



**SOFTITO 4. DÖNEM 1. GRUP
BİLİŞİM GÜVENLİĞİ MİMARİSİ
BİTİRME PROJESİ RAPORU**



**OrderFlow: Rol Tabanlı Kurumsal İş Akışı
ve Lojistik Yönetim Sistemi**

React 19 · Vite · TypeScript · Tailwind CSS v4

Ezgi

Ön Uç (Frontend) Geliştirici

Ekim, 2025

softITO Yazılım Bilişim Akademisi, İstanbul

ORDERFLOW

PROJE KÜNYESİ

Alan	Bilgi
Proje Adı	OrderFlow İş ve Dağıtım Yönetim Sistemi
Geliştirici	Ezgi
Dönem / Grup	4. Dönem – 1. Grup
Ders	Bilişim Güvenliği Mimarisi
Eğitim Kurumu	softITo Yazılım Bilişim Akademisi
Konum	İstanbul, Türkiye
Tarih	Ekim, 2025
Teknoloji Yığını	React 19, Vite, TypeScript, Tailwind CSS v4, Lucide React

ÖZET

Bu çalışmada, kurumsal bir firmanın satış teklifinden ürünün müşteriye teslim edilmesine kadar uzanan iş akışını tek bir arayüz üzerinden yönetebilen **OrderFlow** adlı rol tabanlı bir ERP (Kurumsal Kaynak Planlaması) simülasyonu geliştirilmiştir. Uygulama; React 19, Vite ve TypeScript üzerine kurulmuş, Tailwind CSS v4 ile tasarlanmış, tamamen istemci tarafında (client-side) çalışan bir tek sayfa uygulamasıdır (SPA). Sistemin güvenlik mimarisi, Rol Tabanlı Erişim Denetimi (RBAC) prensibine dayanmakta; Yönetici, Satış, Muhasebe ve Depo rolleri yalnızca kendi görev alanlarına ait modülleri görüntüleyebilmektedir. Veri kalıcılığı tarayıcının `localStorage` alanı üzerinden sağlanmış, tüm kullanıcı işlemleri denetim izi (audit trail) günlüğüne kaydedilmiştir. Çalışmanın sonunda, en az yetki (least privilege) ilkesinin bir ön uç uygulamasında nasıl uygulanabileceği ve istemci taraflı yetkilendirmenin sınırları tartışılmıştır.

Anahtar Kelimeler: RBAC, ERP, React 19, TypeScript, Tailwind CSS v4, Denetim İz, En Az Yetki İlkesi

• 1. GİRİŞ

- 1.1. Projenin Amacı ve Önemi
- 1.2. Projenin Hedef Kitlesi
- 1.3. Proje Kısıtları ve Sınırları
- 1.4. Kapsanan İş Akışı

• 2. MATERYAL VE YÖNTEM

- 2.1. Ön Uç (Frontend) Teknolojileri
 - 2.1.1. React 19 Mimarisi
 - 2.1.2. Vite Derleme Motoru
 - 2.1.3. TypeScript Kullanımı
- 2.2. Arayüz ve Tasarım Standartları
 - 2.2.1. Tailwind CSS v4 Entegrasyonu
 - 2.2.2. Lucide React İkonografisi
 - 2.2.3. Gece Modu (Dark Mode) ve Custom Variant Yapılandırması
 - 2.2.4. Premium Glassmorphism Tasarım İlkeleri
- 2.3. Veri Yönetimi ve Kalıcılık
 - 2.3.1. LocalStorage Serializasyonu
 - 2.3.2. Durum Yönetimi (State Management)
 - 2.3.3. Veri Modelleri
- 2.4. Proje Dizin Yapısı

• 3. SİSTEM MİMARİSİ VE ROL TABANLI ERİŞİM DENETİMİ (RBAC)

- 3.1. RBAC Kavramı ve En Az Yetki İlkesi
- 3.2. Rol Matrisi ve Erişim Kısıtlamaları
- 3.3. Giriş Kimlik Doğrulama Süreci
- 3.4. Menü Filtreleme ve Yetki Doğrulama Katmanı
- 3.5. Denetim İzi (Audit Trail) Mimarisi
- 3.6. Güvenlik Değerlendirmesi ve İstemci Tarafı Yetkilendirmenin Sınırları

• 4. UYGULAMA BİLEŞENLERİ VE DETAYLI MODÜL ANALİZİ

- 4.1. Ana Uygulama Modülü (`App.tsx`)
- 4.2. Kimlik Doğrulama Ekranı (`LoginView.tsx`)
- 4.3. Navigasyon ve Kontrol Panelleri (`Sidebar.tsx` , `Header.tsx`)
- 4.4. Kontrol Paneli (`DashboardView.tsx`)
- 4.5. Müşteri Yönetimi (`CustomersView.tsx`)
- 4.6. Teklif Yönetimi (`QuotationsView.tsx`)
- 4.7. Sipariş Yönetimi (`OrdersView.tsx`)
- 4.8. Finans ve Ödeme Takibi (`PaymentsView.tsx`)
- 4.9. Lojistik ve Teslimat Takibi (`DeliveriesView.tsx`)
- 4.10. Operasyonel Takvim (`CalendarView.tsx`)
- 4.11. Analizler ve Raporlama (`ReportingView.tsx`)
- 4.12. Sistem Etkinlik Günlüğü (`ActivityLogView.tsx`)
- 4.13. Ortak Ekleme Modalı (`CreateNewModal.tsx`)

• 5. KURULUM VE ÇALIŞTIRMA TALİMATLARI

- 5.1. Gereksinimler

- 5.2. Projenin Çalıştırılması
 - 5.3. Test Kullanıcı Bilgileri
 - 5.4. Projenin Derlenmesi (Build)
 - 5.5. Sık Karşılaşılan Sorunlar
 - **6. TEST VE DOĞRULAMA**
 - 6.1. Rol Tabanlı Erişim Test Senaryoları
 - 6.2. İş Akışı Test Senaryoları
 - 6.3. Arayüz ve Kalıcılık Testleri
 - **7. SONUÇ VE GELECEK PLANLAR**
 - **8. KAYNAKÇA**
-

1. GİRİŞ

1.1. Projenin Amacı ve Önemi

Günümüz iş dünyasında süreçlerin dijitalleştirilmesi, operasyonel hız ve doğruluğu artıran en önemli faktördür. Kurumsal firmalarda satış, muhasebe ve lojistik departmanları çoğu zaman birbirinden kopuk araçlar (elektronik tablolar, e-posta zincirleri, ayrı takip yazılımları) kullanmakta; bu durum hem veri tekrarına hem de sorumluluk belirsizliğine yol açmaktadır.

