kullanıcı girişi kontrolü ve diğer server-side işlemlerin kolayca yönetilmesini sağlar.

## **Tasarımın Gerçeğe Dönüşümü**

Figma tasarımından ilham alarak başlanan frontend geliştirme süreci, HTML, CSS ve JavaScript ile şekillendirilmiştir. Kullanıcı arayüzü, Figma'daki tasarıma sadık kalacak şekilde oluşturulmuş, ve projenin amacına uygun olarak işlevsel özellikler eklenmiştir.

## **Visual Studio IDE ve ASP.NET Core Entegrasyonu**

ASP.NET Core Razor 6.0'nın en güçlü yanlarından biri, Visual Studio IDE ile sorunsuz bir entegrasyona sahip olmasıdır. Visual Studio, geliştiricilere kod düzenleme, hata ayıklama ve otomatik tamamlama gibi zengin özellikler sunarak, projenin daha hızlı ve hatasız bir şekilde geliştirilmesine olanak tanır.

## **C# ve ASP.NET Core Entegrasyonu**

ASP.NET Core Razor 6.0 projemizde C# dilinin kullanılması, web uygulamasının backend tarafının geliştirilmesinde büyük bir rol oynamaktadır. ASP.NET Core, C# dilini temel olarak geliştirilmiş bir framework olduğundan, bu ikili entegrasyon sayesinde web uygulamamızın hem güçlü hem de esnek bir backend altyapısına sahip olması sağlanmıştır.

## **Nesne Yönelimli Programlama (OOP) Yetenekleri**

C#, nesne yönelimli programlama (OOP) paradigmasına sıkı sıkıya bağlıdır. Bu, projede karmaşık veri yapıları ve işlemlerini daha düzenli ve modüler bir şekilde yönetmeyi sağlar. Sınıflar, nesneler ve kalıtım gibi OOP kavramları, projenin daha sürdürülebilir ve genişletilebilir olmasına katkı sağlar.

## **Güçlü Tip Güvenliği**

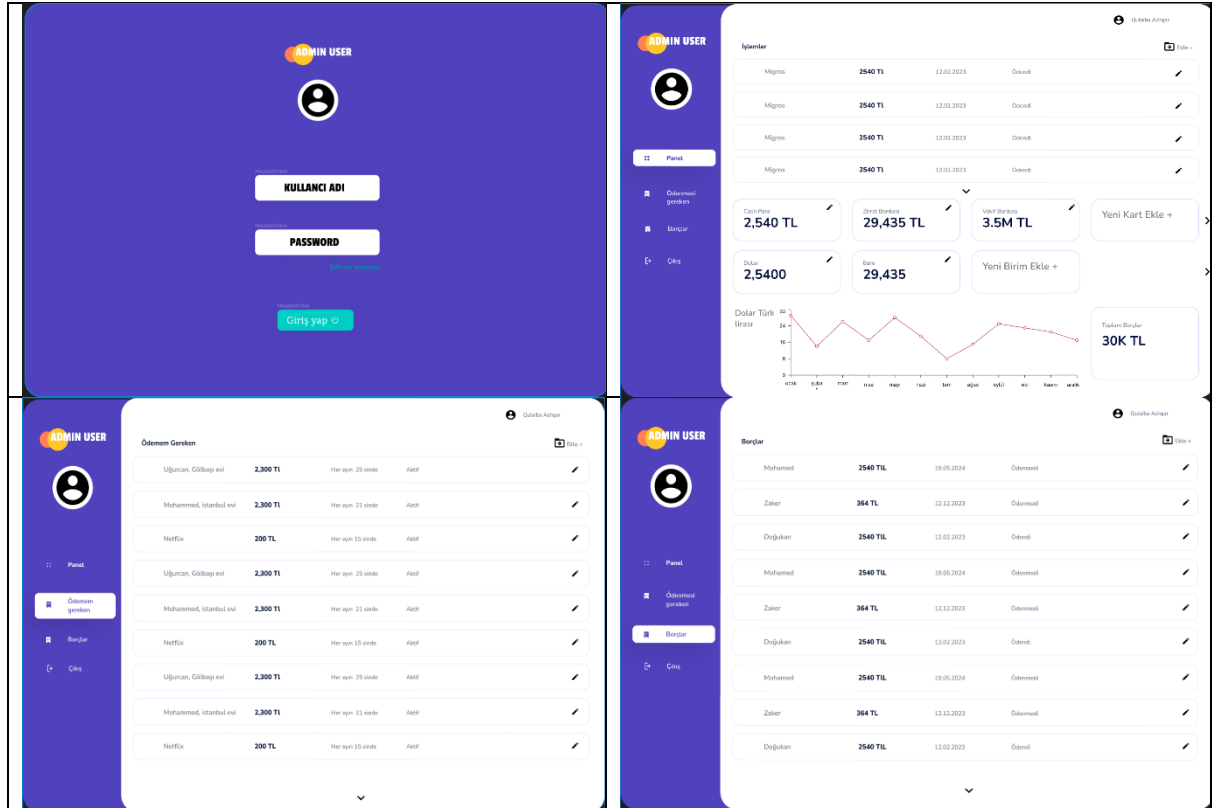
C#, güçlü bir tip güvenliği ile öne çıkar. Bu, hataların daha erken tespit edilmesine ve daha güvenli bir kod yazma sürecine olanak tanır. Bu özellik, projenin kararlılığını ve performansını artırır.

