1. Kimlik Doğrulama Sistemi: JWT ve httpOnly Cookie Seçimi

Projenin kullanıcı giriş sistemi için normal session tabanlı yaklaşım yerine JWT kullanmaya karar verdim.

JWT'yi seçmemin ana sebebi stateless olması. Yani sunucu her kullanıcının oturum bilgisini hafızasında tutmuyor. Kullanıcı giriş yaptığında ona imzalı bir token veriyorum ve sonraki isteklerde bu token yeterli oluyor. Bu özellikle uygulama büyüdüğünde sunucu performansı açısından çok faydalı.

Token'ı nereye saklayacağım konusunda ise httpOnly cookie tercih ettim. Çünkü localStorage gibi yerlerde saklamak XSS saldırılarına açık kapı bırakıyor. httpOnly cookie ile token'a sadece sunucu erişebiliyor, JavaScript kodlarım bile okuyamıyor. Frontend tarafında kimlik kontrolü için /api/me gibi endpoint'lere istek atmam gerekiyor ama bu güvenlik açısından çok daha sağlam bir yöntem.

2. Socket.IO İçin Özel Sunucu Kurulumu

Next.js'in kendi sunucusu Socket.IO gibi WebSocket bağlantıları için yeterli değil. Çünkü WebSocket'ler kalıcı bağlantı gerektiriyor ve Next.js'in standart sunucusu bunun için optimize edilmemiş.

Bu yüzden server.ts dosyasında Node.js'in http modülüyle kendi sunucumu oluşturdum. Bu sunucu şöyle çalışıyor: gelen isteklere bakıyor, eğer normal web sayfası isteği ise Next.js'e yönlendiriyor, WebSocket isteği ise Socket.IO'ya gönderiyor. Böylece tek port üzerinden hem web sitemi hem sohbet sistemimi çalıştırabiliyorum.

Geliştirme sırasında nodemon kullandım çünkü kodu değiştirdiğimde sunucuyu sürekli manuel olarak yeniden başlatmak zorunda kalmak istemedim. Hem daha kolay ve sorunsuz çalıştırabilidğim için kullanmak istedim.

3. Veritabanı Tasarımı: Mongoose Tercihi

Case dokümanda MongoDB için Mongoose veya Prisma seçeneği vardı, ben Mongoose'u tercih ettim.

MongoDB normalde şemasız ama bu projede kullanıcı, mesaj, oda gibi belirli yapılar var. Mongoose ile bu yapılar için katı şemalar tanımlayabiliyorum. Bu da kodda veri tutarlılığını sağlıyor ve hataları önlüyor.

Özellikle mesajları çekerken kullanıcı adlarını da göstermek için Mongoose'un populate özelliğini kullandım. Bu tek satırda ilişkisel veri çekmeyi çok kolay hale getiriyor.

4. API Organizasyonu ve Veri Yönetimi

Backend tarafında Next.js App Router'ın route.ts yapısını kullandım. Bu hem serverless ortamlara kolay deploy edilebilen hem de düzenli bir yapı sunuyor.

Frontend'te API isteklerini direkt komponentlerin içinde yapmak yerine lib/api.ts gibi merkezi dosyalarda toplamamı tercih ettim. Bu şekilde aynı kodu tekrar tekrar yazmıyorum ve API endpoint'lerinden biri değiştiğinde tek yerden düzenleyebiliyorum. Kod daha temiz ve bakımı daha kolay oluyor.