OrderFlow, kurumsal firmaların satış tekliflerinden teslimat anına kadar olan tüm iş adımlarını tek bir ekran üzerinden koordine edebilmesi amacıyla geliştirilmiş bütünsel bir ERP simülasyonudur. Projenin temel hedefleri şunlardır:

- Süreç bütünlüğü:** Teklif → Sipariş → Ödeme → Teslimat zincirini kopmadan, tek veri kaynağı üzerinden yürütmek.
- Yetki ayrıştırması:** Her departmanın yalnızca kendi sorumluluk alanındaki verilere erişmesini sağlayarak **en az yetki (least privilege)** ilkesini uygulamak.
- Şeffaflık ve denetlenebilirlik:** Sistemde yapılan her işlemi kim, ne zaman, hangi kayıt üzerinde yaptı sorularına yanıt verecek şekilde günlüklemek.
- Modern arayüz standartları:** Karanlık mod, erişilebilir renk kontrastları ve responsive tasarım ile her cihazda tutarlı bir deneyim sunmak.

Ders kapsamı olan **Bilişim Güvenliği Mimarisi** açısından projenin önemi, RBAC (Role-Based Access Control) modelinin teorik tanımından somut bir yazılım mimarisine dönüştürülmesi ve bu dönüşümün getirdiği tasarım kararlarının (menü filtreleme, oturum yönetimi, denetim izi, rol değişiminde otomatik çıkış) uygulamalı olarak gösterilmesidir.

1.2. Projenin Hedef Kitlesi

Uygulama; satış ekipleri, muhasebe uzmanları, depo ve lojistik yöneticileri ile genel yöneticilerden (Admin) oluşan geniş bir kullanıcı kitlesine hitap eder. Her kullanıcı grubu, yalnızca kendi görev alanı ile ilgili sayfaları görüntüleyerek bilgi kirliliğinden uzak, sadeleştirilmiş bir çalışma alanına sahip olur.

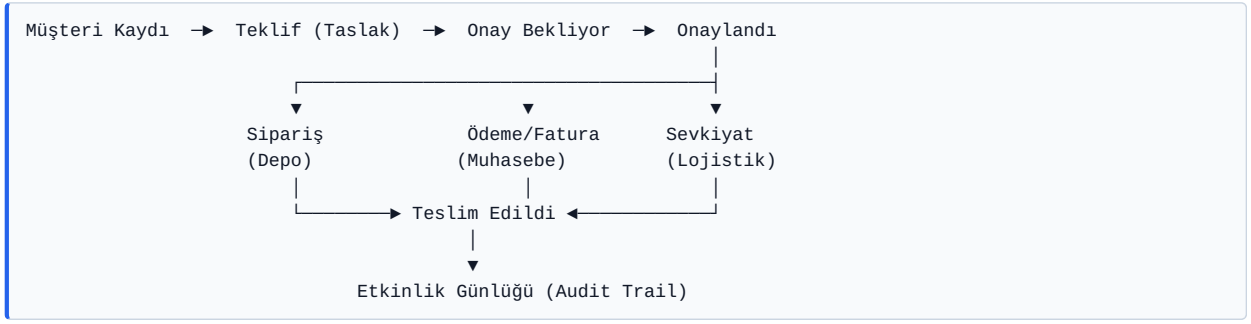
Rol	Birincil Sorumluluk	Sistemdeki Temel Görevi
Yönetici (Admin)	Genel koordinasyon ve denetim	Tüm modüllere erişim, etkinlik günlüğü incelemesi
Satış (Sales)	Müşteri ilişkileri ve teklif	Müşteri kaydı açma, teklif oluşturma ve takip
Muhasebe (Accounting)	Finansal takip	Fatura ve tahsilat yönetimi, mali raporlama
Depo (Warehouse)	Lojistik operasyon	Sipariş hazırlama, sevkiyat ve teslimat takibi

1.3. Proje Kısıtları ve Sınırları

- İstemci Odaklı Çalışma:** Projede herhangi bir fiziksel arka uç (Backend) sunucusu ve SQL/NoSQL veritabanı kurulmamıştır. Tüm işlemler istemci tarayıcısında gerçekleşir. Bu tercih, dersin kapsamının mimari tasarım ve erişim denetimi modellemesi olması nedeniyle bilinçli olarak yapılmıştır.
- Statik Kimlik Doğrulama:** Güvenlik katmanı, departmanlar için belirlenmiş statik roller ve bu rollere özel şifre kutuları ile **simüle edilmiştir**. Şifreler hash'lenmemekte, sunucu tarafında doğrulanmamaktadır; dolayısıyla gerçek bir üretim güvenliği sağlamaz (bkz. Bölüm 3.6).
- Simüle Telemetry Verileri:** Dağıtım ekranındaki GPS harita hareketleri ve durum ilerletme simülatörü algoritmik olarak anlık tetiklenmektedir; gerçek bir araç takip donanımından veri alınmamaktadır.
- Tek Kullanıcı Oturum:** Uygulama aynı anda tek bir tarayıcı oturumu varsayar; eşzamanlı kullanıcı çakışması (concurrency) senaryoları kapsam dışıdır.

1.4. Kapsanan İş Akışı

OrderFlow'un modellediği kurumsal süreç, aşağıdaki doğrusal zincir üzerine kurgulanmıştır:



Bu zincirin kritik noktası, teklifin onaylanmasıdır: onay anında sistem tek bir işlemle sipariş, fatura ve sevkiyat kaydını eşzamanlı olarak üretir. Böylece departmanlar arası veri girişi tekrarı ortadan kalkar.