## Visual Studio IDE ve C# Uyumlu Çalışma

Visual Studio IDE, C# dilini kullanarak projenin geliştirilmesini kolaylaştıran birçok özelliği içerir. Kod tamamlama, hata ayıklama ve performans analizi gibi araçlar, geliştiricilere daha etkili bir kod yazma deneyimi sunar.

Gelirim Yönetme Paneli projesinde C# dilinin kullanımı, projenin başarılı bir şekilde geliştirilmesine ve sürdürülmesine olanak tanımaktadır. Bu dilin sunduğu özellikler, projenin karmaşıklığına uyum sağlayarak, gelir yönetimi uygulamasının beklenen performansı ve işlevselliği elde etmesini sağlamıştır. Detaylı inceleme, C# dilinin projedeki önemini ve avantajlarını daha ayrıntılı bir şekilde açığa çıkaracaktır. Aynı zamanda ASP.NET Core Razor 6.0'nın sunduğu avantajları kullanarak, gelir yönetimi odaklı bir web uygulamasını başarılı bir şekilde hayata geçirmiştir. Detaylı bir inceleme, bu teknolojinin projede nasıl kullanıldığını ve projenin başarısına nasıl katkı sağladığını açığa çıkaracaktır.

## Tasarladığım Figma'yı



## **1.2. Program Yapısı**

Giriş sayfası ilk izlenim için hayati önem taşır ve tasarımın kullanıcıyı karşılayan modern ve şık bir görünüme sahip. Sayfanın düzeni, kullanıcıların dikkatini doğrudan giriş formuna çekiyor ve renklerin canlılığı marka kimliğinizi güçlendiriyor.

### **Renk Seçimi Zorlukları**

Renk paleti seçimi, projenin başlangıcında bir meydan okuma olarak ortaya çıkmış. Kullanıcıların duygusal tepkilerini ve marka algısını etkileyecek renklerin seçimi, dikkatli bir biçimde ele alınmış. Uyumlu bir renk skalası belirleme süreci, projenin görsel kimliğini belirginleştirmekte önemli bir rol oynamış.

### **JavaScript ile Scroller Yapısı**

Sayfa içi gezinme için scroller yapısının entegrasyonu, kullanıcıların sayfada rahat hareket edebilmelerini sağlayan bir özellik. JavaScript kullanarak geliştirilen bu dinamik öğe, kullanıcı etkileşiminde akıcılığı artırmıştı.

### **Ana Sayfa ve Navigasyon**

#### **- Side Navbar ve Kullanıcı Deneyimi**

Ana sayfanın Side navbar'ı, etkili bir kullanıcı deneyimi için kritik bir unsur. Tüm ana işlemlere hızlı erişim sağlamak için tasarlanmıştı bu bölüm, sitenin kullanılabilirliğini büyük ölçüde artırmış.

### **Yeni Not Ekleme İşlevselliği**

Kullanıcının etkileşimde bulunması beklenen alanlardan biri de not ekleme özelliği. Bu özellik, veritabanı ve frontend arasındaki entegrasyonun başarılı bir şekilde gerçekleştirilmesini gerektirdi ve kullanıcıların içerik yönetimini kolaylaştırdı.

### **Finansal İşlemler Sayfası**

#### **- Tablolar ve Veri Sunumu**

Finansal işlemlerin listelendiği bu sayfada, verilerin düzenli ve anlaşılır bir formatla sunulması sağlanmış. CRUD işlemleri (oluşturma, okuma, güncelleme, silme) için gerekli butonlar, kullanıcıların veri yönetimini kolaylaştırmak üzere yerleştirilmiştir.

### **Grafikler ve Analitik Gösterim**

Ekonomik verilerin analizi ve gösterimi için kullanılan grafikler, kullanıcıların bilgiyi daha hızlı işlemesine yardımcı olmak için tasarlanmıştır. Bu görsel araçlar, kompleks veri setlerini anlamlandırmada kullanıcıları desteklemiştir.

### **Kullanıcı Profili ve Ayarlar Sayfası**

#### **- Kullanıcı Bilgileri Yönetimi**

Bu sayfa üzerinde, kullanıcılar kişisel bilgilerini ve sistem tercihlerini yönetebilmekte. Kullanıcı deneyimi açısından bu bölümün kolay erişilebilir ve kullanışlı olması gerektiğinden, tasarımı özenle yapılmıştır.

### **Footer Tasarımı ve Responsive Zorluklar**

Her sayfanın altında yer alan footer, web uygulamanızın profesyonel görünümünü tamamlayan bir öğe. Responsive tasarım zorlukları, farklı cihazlar ve ekran boyutlarına uyum sağlama konusunda sizin becerilerinizi geliştirmiştir. Footer'ın uygun şekilde konumlandırılması, tasarımın tamamlayıcı bir parçası olarak işlev görmüştür.

### **Dikkatli ve Performans Odaklı Tasarım**

Proje boyunca dikkatli bir planlama ve tasarım süreci izlenmeye çalıştım. Her bir öğe, hem estetik hem de performans açısından ince eleyp sık dokunarak seçmeye çalıştım. Özellikle, kullanıcı etkileşimi gerektiren öğelerde, performansın ve kullanıcı deneyiminin optimize edilmesine büyük önem verdim.

