BLM4531 - Web Programlama Final Projesi

İZİN VE PUANTAJ TAKİP SİSTEMİ PROJE RAPORU

Öğrenci Adı: İlayda Nihal Kavanoz **Öğrenci No:** 22291015 **GitHub:** https://github.com/nia1da/izin-puantaj-sistemi

Videosu: https://drive.google.com/file/d/1ubapCLStQ0y0iJM2KQoxEd7JyrdQ9R0P/view?usp=sharing

3. Projenin Temel Özellikleri ve Çalışma Mantığı

Sistemi sadece veri kaydeden bir defter gibi değil, veriyi işleyen akıllı bir araç olarak tasarladım.

A. Yönetici (Admin) Paneli ve Hız Admin paneli projenin kumanda merkezidir. Burada **React** kullanmamın en büyük avantajını görüyoruz: Bir personeli sildiğimde veya iznini onayladığımda sayfanın yenilenmesini beklemiyoruz. Her şey anlık olarak (State değişimiyle) ekrana yansıyor. Bu da kullanıcıya çok hızlı bir deneyim sunuyor.

B. "Sunucuyu Yormayan" Puantaj Hesaplaması Bu projede yaptığım en önemli optimizasyon budur. Yöneticinin "Ahmet bu ay toplam kaç saat çalıştı?" sorusunun cevabını sunucuya (Backend'e) yüklemedim.

Backend bana sadece giriş-çıkış saatlerini (ham veriyi) gönderiyor. Ben bu veriyi alıp, JavaScript'in gücünü kullanarak tarayıcıda hesaplıyorum.

C. İzin Onay Mekanizması İzin sistemi "Talep Et -> Bekle -> Onayla" mantığıyla çalışır.

1. Personel izin ister, talep veritabanına **"Bekliyor" (Pending)** olarak düşer.
2. Admin panelinde sadece bekleyenler listelenir.
3. Admin "Onayla" dediğinde sistem hem iznin durumunu günceller hem de personelin kalan izin hakkından o gün sayısını düşer.

1. Projenin Amacı ve Kapsamı

Bu projenin temel amacı, kurumsal firmalarda kağıt üzerinde yürütülen izin talep süreçlerini ve çalışanların giriş-çıkış (puantaj) takiplerini dijitalleştirerek merkezi bir platformda toplamaktır.

Manuel süreçlerde yaşanan "İzin formum kayboldu", "Geçen ay kaç saat çalıştım?" gibi sorunları ortadan kaldırmayı hedefledim. Sistem sayesinde çalışanlar şeffaf bir şekilde kendi kayıtlarını görebilirken, yöneticiler (Admin) tek bir panelden tüm personeli yönetebilmektedir.

2. Kullanılan Teknolojiler ve Mimari

Projeyi geliştirirken modern ve endüstri standardı olan **Full-Stack** bir yapı tercih ettim.

- **Backend (Sunucu):** .NET 8.0 Web API (C#)
- **Frontend (Arayüz):** React.js + Vite (JavaScript)
- **Veritabanı:** SQLite (Entity Framework Core ile Code-First Yaklaşımı)
- **Tasarım:** Tailwind CSS (Responsive ve hızlı tasarım için)
- **Versiyon Kontrol:** Git & GitHub

Sistem, **Client-Server** mimarisiyle çalışmaktadır. Frontend ve Backend tamamen birbirinden bağımsız çalışır ve `RESTful API` üzerinden JSON verisi ile haberleşir.

3. Temel Özellikler ve Kod Yapısı

A. Puantaj (Check-In / Check-Out) Mantığı

```
lic IActionResult CheckOut([FromBody] CheckOutRequest request)

// 1. Kullanıcının o günkü kapanmamış kaydını bul
var record = _context.AttendanceRecords
    .FirstOrDefault(r => r.UserId == request.UserId && r.Date.Date == DateTime.T

if (record == null) return BadRequest("Aktif bir giriş kaydı bulunamadı.");

// 2. Çıkış saatini işle
record.CheckOutTime = DateTime.Now.TimeOfDay.ToString(@"hh\:mm\:ss");

// 3. Giriş ve Çıkış arasındaki farkı (TimeSpan) hesapla
TimeSpan duration = TimeSpan.Parse(record.CheckOutTime) - TimeSpan.Parse(record.

// 4. Toplam saati kaydet
record.TotalHours = duration.TotalHours;

_context.SaveChanges();
return Ok(new { message = "Çıkış yapıldı.", totalHours = record.TotalHours });
```

Çalışan işe geldiğinde "Giriş Yap" butonuna basar. Sistem o anki saati kaydeder. Çıkış yaparken ise sistem, giriş saatini bulur ve aradaki farkı alarak "Toplam Çalışma Süresi"ni hesaplar.**Backend Kodu (`AttendanceController.cs`):** Bu kısımda `CheckOut` işlemi sırasında süreyi nasıl hesapladığımı görebilirsiniz:

EF Core kullanarak veritabanındaki ilgili kaydı çekiyorum ve C#'ın `TimeSpan` özelliği ile iki saat arasındaki farkı matematiksel olarak hesaplatıyorum.

Admin Panelinde Dinamik Raporlama

Yöneticinin, personelin aylık ne kadar çalıştığını görmesi gerekiyordu. Bu işlem için Backend'den tüm ham veriyi çekip, hesaplamayı Frontend tarafında yaparak sunucu yükünü azalttım.

Frontend Kodu (`AdminDashboard.jsx`): Backend'den gelen ham listeyi, kişi bazında gruplayıp toplamak için JavaScript'in `reduce` fonksiyonunu kullandım:

```
onst userTotals = attendanceRecords.reduce((acc, record) => {
 const name = record.userName || "Bilinmeyen";

 // Eğer bu kişi daha önce listeye eklenmediyse, başlat
 if (!acc[name]) acc[name] = 0;

 // Kişinin toplam saatine, bu kayıttaki süreyi ekle
 acc[name] += (record.totalHours || 0);

 return acc;
, {});
```

C. Frontend Güvenlik Katmanı (Protected Routes)

```
const ProtectedRoute = ({ children }) => {
  // Tarayıcı hafızasından (localStorage) kullanıcı bilgisini kontr
  const user = JSON.parse(localStorage.getItem("user"));

  // Eğer kullanıcı giriş yapmamışsa, onu direkt Login sayfasına fı
  if (!user) {
    return <Navigate to="/" replace />;
  }

  // Giriş yapmışsa gitmek istediği sayfayı (children) göster
  return children;
};
```

Kullanıcı oturum açmadan bu sayfalara erişmeye çalıştığında, sistem onları otomatik olarak yakalayıp giriş ekranına yönlendiriyor. Bu da uygulamanın güvenliğini istemci tarafında garanti altına alıyor.

D. Veritabanı Başlangıç Verisi

Projeyi her sıfırladığımda veritabanına girip elle "Admin" kullanıcısı oluşturmamak için Bunu tekte yapmak çin **"Seed Data"** özelliğini kullandım.

```
// Veritabanı ilk oluştuğu anda bu kullan
modelBuilder.Entity<User>().HasData(
    new User
    {
        Id = 1,
        Username = "admin",
        Password = "1234", // Gerçek haya
        Department = "Yönetim",
        Role = "admin"
    }
);
```