2. MATERYAL VE YÖNTEM

2.1. Ön Uç (Frontend) Teknolojileri

2.1.1. React 19 Mimarisi

Proje, React kütüphanesinin en güncel sürümü olan **React 19** tabanlı bileşen mimarisiyle geliştirilmiştir. Fonksiyonel bileşenler, React Hook'ları (`useState`, `useEffect`, `useMemo`) ile yönetilerek sayfa içi durum güncellemelerinin minimum kaynak tüketimiyle anında DOM'a yansıtılması sağlanmıştır.

Mimaride benimsenen temel ilkeler:

- **Bileşen ayrıştırması:** Her ekran (view) kendi dosyasında tanımlanmış, ortak öğeler (modal, kart, rozet) tekrar kullanılabilir bileşenlere bölünmüştür.
- **Yukarı taşınmış durum (lifted state):** Paylaşılan veriler `App.tsx` içinde tutulur, alt bileşenlere prop olarak iletilir.
- **Türetilmiş veri için `useMemo`:** Filtreleme, arama ve toplam hesaplama gibi işlemler her render'da yeniden hesaplanmak yerine bağımlılıkları değiştiğinde yeniden hesaplanır.

```
// Arama ve rol filtresinin useMemo ile optimize edilmesi
const filteredOrders = useMemo(() => {
  return orders.filter((order) =>
    order.customerName.toLowerCase().includes(search.toLowerCase())
  );
}, [orders, search]);
```

2.1.2. Vite Derleme Motoru

Uygulama, geleneksel derleyicilere oranla çok daha hızlı modül sıcak yüklemesi (HMR) ve optimize edilmiş üretim derlemeleri sunan **Vite** mimarisi üzerine kurulmuştur. Bu sayede dosya değişiklikleri milisaniyeler seviyesinde tarayıcıda güncellenmektedir.

Vite'in tercih edilme gerekçeleri:

Kriter	Vite	Klasik Bundler (Webpack)
Geliştirme sunucusu açılışı	Anlık (ESM tabanlı)	Tüm paket derlenir, yavaş
Sıcak yükleme (HMR)	Milisaniye seviyesinde	Proje büyüdükçe yavaşlar
Üretim derlemesi	Rollup ile optimize	Yapılandırma yükü fazla
Yapılandırma	Sade <code>vite.config.ts</code>	Kapsamlı yapılandırma gerekir

2.1.3. TypeScript Kullanımı

Kod güvenliği ve sürdürülebilirlik amacıyla tüm veri modelleri, parametre tipleri ve fonksiyon imzaları **TypeScript** ile katı kurallara (Strict Mode) bağlı kalınarak tiplendirilmiştir. Bu sayede çalışma zamanı (runtime) hataları geliştirme aşamasında önlenmiştir.

Özellikle rol tanımlarının birleşim tipi (union type) olarak modellenmesi, yetkilendirme kodunda yazım hatası kaynaklı güvenlik açıklarını derleme zamanında engellemektedir:

```
export type Role = "admin" | "sales" | "accounting" | "warehouse";

export interface NavItem {
  id: string;
  label: string;
  allowedRoles: Role[]; // yalnızca geçerli roller yazılabilir
}
```

2.2. Arayüz ve Tasarım Standartları

2.2.1. Tailwind CSS v4 Entegrasyonu

CSS yazım yükünü azaltmak ve bileşen düzeyinde özelleştirilebilir tasarımlar oluşturmak için **Tailwind CSS v4** tercih edilmiştir. Bu sürümün getirdiği modern CSS tabanlı tema özelleştirme mekanizmaları (`@theme`, CSS değişkenleri) kullanılarak temiz ve performanslı bir stil dosyası oluşturulmuştur. Yardımcı sınıf (utility-first) yaklaşımı sayesinde ayrı bir CSS dosyası yönetme ihtiyacı ortadan kalkmış, stil kodu doğrudan bileşenin yanında konumlanmıştır.

2.2.2. Lucide React İkonografisi

Kullanıcı deneyimini güçlendirmek amacıyla uygulama genelinde minimalist ve vektörel bir simge seti sunan **Lucide React** entegre edilmiştir. Simgeler ağaç sarsma (tree-shaking) ile yalnızca kullanılanlar paketlendiğinden derleme boyutuna etkisi düşüktür; renk geçişleri ve animasyonlarla uyumlu şekilde konumlandırılmıştır.

2.2.3. Gece Modu (Dark Mode) ve Custom Variant Yapılandırması

Tailwind v4'ün seçici tabanlı karanlık mod desteğini devreye almak amacıyla `src/index.css` dosyasında aşağıdaki yapılandırma uygulanmıştır:

```
@import "tailwindcss";
@custom-variant dark (&:where(.dark, .dark *));
```

Bu sayede kullanıcının seçimine göre tüm ekran renkleri anında derin gece tonlarına (`#090d16`) uyarlanabilmektedir. Tema tercihi `localStorage` üzerinde saklandığından, sayfa yenilendiğinde kullanıcının seçimi korunur.

2.2.4. Premium Glassmorphism Tasarım İlkeleri

Uygulama arayüzünde modern tasarım akımlarından biri olan **Glassmorphism** (cam efekti) uygulanmıştır. Kullanılan yardımcı sınıflar ve amaçları:

Efekt	Tailwind Sınıfı	Amaç
Yarı geçirgen arka plan	<code>bg-white/80</code> , <code>bg-slate-900/80</code>	Katman derinliği hissi
Bulanıklaştırma filtresi	<code>backdrop-blur-md</code>	Cam yüzey etkisi
İnce kenarlık	<code>border-slate-200/50</code>	Kart sınırlarının yumuşak ayrımı
Yumuşak gölge	<code>shadow-lg</code>	Yükseklik (elevation) algısı

Bu kombinasyon ile kullanıcılara üst düzey bir kurumsal panel hissi verilmiştir.