### **Kur Değişikliklerinin Yönetimi**

Ekran görüntülerinden birinde, bir döviz kuru grafiği görünüyor. Bu, uygulamanın kullanıcıların TL ve USD arasındaki döviz kuru değişimlerini takip etmelerine ve finansal planlamalarını buna göre yapmalarına olanak tanıdığını gösteriyor. Döviz kurlarındaki dalgalanmalar, özellikle ithalat ve ihracat yapan işletmeler için önem taşıyor ve bu yüzden, bu tür bir uygulama işletmelerin finansal risklerini yönetmelerine yardımcı olabilir.

### **Finansal İşlemlerin Takibi**



Uygulamanın, kullanıcılara TL ve Dolar cinsinden ödeme kayıtlarını tutma, borçları izleme ve bu verileri düzenleme imkanı verdiği anlaşıyor. Örneğin, "Ödenecek Para" bölümünde, farklı tarihlerde gerçekleştirilen ödemelerin TL ve Dolar cinsinden listelendiği görülüyor. Bu özellik, finansal durumun net bir şekilde görülebilmesi ve yönetilebilmesi için kritik önem taşır.

### **Çoklu Para Birimi ile Çalışma Esnekliği**

Çoklu para birimi desteği sunan bu tür bir panel, uluslararası iş yapma kapasitesine sahip firmalar için özellikle faydalı olabilir. Kullanıcıların farklı para birimlerindeki işlemleri kolayca yönetebilmeleri, iş akışlarını ve muhasebe süreçlerini basitleştirir.

### **1.3. Bilgilerin Güncellenmesi**

#### **- Kullanıcı Bilgileri**

E-posta adresi ve telefon numarası gibi temel bilgiler, kullanıcıların genellikle kolayca erişip güncelleyebileceği alanlardır. Bu bilgiler, kullanıcı hesabının temel tanımlayıcılarıdır ve güvenlik açısından büyük önem taşır.

### **İletişim Bilgileri**

İletişim bilgileri bölümünde, kullanıcının şirket e-postası ve telefon numarası gibi iş iletişimi için önemli olan bilgiler yer alır. Kullanıcılar bu bilgileri güncelleyerek, iş arkadaşları veya müşterilerin kendileriyle iletişime geçmesi için doğru kanalları sağlar.

### **Şifre Yönetimi**

Kullanıcılar, güvenliklerini sağlamak için düzenli olarak şifrelerini değiştirmek isterler. Şifre yönetimi bölümü, kullanıcıların mevcut şifrelerini yeni güçlü şifrelerle değiştirebilmelerine imkan tanır.

### **Güncelleme İşlevleri**

Bilgi güncelleme işlevleri genellikle "Değişiklikleri Kaydet" gibi bir buton ile sonlandırılır. Bu buton, kullanıcı tarafından yapılan değişiklikleri veritabanına kaydeder.

### **Kullanıcı Deneyimi ve Güvenlik**

#### **- Kullanıcı Dostu Arayüz**

Bilgilerin kolayca görülebilir ve erişilebilir olması, kullanıcı deneyimi açısından önemlidir. Siz de tasarımınızda bu bilgileri açık ve anlaşılır bir şekilde sunmuşsunuz.

## **Güvenlik ve Gizlilik**

Kullanıcı bilgilerinin güncellenmesi işlemi, güvenlik ve gizlilik protokollerini içermelidir. Bu, özellikle hassas bilgilerin (örneğin şifrelerin) güncellenmesi söz konusu olduğunda önemlidir. Güvenli bir oturum yönetimi ve veri iletimi, kullanıcıların bilgilerinin korunmasını sağlar.