2.3. Veri Yönetimi ve Kalıcılık

2.3.1. LocalStorage Serializasyonu

Uygulamada oluşturulan yeni müşteriler, teklifler, siparişler, ödemeler, kargo sevkiyatları ve sistem logları JSON formatına dönüştürülerek (serialize) tarayıcının yerel depolama alanına (`localStorage`) kaydedilir. Bu sayede sayfa yenilendiğinde veriler kaybolmaz.

```
// Durum değiştiğinde otomatik kalıcılık
useEffect(() => {
  localStorage.setItem("orderflow_orders", JSON.stringify(orders));
}, [orders]);

// İlk açılışta okuma; kayıt yoksa başlangıç verisine düşülür
const [orders, setOrders] = useState<Order[]>(() => {
  const saved = localStorage.getItem("orderflow_orders");
  return saved ? JSON.parse(saved) : initialOrders;
});
```

Kullanılan anahtarlar `orderflow_` ön eki ile isim çakışmasına karşı korunmuştur:

Anahtar	İçerik
orderflow_logged_in	Oturum durumu ("true" / yok)
orderflow_role	Aktif kullanıcı rolü
orderflow_customers	Müşteri kayıtları dizisi
orderflow_quotations	Teklif kayıtları dizisi
orderflow_orders	Sipariş kayıtları dizisi
orderflow_payments	Ödeme/fatura kayıtları dizisi
orderflow_deliveries	Sevkiyat kayıtları dizisi
orderflow_logs	Etkinlik günlüğü kayıtları
orderflow_theme	Tema tercihi (light / dark)

2.3.2. Durum Yönetimi (State Management)

Uygulama genelinde durumlar `App.tsx` dosyasında merkezi olarak yönetilir. Veri tablosu güncellemeleri, modal açılışları ve arama filtreleri alt bileşenlere React propları aracılığıyla aktararak **tek yönlü veri akışı** ve **tek doğruluk kaynağı (Single Source of Truth)** prensibine uyulmuştur. Harici bir durum yönetimi kütüphanesi (Redux, Zustand) kullanılmamış; proje ölçeğinde React'in yerleşik hook'larının yeterli olduğu değerlendirilmiştir.

2.3.3. Veri Modelleri

Sistemin temel varlıkları TypeScript arayüzleri (interface) ile tiplendirilmiştir:

```
export interface Customer {
  id: string;
  name: string;
  taxOffice: string;    // Vergi dairesi
  taxNumber: string;    // Vergi numarası
  email: string;
  phone: string;
  createdAt: string;
}

export interface Quotation {
  id: string;
  customerId: string;
  items: QuotationItem[];
  total: number;
  status: "draft" | "pending" | "approved" | "rejected" | "expired";
  validUntil: string;
}

export interface Order {
  id: string;
  quotationId: string;
  status: "new" | "preparing" | "shipped" | "delivered";
  priority: "low" | "medium" | "high";
}

export interface ActivityLog {
  id: string;
  actor: Role;          // İşlemi yapan rol
  action: string;       // Yapılan işlem
  entity: string;       // Etkilenen kayıt
  timestamp: string;    // ISO 8601
}
```

Varlıklar arasındaki ilişki, teklif kimliği (`quotationId`) ve müşteri kimliği (`customerId`) üzerinden kurulan yumuşak referanslarla sağlanır.

2.4. Proje Dizin Yapısı

```
orderflow/
├─ public/
├─ src/
│   ├─ components/
│   │   ├─ Sidebar.tsx
│   │   ├─ Header.tsx
│   │   ├─ CreateNewModal.tsx
│   │   └─ AlertModal.tsx
│   └─ views/
│       ├─ LoginView.tsx
│       ├─ DashboardView.tsx
│       ├─ CustomersView.tsx
│       ├─ QuotationsView.tsx
│       ├─ OrdersView.tsx
│       ├─ PaymentsView.tsx
│       ├─ DeliveriesView.tsx
│       ├─ CalendarView.tsx
│       ├─ ReportingView.tsx
│       └─ ActivityLogView.tsx
├─ types.ts          // Ortak TypeScript tipleri
├─ data.ts           // Başlangıç (seed) verileri
├─ App.tsx           // Merkezi durum ve yönlendirme
├─ main.tsx          // Giriş noktası
├─ index.css         // Tailwind v4 + dark variant
├─ index.html
├─ package.json
├─ tsconfig.json
└─ vite.config.ts
```

3. SİSTEM MİMARİSİ VE ROL TABANLI ERİŞİM DENETİMİ (RBAC)

3.1. RBAC Kavramı ve En Az Yetki İlkesi

Rol Tabanlı Erişim Denetimi (Role-Based Access Control), yetkilerin doğrudan kullanıcılara değil, kullanıcıların atandığı **rollere** tanımlandığı bir erişim denetimi modelidir. Kullanıcı → Rol → İzin şeklindeki bu dolaylı bağ sayesinde, bir departmanın yetkisi değiştiğinde tek tek kullanıcılar yerine yalnızca rol tanımı güncellenir.

OrderFlow'da RBAC üç katmanda uygulanmıştır:

- Kimlik doğrulama (authentication):** Kullanıcının hangi rol olduğunu belirlenmesi (`LoginView`).
- Yetkilendirme (authorization):** Rolün hangi modülleri görebileceğinin belirlenmesi (`Sidebar` menü filtresi + `App.tsx` görünüm yönlendirmesi).
- Hesap verebilirlik (accountability):** Yapılan işlemin rol bilgisiyle günlüğe yazılması (`ActivityLogView`).

Bu üç katman birlikte, bilişim güvenliğinin temel ilkelerinden **en az yetki (least privilege)** ve **görevler ayrılığı (separation of duties)** ilkelerini uygular. Örneğin satış temsilcisinin ödeme tahsil edememesi, muhasebenin sevkiyat kaydını değiştirememesi bilinçli bir görev ayrılığı tasarımıdır.