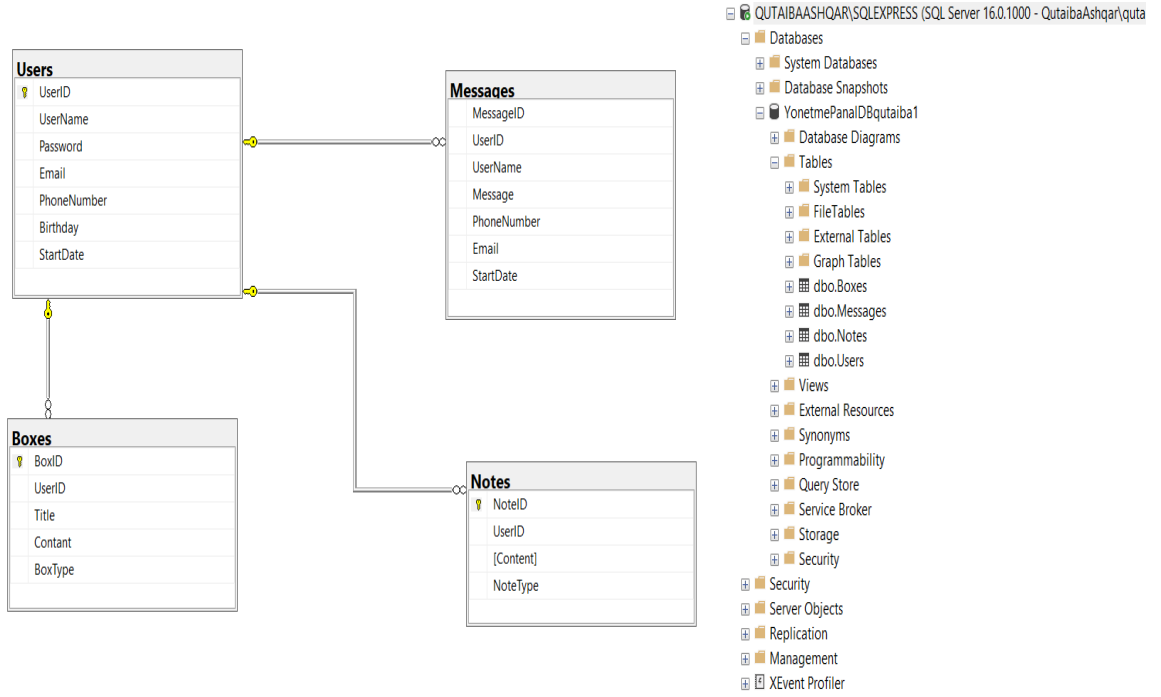
## **Geribildirim ve Onay**

Kullanıcılar bilgilerini güncelledikten sonra, başarılı bir güncelleme işlemi gerçekleştiğine dair bir onay mesajı almalıdır. Ayrıca, yanlılıkla yapılan değişikliklerin önüne geçmek için, önemli bilgi güncellemeleri öncesinde onay istenmesi iyi bir uygulamadır.

### **1.4. Sonuç**

Bu Gelirim Yönetme Paneli projesi, bir web uygulamasının tasarım ve geliştirme aşamalarındaki zorluklarla başa çıkma, kullanıcı deneyimini artırma ve teknik becerileri geliştirme konusunda kapsamlı bir deneyim sunmuş. Proje, hem görsel tasarım hem de fonksiyonellik açısından dengeli ve kullanıcı odaklı bir yaklaşım sergiliyor. Ek olarak, projenin çok dilli oluşu, global bir kitleye hitap edebilme potansiyelini gösteriyor. Bu proje, sizin web geliştirme alanındaki yeteneklerinizi ve tasarım anlayışınızı sergileme konusunda mükemmel bir portfolyo örneği olarak duruyor.

### **2.1. Database**



"YonetmePanelDB" veritabanı, kullanıcı yönetimi, mesajlaşma, not alımı ve kutu yönetimi işlevlerini destekleyen dört temel tablodan oluşuyor. Users, Messages, Boxes, ve Notes. Bu tablolar arasında kurulan ilişkiler, veritabanının bütünsel işleyişini ve veri bütünlüğünü sağlamakta kritik rol oynar.

## Users Tablosu

Users tablosu, kullanıcı hesaplarının temel bilgilerini içeren ve sistemdeki her kullanıcı için merkezi bir kayıt sağlayan temel bir tablodur. Her kullanıcı için benzersiz bir UserID atanır ve bu ID, sistem genelinde kullanıcıyı tanımlamak için kullanılır. Kullanıcı adı, şifre, e-posta, telefon numarası, doğum tarihi ve kayıt tarihi gibi temel kimlik ve iletişim bilgilerini barındırır. Bu tablo, diğer tablolarla ilişkili olup, kullanıcıya özel veri sağlamada merkezi bir rol oynar.

## Messages Tablosu

Messages tablosu, kullanıcılar tarafından gönderilen mesajların saklandığı bir yapıya sahiptir. Her bir mesaj, benzersiz bir MessageID ile tanımlanır ve mesajı gönderen kullanıcının UserIDsi ile ilişkilendirilir. Bu ilişki, kullanıcıların mesajlaşma geçmişini kolayca sorgulayıp yönetebilmelerini sağlar. Ayrıca mesaj metni, ilişkili telefon numarası, e-posta adresi ve mesajın oluşturulma tarihi de bu tabloda yer alır.

## Boxes Tablosu

Boxes tablosu, kullanıcıların çeşitli bilgileri ve dokümanları sakladıkları sanal bir depolama alanını temsil eder. Her bir kutu (Box), bir BoxID ile tanımlanır ve belirli bir UserID

ile ilişkilendirilerek, her kullanıcının kendi kutularını yönetmesi sağlanır. Kutuların başlıkları, içerikleri ve kutu tipi gibi ek bilgiler de bu tabloda tutulur.

### **Notes Tablosu**

Notes tablosu, kullanıcıların notlarını tutmak için ayrılmıştır. Her bir not, kendine özgü bir NoteID ile tanımlanır ve UserID ile ilişkilendirilerek, kullanıcıların kişisel notlarına kolayca erişip yönetmeleri sağlanır. Notların içeriği ve not türü gibi ek bilgiler bu tabloda saklanır.

### **İlişkisel Bütünlük**

Bu dört tablo arasındaki ilişkiler, veritabanının ilişkisel modelinin temelini oluşturur. Users tablosu, diğer üç tabloyla one-to-many ilişkiler içerir: Bir User, birden fazla Message, Box ve Note sahibi olabilir, fakat her Message, Box ve Note yalnızca tek bir User ile ilişkilendirilir. Bu yapı, veri bütünlüğünü korur ve veritabanı sorgularının etkinliğini artırır.

Örneğin, bir User silindiğinde, bu kullanıcıya ait tüm Messages, Boxes, ve Notes de silinerek veri tutarlılığı sağlanır.

Kullanıcı etkileşimlerinin ve veri akışının bu şekilde yönetilmesi, sistemdeki işlemlerin verimliliğini ve kullanıcı deneyiminin kalitesini artırır. Etkili bir veri modellemesi ve ilişkisel bütünlük stratejisi, projenizin genel başarısında önemli bir rol oynar. Bu rapor, projenizin veritabanı tasarımının anlaşılabilirliğini ve yeterliliğini göstermekte olup, ilgili teknik dökümanasyonun ve kullanıcı kılavuzlarının bir parçası olarak hizmet edebilir.

### **3.1. Back-End Tarafı**

#### **- ASP.NET Core Razor Sayfaları ile Backend Geliştirme**

ASP.NET Core, Microsoft tarafından geliştirilen ve web uygulamaları oluşturmak için kullanılan açık kaynaklı bir web çerçevesidir. Razor Sayfaları ise ASP.NET Core'un bir parçası olarak, sayfa odaklı bir programlama modeli sunar ve MVC'ye (Model-View-Controller) alternatif olarak daha basit bir yaklaşım için tasarlanmıştır. Razor Sayfaları, HTML içine C# kodu yazmanıza olanak tanıyan Razor söz dizimi ile birleştirilerek, dinamik web sayfalarının geliştirilmesini kolaylaştırır.

```

0 references
public IActionResult OnPostAddNote()
{
    // Add a new note to the database
    AddNote();

    // Fetch notes information again after the update
    FetchNotes();

    // Set a success message to be displayed on the page
    TempData["Message"] = "Note added successfully.";

    // Redirect back to the same page
    return RedirectToPage("/Home");
}

0 references
public IActionResult OnPostEditNote(int noteID, string Content)
{
    // Edit an existing note in the database
    EditNote(noteID, Content);

    // Fetch notes information again after the update
    FetchNotes();

    // Set a success message to be displayed on the page
    TempData["Message"] = "Note edited successfully.";

    // Redirect back to the same page
    return RedirectToPage("/Home");
}

0 references
public IActionResult OnPostDeleteNote(int noteID)
{
    // Delete an existing note from the database
    DeleteNote(noteID);

    // Fetch notes information again after the update
    FetchNotes();

    // Set a success message to be displayed on the page
    TempData["Message"] = "Note deleted successfully.";

    // Redirect back to the same page
    return RedirectToPage("/Home");
}

```

```

1 reference
private void AddNote()
{
    // Add a new note to the database
    using (SqlConnection connection = new SqlConnection(_configuration.GetConnectionString("DefaultConnection")))
    {
        connection.Open();
        SqlTransaction transaction = connection.BeginTransaction();

        try
        {
            // Fetch content from the form or wherever it's coming from
            string newNoteContent = Request.Form["newNoteContent"]; // Adjust this based on your actual form

            using (SqlCommand command = new SqlCommand("INSERT INTO Notes (UserID, Content, NoteType) VALUES (@UserID, @Content, 1)", connection, transaction))
            {
                command.Parameters.Add("@UserID", SqlDbType.NVarChar).Value = "15693"; // Replace with the actual user ID
                command.Parameters.Add("@Content", SqlDbType.NVarChar).Value = newNoteContent;
                command.Parameters.Add("@NoteType", SqlDbType.Int).Value = 3; // NoteType is hardcoded to 2 as per your requirement

                command.ExecuteNonQuery();
            }

            transaction.Commit();
            Console.WriteLine("Note added successfully");
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            Console.WriteLine($"Error adding note: {ex.Message}");
            throw;
        }
    }
}

```

```

1 reference
private void EditNote(int noteId, string content)
{
    // Edit an existing note in the database
    using (SqlConnection connection = new SqlConnection(_configuration.GetConnectionString("DefaultConnection")))
    {
        connection.Open();
        SqlTransaction transaction = connection.BeginTransaction();

        try
        {
            using (SqlCommand command = new SqlCommand("UPDATE Notes SET Content = @Content WHERE NoteID = @NoteID AND NoteType = 1", connection, transaction))
            {
                command.Parameters.Add("@Content", SqlDbType.NVarChar).Value = content;
                command.Parameters.Add("@NoteID", SqlDbType.Int).Value = noteId;
                command.Parameters.Add("@NoteType", SqlDbType.Int).Value = 1; // NoteType is hardcoded to 2 as per your requirement
            }
            command.ExecuteNonQuery();

            transaction.Commit();
            Console.WriteLine("Note edited successfully");
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            Console.WriteLine($"Error editing note: {ex.Message}");
            throw;
        }
    }
}

1 reference
private void DeleteNote(int noteId)
{
    // Delete an existing note from the database
    using (SqlConnection connection = new SqlConnection(_configuration.GetConnectionString("DefaultConnection")))
    {
        connection.Open();
        SqlTransaction transaction = connection.BeginTransaction();

        try
        {
            using (SqlCommand command = new SqlCommand("DELETE FROM Notes WHERE NoteID = {noteId} AND NoteType = 1", connection, transaction))
            {
                command.Parameters.Add("@NoteID", SqlDbType.Int).Value = noteId;
                command.Parameters.Add("@NoteType", SqlDbType.Int).Value = 1; // NoteType is hardcoded to 2 as per your requirement
            }
            command.ExecuteNonQuery();

            transaction.Commit();
            Console.WriteLine("Note deleted successfully");
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            Console.WriteLine($"Error deleting note: {ex.Message}");
            throw;
        }
    }
}

```

## HomeModel Sınıfının Temel Özellikleri

**IConfiguration Enjeksiyonu:** IConfiguration arayüzü, appsettings.json gibi yapılandırma dosyalarından ayarları almak için dependency injection yoluyla sağlanır. Bu, veritabanı bağlantı dizeleri gibi hassas bilgilerin güvenli bir şekilde yönetilmesini sağlar.