3.2. Rol Matrisi ve Erişim Kısıtlamaları

Sistem güvenliği ve departman kısıtlamaları çerçevesinde rollerin görebileceği sayfalar aşağıdaki matris ile sınırlandırılmıştır:

Modül / Ekran	Yönetici (Admin)	Satış (Sales)	Muhasebe (Accounting)	Depo (Warehouse)
Kontrol Paneli (Dashboard)	Evet	Evet	Evet	Evet
Müşteriler (Customers)	Evet	Evet	Evet	Hayır
Teklifler (Quotations)	Evet	Evet	Hayır	Hayır
Siparişler (Orders)	Evet	Evet	Evet	Evet
Ödemeler (Payments)	Evet	Hayır	Evet	Hayır
Teslimatlar (Deliveries)	Evet	Hayır	Hayır	Evet
Takvim (Calendar)	Evet	Evet	Evet	Hayır
Raporlar (Reports)	Evet	Hayır	Evet	Hayır
Etkinlik Günlüğü (Audit Logs)	Evet	Hayır	Hayır	Hayır

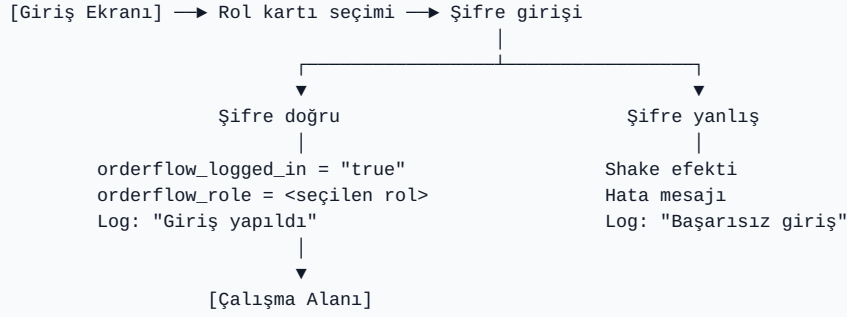
Matrisin tasarım gerekçeleri:

- Etkinlik günlüğü yalnızca Yönetici'de:** Denetim izinin, denetlenen kişi tarafından görülememesi denetim bağımsızlığı açısından tercih edilmiştir.
- Teklifler Muhasebe'ye kapalı:** Muhasebe, ancak teklif onaylanıp faturaya dönüştüğünde sürece dahil olur; taslak aşamasındaki fiyat pazarlığı bilgisine ihtiyacı yoktur.
- Müşteriler Depo'ya kapalı:** Depo yalnızca sevkiyat adresine ihtiyaç duyar; müşterinin vergi/cari bilgilerine erişimi gereksizdir.
- Siparişler tüm rollerde:** Sipariş, dört departmanın da ortak referans noktasıdır; ancak roller sipariş üzerinde farklı işlemler yapabilir.

3.3. Giriş Kimlik Doğrulama Süreci

Kullanıcı giriş ekranına ulaştığında 4 rol kartından birini seçmek zorundadır. Rol kartı seçildikten sonra ilgili rol için tanımlanmış statik şifre istenir. Şifre doğrulama işlemi tamamen tarayıcı tarafında yapılır; başarılı olması durumunda `localStorage` alanına `orderflow_logged_in: "true"` anahtarı yazılarak çalışma alanına geçiş hakkı tanınır. Yanlış şifre girişlerinde sarsıntı (shake) efekti ile kullanıcı uyarılır.

Akış şu şekildedir:



3.4. Menü Filtreleme ve Yetki Doğrulama Katmanı

Sidebar bileşeni, menü öğelerini aktif rolün yetki listesine göre dinamik olarak süzer. Böylece kullanıcı, erişemeyeceği bir modülün varlığını dahi görmez (bilgi sızıntısının önlenmesi):

```

const navItems: NavItem[] = [
  { id: "dashboard", label: "Kontrol Paneli", allowedRoles: ["admin", "sales", "accounting", "warehouse"] },
  { id: "customers", label: "Müşteriler", allowedRoles: ["admin", "sales", "accounting"] },
  { id: "quotations", label: "Teklifler", allowedRoles: ["admin", "sales"] },
  { id: "payments", label: "Ödemeler", allowedRoles: ["admin", "accounting"] },
  { id: "deliveries", label: "Teslimatlar", allowedRoles: ["admin", "warehouse"] },
  { id: "logs", label: "Etkinlik Günlüğü", allowedRoles: ["admin"] },
];

const visibleItems = navItems.filter((item) =>
  item.allowedRoles.includes(currentRole)
);

```

Menü gizlemenin tek başına yeterli olmadığı gerçeğinden hareketle, App.tsx içindeki görünüm yönlendiricisi de ikinci bir denetim uygulayarak aktif sekme kullanıcının yetkisinde değilse, kullanıcı otomatik olarak kontrol paneline döndürülür. Bu **derinlemesine savunma (defense in depth)** yaklaşımının basit bir uygulamasıdır.

3.5. Denetim İzi (Audit Trail) Mimarisi

Sistemdeki her durum değiştiren işlem (giriş/çıkış, kayıt ekleme, durum güncelleme, tahsilat) merkezi bir günlükleme fonksiyonundan geçirilir:

```

const addLog = (action: string, entity: string) => {
  const entry: ActivityLog = {
    id: crypto.randomUUID(),
    actor: currentRole,
    action,
    entity,
    timestamp: new Date().toISOString(),
  };
  setLogs((prev) => [entry, ...prev]);
};

```

Günlük kayıtları yalnızca eklenebilir (append-only) biçimde tutulur; arayüz üzerinden silme veya düzenleme işlevi bilinçli olarak sunulmamıştır. Bu, denetim izinin bütünlüğünü (integrity) korumaya yönelik bir tasarım kararıdır.

3.6. Güvenlik Değerlendirmesi ve İstemci Tarafı Yetkilendirmenin Sınırları

Projenin akademik dürüstlüğü açısından, uygulanan güvenlik modelinin sınırlarının açıkça belirtilmesi gerekmektedir:

Uygulanan Önlem	Sağladığı Koruma	Sınırı
Rol bazlı menü filtreleme	Bilgi kirliliğini ve kazara erişimi önler	Tarayıcı geliştirici araçlarıyla aşılabılır
Statik şifre doğrulama	Rol ayırımını simüle eder	Şifreler istemci kodunda; hash'lenmemiş
localStorage oturum anahtarı	Sayfa yenilemede oturumu korur	Kullanıcı tarafından elle değiştirilebilir
Rol değişiminde otomatik çıkış	Oturum karışmasını engeller	Sunucu tarafı oturum sonlandırma yoktur
Append-only denetim izi	İşlem geçmişi korur	Aynı tarayıcıda localStorage temizlenebilir

Temel çıkarım: İstemci tarafında yapılan yetkilendirme bir **kullanılabilirlik ve düzen** önlemidir, bir **güvenlik sınırı** değildir. Gerçek bir güvenlik sınırı ancak sunucu tarafında, her istek için tekrar doğrulanan yetki denetimiyle (JWT/OAuth + sunucu taraflı RBAC) kurulabilir. Bu tespit, Bölüm 7'de önerilen gelecek çalışmaların da temel gerekçesidir.

4. UYGULAMA BİLEŞENLERİ VE DETAYLI MODÜL ANALİZİ

4.1. Ana Uygulama Modülü (App .tsx)

Uygulamanın kalbini oluşturur. Sistem durumlarını (states) barındırır, `localStorage` okuma/yazma işlemlerini koordine eder ve aktif sekmeye göre ilgili alt görünümü çağırır. Ayrıca sistem genelinde çalışan özel tasarlanmış **Alert Modal** sistemini barındırarak tarayıcının standart `alert()` pencerelerini engeller; böylece tüm bildirimler uygulama tasarım diliyle tutarlı hale gelir.

Sorumlulukları:

- Oturum ve rol durumunun yönetimi
- Tüm varlık listelerinin (müşteri, teklif, sipariş, ödeme, sevkiyat, log) tutulması
- Kalıcılık `useEffect` 'lerinin çalıştırılması
- Aktif sekmeye göre görünüm yönlendirmesi ve yetki doğrulaması
- Ortak modal ve bildirim altyapısı

4.2. Kimlik Doğrulama Ekranı (LoginView .tsx)

Aydınlık ve karanlık mod seçimi sunan, 4 departman üyesinin fotoğrafları ve unvanlarıyla listelendiği interaktif giriş arayüzüdür. Şifre gizleme/gösterme özelliği ve şifre ipuçları tamamen Türkçe olarak tasarlanmıştır. Hatalı girişte CSS keyframe animasyonu ile sarsıntı efekti tetiklenir ve başarısız deneme etkinlik günlüğüne yazılır.

4.3. Navigasyon ve Kontrol Panelleri (Sidebar .tsx , Header .tsx)

- **Sidebar:** Kullanıcının rolüne göre menü öğelerini dinamik olarak filtreler (bkz. Bölüm 3.4). Üst kısmında "Yeni Ekle" hızlı işlem butonu, alt kısmında **Ayarlar** ve **Destek** sekmeleri yer alır; aktif sekme durumu görsel olarak vurgulanır.
- **Header:** Evrensel arama çubuğunu barındırır. Rol değiştirme dropdown menüsü, gelen sistem bildirimleri kutusu ve profil kartını içerir. Başka bir role geçildiğinde güvenlik gereği otomatik çıkış (logout) tetiklenir ve kullanıcı yeniden şifre girmek zorunda kalır.

4.4. Kontrol Paneli (DashboardView .tsx)

Kurumsal performansın anlık özetidir. Toplam müşteri, aktif sipariş, bekleyen teklif, tahsil edilecek tutar ve aktif sevkiyat sayılarını gösteren interaktif kartları içerir. **Gelir Akışı Grafiği** ile son 6 aylık veya 1 yıllık mali durumu gösterir.

Teknik not: Grafik barları, Tailwind CSS v4'ün kullanılmayan sınıfları derlemeden çıkarma (purge/tree-shaking) davranışından etkilenmemesi için dinamik sınıf adı yerine satır içi (inline) yükseklik değerleriyle çizilmiştir. Bunun nedeni, `h-[${value}px]` gibi çalışma zamanında üretilen sınıf adlarının Tailwind tarafından statik analizde tespit edilememesidir.

4.5. Müşteri Yönetimi (CustomersView .tsx)

Müşteri profillerinin listelendiği, arandığı ve yönetildiği sayfadır. Her müşterinin vergi dairesi, vergi numarası gibi kurumsal bilgileri yan çekmecede (detail drawer) gösterilir. Müşteri silme işlemleri için özel tasarlanmış bir silme onay modali tetiklenir; böylece geri döndürülemez işlemlerde kazara tıklama riski azaltılır.

4.6. Teklif Yönetimi (QuotationsView .tsx)

Tekliflerin **Taslak**, **Onay Bekliyor**, **Onaylandı**, **Reddedildi** ve **Süresi Dolan** durumlarına göre sekmeli olarak filtrelendiği alandır. Modülün en kritik işlevi, teklif onaylandığında tek tuşla üç kaydın otomatik olarak üretilmesidir:

1. **Aktif müşteri siparişi** (Depo modülüne düşer)
2. **Cari ödeme faturası** (Muhasebe modülüne düşer)
3. **Lojistik kargo kaydı** (Teslimat modülüne düşer)

Bu dönüşüm, iş akışındaki departmanlar arası veri girişi tekrarını ortadan kaldırır ve her üç kaydın aynı teklif kimliğine bağlanmasını garanti eder. Ayrıca modül, profesyonel PDF şablonunda fatura çıktısı alma desteği sunar.

4.7. Sipariş Yönetimi (OrdersView .tsx)

Siparişlerin **liste** veya **Kanban kart** görünümüyle takip edildiği modüldür. Kanban panolarında sipariş durumları (Yeni, Hazırlanıyor, Sevk Edildi, Teslim Edildi) sütunlar halinde gösterilir; kartlar öncelik derecelerine (Düşük, Orta, Yüksek) göre renkli rozetlerle etiketlenir. Görünüm tercihi kullanıcı bazında korunur.