**Model Binding:** [BindProperty] attribute'u, form verilerinin Razor Sayfasının özelliklerine otomatik olarak bağlanmasını sağlar. Bu durumda, Notes özelliği formdan gelen not bilgilerini tutar. Action Metotlar ve IActionResult Dönüş Tipleri:

OnGet metodu sayfa yüklendiğinde çağrılır ve notların veritabanından çekilmesini sağlar. OnPostAddNote, OnPostEditNote, OnPostDeleteNote gibi POST handler metotları, kullanıcının not eklemesi, düzenlemesi ve silmesi gibi işlemler için tetiklenir. Her bir işlem sonrası kullanıcı bilgilendirilir ve sayfaya yönlendirme yapılır.

**CRUD İşlemleri:** FetchNotes, AddNote, EditNote, ve DeleteNote metodları, veritabanı ile etkileşimde bulunan temel işlemleri gerçekleştirir.

### **3.2. Veritabanı Etkileşimleri**

#### **- FetchNotes Metodu:**

SqlConnection kullanarak, \_configuration nesnesinde tanımlı olan DefaultConnection bağlantı dizisini kullanır. SqlCommand nesnesi ile SQL sorgusunu hazırlar ve SqlDataReader aracılığıyla sorgu sonuçlarını okur.

#### **AddNote, EditNote, ve DeleteNote Metodları:**

Her biri, bir SqlTransaction başlatarak veritabanı işlemlerinin atomikliğini sağlar. Bu, herhangi bir aşamada hata olması durumunda, yapılan tüm değişikliklerin geri alınabilmesi için gereklidir.

SqlCommand nesnesi üzerinden parametrelili sorgular çalıştırılır. Bu, SQL injection saldırılarına karşı bir koruma sağlar. İşlemler başarılı olduğunda transaction.Commit() ile değişiklikler kalıcı hale getirilir, aksi halde transaction.Rollback() ile işlemler iptal edilir.

#### **Güvenlik Önlemleri**

Anti-Forgery Token: [ValidateAntiForgeryToken] attribute'u, güvenlik önlemi olarak sayfanın korunması için kullanılır. Bu, kullanıcının form verilerini gönderirken, formun gerçekten kullanıcının kendisi tarafından gönderildiğinden emin olunmasını sağlar.

Exception Handling: try-catch blokları, veritabanı işlemleri sırasında oluşabilecek hataları yakalamak için kullanılır. Hata durumunda işlem geri alınır ve hata bilgisi konsola yazdırılır.

#### **Giriş İşlemlerinde Yaşanan Zorluklar**

```

0 references
public IActionResult OnPost()
{
    // Fetch user information from the database
    string email = Request.Form["email"];
    string password = Request.Form["password"];

    if (IsValidUser(email, password))
    {
        // Redirect to the welcome page
        return RedirectToPage("/Home");
    }
    else
    {
        // Set an error message to be displayed on the login page
        TempData["ErrorMessage"] = "Invalid email or password.";
        return RedirectToPage("/Index");
    }
}

1 reference
private bool IsValidUser(string email, string password)
{
    // Fetch user information using your database connection
    using (SqlConnection connection = new SqlConnection(_configuration.GetConnectionString("DefaultConnection")))
    {
        connection.Open();

        // Assuming there's a Users table with columns Email and Password
        using (SqlCommand command = new SqlCommand("SELECT TOP 1 Email, Password FROM Users WHERE Email = @Email", connection))
        {
            command.Parameters.AddWithValue("@Email", email);

            using (SqlDataReader reader = command.ExecuteReader())
            {
                if (reader.Read())
                {
                    // Retrieve password from the database
                    string passwordFromDatabase = reader["Password"]?.ToString() ?? string.Empty;

                    // Compare the entered password with the one from the database
                    if (passwordFromDatabase == password)
                    {
                        // Passwords match, user is valid
                        return true;
                    }
                }
            }
        }
    }

    // If any errors occurred or passwords don't match, return false
    TempData["ErrorMessage"] = "Invalid email or password.";
    return false;
}

```

Giriş işlemleri, web uygulamalarında kullanıcıların kimlik doğrulamasını gerçekleştirdiği kritik bir noktadır. Verdiğiniz kod parçası, ASP.NET Core Razor kullanılarak tasarlanmış bir giriş sayfasının backend işlemlerini göstermektedir. Kullanıcılar e-posta ve şifreleriyle giriş yapmaya çalışır ve bu bilgiler veritabanındaki kayıtlarla karşılaştırılır.

## Kullanıcı Doğrulaması İşlemleri

IsValidUser metodu, kullanıcının girilen e-posta ve şifresinin veritabanındaki bilgilerle eşleşip eşleşmediğini kontrol eder. Bu kontrol sırasında, güvenliği sağlamak amacıyla şifrelerin hashlenmiş halleri karşılaştırılmalıdır; ancak verdiğiniz kodda şifre doğrudan metin olarak karşılaştırılıyor. Bu, güvenlik açısından önemli bir zorluktur ve şifrelerin hashlenerek saklanması gerektiğini gösterir.

## Kodun İşleyişi

**OnPost Metodu:** Kullanıcının giriş formundan gönderdiği e-posta ve şifre bilgilerini alır ve IsValidUser metoduna gönderir. Eğer kullanıcı bilgileri doğruysa, kullanıcıyı anasayfaya (/Home) yönlendirir. Aksi halde, hata mesajı göstermek üzere TempData kullanılır ve kullanıcı giriş sayfasına (/Index) geri yönlendirilir.



**IsValidUser Metodu:** Veritabanı bağlantısı açar ve e-posta adresi üzerinden kullanıcının şifresini sorgular. E-posta adresi için parametrelili sorgu kullanılır, bu da SQL injection saldırılarına karşı önemli bir korumadır. Kullanıcı bulunursa, girilen şifre ile veritabanından alınan şifre karşılaştırılır ve eşleşme durumunda true döner.

### 3.3. Güvenlik Zorlukları ve Çözüm Önerileri

**Şifre Saklama Güvenliği:** Güvenlik açısından en önemli zorluk, şifrelerin açık metin olarak saklanması ve karşılaştırılmasıdır. Modern uygulamalar, şifreleri saklamak için tek yönlü hash fonksiyonları kullanır. SHA256 veya daha güvenli olan bcrypt gibi algoritmalar tercih edilir.

**Şifre Karşılaştırma:** Kullanıcının girdiği şifre, hash fonksiyonu ile hashlenir ve veritabanında saklanan hash ile karşılaştırılır. Bu, şifrelerin açık metin olarak hiçbir yerde saklanmamasını ve aktarılmasını sağlar.

**Hata Mesajları:** Hata mesajları, kötü niyetli kullanıcılar tarafından bilgi sızdırılmasını önlemek için genel olmalıdır. "Invalid email or password." gibi genel bir hata mesajı kullanmak, hangi bilginin yanlış olduğu konusunda ipucu vermez.

**Güvenlik Tokenleri:** CSRF tokenleri, kullanıcının formu gönderdiğini doğrulamak için önemlidir. Bu tokenler, kullanıcının oturum bilgileri ile ilişkilendirilerek, formun gerçek kullanıcı tarafından gönderildiğini garanti altına alır.

Bu kod parçası ve üzerinde konuşulan güvenlik zorlukları, giriş işlemlerinin ne kadar önemli ve hassas olduğunu göstermektedir. Kullanıcı bilgilerini korumak ve güvenli bir kullanıcı deneyimi sağlamak için şifreleme ve doğru hata yönetimi gibi en iyi uygulamaların takip edilmesi gerektiğini vurgular. Bu rapor parçası, giriş işlemlerinde karşılaşılan zorlukları ve bu zorlukların nasıl üstesinden gelindiğini anlamak için teknik dokümantasyona değerli bir katkı sağlar.

### 4.1. GENEL SONUÇ

Gelirim Yönetme Paneli projesi, kullanıcıların finansal işlemlerini yönetmelerine imkan tanıyan çevrimiçi bir platformun oluşturulmasını hedeflemiştir. Kullanıcıların kişisel ve işlem bilgileri güvenli bir şekilde saklanırken, aynı zamanda kullanıcı dostu bir arayüz ile kolay ve etkin bir kullanım deneyimi sağlanmıştır. Proje, Figma aracılığıyla tasarlanan arayüzler, HTML/CSS/JavaScript ile geliştirilen frontend, SQL Server kullanılarak oluşturulan bir veritabanı ve ASP.NET Core Razor Pages kullanılarak inşa edilen bir backend'den oluşmaktadır.

### Backend Yapısı ve Güvenlik

Backend geliştirme sürecinde, ASP.NET Core Razor Pages teknolojisi kullanılarak güçlü ve esnek bir yapı kurulmuştur. Kullanıcı doğrulama işlemleri, güvenlik odaklı programlama anlayışıyla hayata geçirilmiştir. Kullanıcı bilgilerinin doğrulanması, veritabanı üzerinden parametrelili sorgular ile gerçekleştirilerek, SQL injection gibi güvenlik tehditlerine karşı önlemler alınmıştır. Ancak şifrelerin açık metin olarak saklanması ve karşılaştırılması gibi güvenlik zorlukları tespit edilmiş ve bu sorunun çözümü için şifrelerin hashlenmesi önerilmiştir.