4.8. Finans ve Ödeme Takibi (PaymentsView .tsx)

Toplam ciro, tahsil edilen tutar ve kalan alacak bakiyelerini gösteren finansal özet modülüdür. Tabloda ödeme yöntemleri (Havale, Kredi Kartı, Nakit) gösterilir. "Ödemeyi Tahsil Et / Kaydet" butonuna tıklandığında açılan formda bakiye tahsilatı anında gerçekleştirilir, ilgili sipariş ve müşteri cari durumu güncellenir ve işlem etkinlik günlüğüne yazılır.

4.9. Lojistik ve Teslimat Takibi (DeliveriesView.tsx)

Dağıtımdaki kargo araçlarının telemetri durumlarının harita üzerinden sinyal vericilerle takip edildiği lojistik ekranıdır. Enlem ve boylam koordinatları JavaScript setInterval fonksiyonu ile düzenli aralıklarla küçük rastgele sapmalarla titreştirilerek canlı takip hissi pekiştirilmiştir. "Sonraki Durak Aşamasını Simüle Et" butonu ile paketlerin teslimat aşamaları (Depoda → Yolda → Dağıtımda → Teslim Edildi) sırayla ilerletilir.

Not: Bileşen kaldırıldığında (unmount) setInterval temizlenerek bellek sızıntısı (memory leak) önlenmiştir:

```
useEffect(() => {  
  const timer = setInterval(() => jitterCoordinates(), 2000);  
  return () => clearInterval(timer); // temizlik fonksiyonu  
}, []);
```

4.10. Operasyonel Takvim (CalendarView.tsx)

Ocak – Aralık ayları arasındaki tüm teslimat vadelerini, fatura ödeme günlerini, sipariş tarihleri ve teklif son günlerini tek bir takvim matrisi üzerinde listeleyen operasyonel zaman çizelgesidir. Seçilen güne tıklandığında o güne ait görevler yan panelde listelenir; her görev türü farklı renk kodu ile ayrıştırılmıştır.

4.11. Analizler ve Raporlama (ReportingView.tsx)

Şirketin en çok harcama yapan müşterilerini, en çok ciro getiren ürünlerini ve aylık sipariş hacimlerini karşılaştıran detaylı raporlama ve veri analiz ekranıdır. Hesaplamalar useMemo ile türetilmiş veri olarak üretilir; ayrıca rapor tablolarının CSV formatında dışa aktarılması desteklenir.

4.12. Sistem Etkinlik Günlüğü (ActivityLogView.tsx)

Sistem üzerinde yapılan tüm işlemlerin (kullanıcı giriş/çıkışları, yeni veri ekleme, durum güncellemeleri, fatura tahsilatları) **kim tarafından** ve **ne zaman** yapıldığını gösteren, filtreleme ve arama desteği sunan güvenlik denetim (audit trail) günlüğüdür. Yalnızca Yönetici rolüne açıktır ve kayıtlar ters kronolojik sırada listelenir.

4.13. Ortak Ekleme Modalı (CreateNewModal.tsx)

Sidebar üzerindeki "Yeni Ekle" butonuna basıldığında açılan; yeni müşteri, yeni teklif, yeni sipariş, yeni ödeme ve sevk kayıtlarını tek bir form yapısından oluşturabilen ortak bileşendir. Form alanları seçilen kayıt türüne göre dinamik olarak değişir. Zorunlu alan doğrulamaları tamamen Türkçe hata mesajları ile denetlenir; geçersiz form gönderimi engellenir.

5. KURULUM VE ÇALIŞTIRMA TALİMATLARI

5.1. Gereksinimler

- Bilgisayarınızda **Node.js** (Sürüm 18 veya üzeri) kurulu olmalıdır.
- npm** paket yöneticisi (Node.js ile birlikte gelir).
- Komut satırı (Terminal / PowerShell) erişimi.
- Modern bir tarayıcı (Chrome, Edge, Firefox güncel sürüm).

5.2. Projenin Çalıştırılması

- Proje klasörünü bilgisayarınıza indirin veya klonlayın.
- Terminal üzerinden proje klasörünün içine girin.
- Gerekli kütüphaneleri kurmak için aşağıdaki komutu çalıştırın:

```
npm install
```

- Geliştirme sunucusunu başlatmak için:

```
npm run dev
```

- Tarayıcınızda `http://localhost:3000` adresini açarak uygulamayı test edebilirsiniz.

5.3. Test Kullanıcı Bilgileri

Sistemdeki rollere giriş yapmak için aşağıdaki kimlik bilgilerini kullanabilirsiniz:

Rol	Kullanıcı Kartı	Unvan	Şifre
Yönetici (Admin)	Alex Rivera	Ops Manager	admin
Satış (Sales)	Sarah Jenkins	Sales Executive	sales
Muhasebe (Accounting)	Emma Stone	Financial Controller	accounting
Depo (Warehouse)	Marcus Thorne	Logistics Manager	warehouse

Uyarı: Bu şifreler yalnızca demo amaçlıdır ve istemci kodunda açık biçimde bulunmaktadır. Üretim ortamında kullanılmamalıdır.