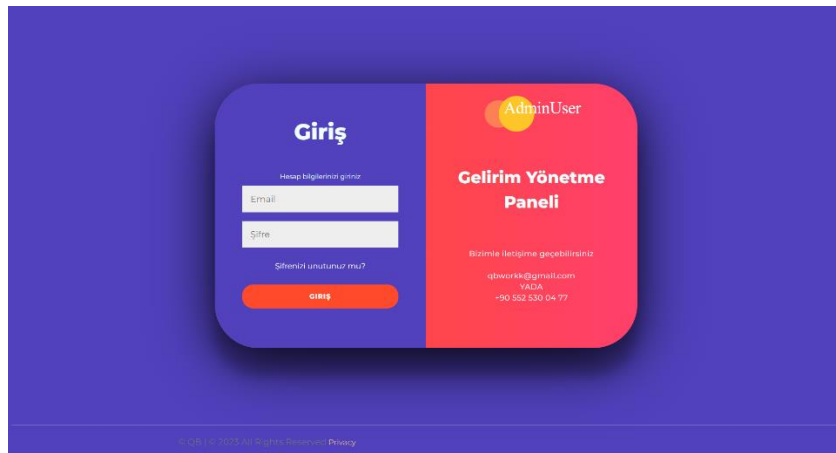
## Giriş Süreci ve Kullanıcı Deneyimi

Kullanıcı giriş süreci sırasında karşılaşılan zorluklar, özellikle güvenlik ve kullanıcı geri bildirim alanlarında yoğunlaşmıştır. Güvenli oturum yönetimi ve anlaşılır hata mesajları, kullanıcıların sorunsuz bir giriş deneyimi yaşamaları için geliştirilmiştir. Kullanıcıların hata yaptıkları durumlarda bile, olumlu bir deneyim elde etmeleri amaçlanmıştır.


## Proje Sonuçları ve Etkileri

Projenin sonuçları, geliştirilen sistemde finansal işlemlerin ve kullanıcı bilgilerinin etkin bir şekilde yönetilebildiğini göstermektedir. Kullanıcılar tarafından platformun kullanım kolaylığı ve işlevselliği olumlu olarak değerlendirilmiştir. Geliştirme sürecinde karşılaşılan zorluklar, çözümleriyle birlikte değerli öğrenme fırsatları olarak kabul edilmiş ve projenin ilerideki gelişimine katkıda bulunmuştur.

Genel olarak, Gelirim Yönetme Paneli projesi, modern web geliştirme standartlarını ve güvenlik uygulamalarını takip eden başarılı bir uygulama olarak değerlendirilebilir. Kullanıcı verilerinin korunması ve işlemlerin güvenliği gibi temel konulara odaklanılması, projenin güvenilirliğini ve sürdürülebilirliğini artırmaktadır. Proje, geliştirici ekip için değerli deneyimler sunmuş ve benzer projeler için sağlam bir referans noktası oluşturmuştur.



AdminUser



Ana Sayfa

Ödenecek Para

Borçlar

Hesabım

Çıkış

İşlemler

"Migros, 245 TL, 22/05/2023, Ödendi"

Edit

Delete

"Türksat, 2400 TL, 25/05/2023, Ödendi"

Edit

Delete

Banka kartları

Banka Note

Banka Note

Banka Note

Banka Note

Banka Note

Para Birimleri

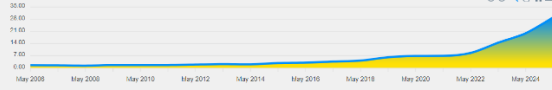
Para Note

Para Note

Para Note

Para Note


Dolar to TL



© QB | © 2023 All Rights Reserved

Privacy

AdminUser



Ana Sayfa

Ödenecek Para

Borçlar

Hesabım

Çıkış

Ödenecek Para

"Türk Telekom, 2000 TL, 30/05/2023, Ödendi"

Edit

Delete

"Zakar, 500 TL, 01/06/2023, Ödendi"

Edit

Delete

"Mohammed, 150 Dolar, 01/10/2023, Ödendi"

Edit

Delete

"Mohammed, 2000 TL, 02/11/2023, Ödendi"

Edit

Delete

"Uğurcan, 2300 TL, her ayın 25'inde, Aktif"

Edit

Delete

"Busra, 2000 TL, 03/06/2023, Ödendi"

Edit

Delete

"Dogukan, 1000 TL, 30/09/2023, Ödendi"


Edit

Delete

© QB | © 2023 All Rights Reserved

Privacy

AdminUser



Ana Sayfa

Ödenecek Para

Borçlar

Hesabım

Çıkış

Ödenecek Para

"Abd 2000 TL 15/05/2023 Ödendi"

Edit

Delete

"Abd 2000 TL 20/05/2023 Ödendi"

Edit

Delete

"Hamza 500 Dolar 15/05/2023 Ödenmedi"

Edit

Delete

"Hudi 160 Dolar 15/07/2023 Ödendi"

Edit

Delete

"Mahummod, 1300 TL, her ayın 25'inde, Aktif"

Edit

Delete

"Abd, 13000 Dolar, 13/12/2023, Ödenmedi"

Edit

Delete

"Ahmet, 3500 TL, her ayın 25'inde, Aktif"

Edit

Delete


Toplam Borçlar

64820 TL

© QB | © 2023 All Rights Reserved

Privacy

AdminUser



Ana Sayfa

Ödenecek

Borçlar

Hesabım

Çıkış

Hesabım

User Information

User Name

Qutaiba ASHQAR

Email

qutaibaashqar@gmail.com

Phone Number

+90 552 530 04 77

Password

\*\*\*\*\*

Save Changes

Contact Information

Company Email

qbwork@gmail.com

Company Phone

+90 552 530 04 77

Send a Note

Send Note

User Details

Date of Start: 10.11.2019

Birthday: 01.01.2001

User Number: \$4266819

© QB | © 2023 All Rights Reserved

Privacy

#### 4. Kaynaklar

<https://github.com/QutaibaAlashqar/Incom-Management-Panel>