5.4. Projenin Derlenmesi (Build)

Uygulamayı optimize edilmiş statik HTML, CSS ve JS paketleri halinde derlemek ve canlı sunucuya aktarmak için:

```
npm run build
```

Derleme sonucunda oluşan dosyalar projenin ana dizinindeki `/dist` klasöründe yer alacaktır. Derlenmiş sürümü yerel olarak önizlemek için:

```
npm run preview
```

5.5. Sık Karşılaşılan Sorunlar

Sorun	Olası Neden	Çözüm
Port 3000 kullanımında hatası	Başka bir uygulama portu tutuyor	<code>vite.config.ts</code> içinde port değeri değiştirilir
Karanlık mod çalışmıyor	<code>@custom-variant</code> satırı eksik	<code>src/index.css</code> yapılandırması kontrol edilir
Veriler kayboluyor	Tarayıcı verisi temizlenmiş	<code>localStorage</code> temizlenince başlangıç verisine dönülür
Stil sınıfları uygulanmıyor	Dinamik sınıf adı üretimi	Satır içi stil veya tam sınıf adı kullanılır

6. TEST VE DOĞRULAMA

6.1. Rol Tabanlı Erişim Test Senaryoları

#	Senaryo	Beklenen Sonuç	Durum
T1	Satış rolü ile giriş yapılır	Ödemeler ve Teslimatlar menüde görünmez	Başarılı
T2	Depo rolü ile giriş yapılır	Müşteriler, Teklifler, Raporlar görünmez	Başarılı
T3	Muhasebe rolü ile giriş yapılır	Etkinlik Günlüğü görünmez	Başarılı
T4	Yönetici rolü ile giriş yapılır	Tüm modüller erişilebilir	Başarılı
T5	Header üzerinden rol değiştirilir	Otomatik çıkış tetiklenir, şifre istenir	Başarılı
T6	Yanlış şifre girilir	Shake efekti, girişe izin verilmez	Başarılı

6.2. İş Akışı Test Senaryoları

#	Senaryo	Beklenen Sonuç	Durum
T7	Teklif onaylanır	Sipariş, fatura ve sevkiyat kaydı otomatik oluşur	Başarılı
T8	Ödeme tahsil edilir	Kalan bakiye güncellenir, log kaydı düşer	Başarılı
T9	Sevkiyat aşaması ileletilir	Durum bir sonraki aşamaya geçer	Başarılı
T10	Yeni müşteri eklenir	Listeye eklenir, günlüğe yazılır	Başarılı
T11	Zorunlu alan boş bırakılır	Türkçe hata mesajı gösterilir, kayıt engellenir	Başarılı

6.3. Arayüz ve Kalıcılık Testleri

#	Senaryo	Beklenen Sonuç	Durum
T12	Sayfa yenilenir (F5)	Veriler ve oturum korunur	Başarılı
T13	Karanlık mod açılır	Tüm ekranlar gece tonlarına geçer, tercih saklanır	Başarılı
T14	Mobil genişlikte görüntülenir	Sidebar daralır, tablolar kaydırılabilir	Başarılı
T15	Etkinlik günlüğü filtrelenir	Yalnızca eşleşen kayıtlar listelenir	Başarılı

7. SONUÇ VE GELECEK PLANLAR

Bu bitirme çalışmasında, modern web teknolojileri (React 19, Vite, Tailwind CSS v4) kullanılarak şık, responsive ve kararlı çalışan rol tabanlı bir ERP arayüzü başarıyla hayata geçirilmiştir. Projede yerel tarayıcı depolama alanları verimli kullanılmış, karanlık mod ve Türkçe dil desteği sorunsuz çalışacak şekilde yapılandırılmıştır.

Bilişim Güvenliği Mimarisi dersi kapsamında elde edilen temel kazanımlar:

1. **RBAC modelinin somutlaştırılması:** Kullanıcı → Rol → İzin bağının, merkezi bir yetki listesi üzerinden yönetilmesinin sürdürülebilirlik avantajı gözlemlenmiştir.
2. **En az yetki ve görevler ayrılığı:** Rol matrisinin tasarımı, güvenliğin yalnızca teknik değil aynı zamanda organizasyonel bir tasarım kararı olduğunu göstermiştir.
3. **Hesap verebilirlik:** Append-only denetim izinin, güvenlik olaylarının sonradan incelenebilmesi (forensics) için kritik olduğu anlaşılmıştır.
4. **İstemci taraflı denetimin sınırları:** Ön uçta yapılan yetkilendirmenin bir kullanılabilirlik önlemi olduğu, gerçek güvenlik sınırının sunucuda kurulması gerektiği tespit edilmiştir (bkz. Bölüm 3.6).

Gelecekte Eklenebilecek Özellikler:

- **RESTful API ve Veritabanı Entegrasyonu:** Tarayıcı `localStorage`'ı yerine PostgreSQL veya MongoDB tabanlı bir veri tabanı ile gerçek zamanlı ve çok kullanıcıli veri kaydı.
 - **JWT ve OAuth Güvenlik Katmanı:** Gerçek şifreleme/hash algoritmaları (bcrypt, Argon2) kullanan oturum açma, token doğrulama ve sunucu taraflı yetki denetimi.
 - **Gelişmiş Raporlama İhracatı:** Raporların yalnızca CSV değil, PDF ve Excel (XLSX) formatlarında da dışa aktarılabilmesi.
 - **Gerçek Telemetri Entegrasyonu:** Simüle koordinatlar yerine bir harita servisi ve araç takip API'si ile canlı konum verisi.
 - **Otomatik Test Altyapısı:** Vitest ve React Testing Library ile birim testleri, Playwright ile uçtan uca rol erişim testleri.
 - **Çok Dilli Destek (i18n):** Türkçe/İngilizce arayüz desteği.
-

8. KAYNAKÇA

- [1] React 19 Resmi Dokümantasyon Portalı: <https://react.dev>
- [2] Vite.js Derleme Motoru ve Geliştirici Kılavuzu: <https://vite.dev>
- [3] Tailwind CSS v4 Yeni Özellikler ve Tema Yönetimi: <https://tailwindcss.com>
- [4] Lucide İkon Seti ve Kullanım Yöntemleri: <https://lucide.dev>
- [5] Web Depolama API Kılavuzu (MDN Web Docs): <https://developer.mozilla.org>
- [6] TypeScript Resmi Dokümantasyonu: <https://www.typescriptlang.org/docs>
- [7] NIST, "Role Based Access Control (RBAC)" – Computer Security Resource Center: <https://csrc.nist.gov/projects/role-based-access-control>
- [8] OWASP Authorization Cheat Sheet: <https://cheatsheetseries.owasp.org>