

Satış Takip CRM Sistemi - Detaylı Proje Dokümantasyonu







İindekiler

1. [Proje Genel Bakış](#)
 2. [Teknoloji Stack](#)
 3. [Proje Yapısı](#)
 4. [Temel Bileşenler](#)
 5. [Sayfalar](#)
 6. [Yardımcı Dosyalar](#)
 7. [Güvenlik ve Yetkilendirme](#)
 8. [Veritabanı Yapısı](#)
 9. [Özellikler](#)
 10. [Kurulum ve Çalıştırma](#)
-

Proje Genel Bakış

Satış Takip CRM Sistemi, satış ekiplerinin müşteri ilişkilerini yönetmek, satış performanslarını takip etmek ve veri analizi yapmak için geliştirilmiş modern bir web uygulamasıdır.

Ana Hedefler:

-  Satış verilerinin merkezi yönetimi
 -  Kullanıcı rolü tabanlı erişim kontrolü
 -  Gerçek zamanlı performans analizi
 -  Ekip içi iletişim sistemi
 -  Mobil uyumlu responsive tasarım
 -  Güvenli veri saklama ve erişim
-

Teknoloji Stack

Frontend

- React 19.1.0 - Modern UI kütüphanesi
- Vite 6.3.5 - Hızlı geliştirme ortamı
- React Router DOM 7.6.2 - Sayfa yönlendirme

- Tailwind CSS 3.4.7 - Utility-first CSS framework
- Lucide React 0.516.0 - Modern icon seti

Backend & Database

- Firebase 11.9.1 - Backend as a Service
 - o Authentication - Kullanıcı kimlik doğrulama
 - o Firestore - NoSQL veritabanı
 - o Hosting - Web barındırma

Utility Libraries

- XLSX 0.18.5 - Excel dosya işlemleri
- jsPDF 3.0.1 - PDF oluşturma
- jsPDF-AutoTable 5.0.2 - PDF tablo oluşturma

Development Tools

- ESLint 9.25.0 - Kod kalitesi kontrolü
- PostCSS 8.5.6 - CSS işleme
- Autoprefixer 10.4.21 - CSS uyumluluk

Proje Yapısı

```
satis-crm/
├── public/
│   ├── 404.html
│   └── vite.svg
├── src/
│   ├── App.jsx
│   ├── main.jsx
│   ├── index.css
│   ├── App.css
│   ├── auth/
│   │   └── firebaseConfig.js
│   ├── components/
│   │   ├── ChatSystem.jsx
│   │   ├── DataUpdate.jsx
│   │   ├── DropdownSettings.jsx
│   │   ├── Modal.jsx
│   │   ├── Navbar.jsx
│   │   ├── RecordForm.jsx
│   │   ├── RecordTable.jsx
│   │   ├── RoleGuard.jsx
│   │   ├── Sidebar.jsx
│   │   ├── TargetManagement.jsx
│   │   └── TeamManagement.jsx
│   └── ...
└── ...
```

Statik dosyalar
404 hata sayfası
Vite logosu
Kaynak kodlar
Ana uygulama bileşeni
Uygulama giriş noktası
Global CSS stilleri
Uygulama özel stilleri
Kimlik doğrulama
Firebase yapılandırması
Yeniden kullanılabilir bileşenler
Anlık mesajlaşma sistemi
Veri güncelleme aracı
Dropdown ayarları
Modal bileşeni
Üst navigasyon
Kayıt ekleme formu
Kayıt tablosu
Rol tabanlı erişim kontrolü
Yan menü
Hedef yönetimi
Ekip yönetimi

└─	UserManagement.jsx	# Kullanıcı yönetimi
└─	context/	# React Context API
└─	AuthContext.jsx	# Kimlik doğrulama context
└─	DarkModeContext.jsx	# Karanlık mod context
└─	pages/	# Sayfa bileşenleri
└─	Analytics.jsx	# Analitik sayfası
└─	Dashboard.jsx	# Ana sayfa
└─	DataImport.jsx	# Veri içe aktarma
└─	DataUpdate.jsx	# Veri güncelleme
└─	DuplicateNumbers.jsx	# Tekrar eden numaralar
└─	Login.jsx	# Giriş sayfası
└─	MonthlyComparison.jsx	# Aylık karşılaştırma
└─	PersonnelDevelopment.jsx	# Personel gelişimi
└─	SystemLogs.jsx	# Sistem logları
└─	SystemSettings.jsx	# Sistem ayarları
└─	TeamPerformance.jsx	# Ekip performansı
└─	UserSwitcher.jsx	# Kullanıcı değiştirici
└─	utils/	# Yardımcı fonksiyonlar
└─	helpers.js	# Genel yardımcı fonksiyonlar
└─	logger.js	# Sistem log yöneticisi
└─	package.json	# Proje bağımlılıkları
└─	vite.config.js	# Vite yapılandırması
└─	tailwind.config.js	# Tailwind CSS yapılandırması
└─	postcss.config.js	# PostCSS yapılandırması
└─	eslint.config.js	# ESLint yapılandırması
└─	firestore.rules	# Firestore güvenlik kuralları

Temel Bileşenler

1. App.jsx - Ana Uygulama Bileşeni

```
// Ana uygulama yapısı ve routing
- AuthProvider ve DarkModeProvider ile sarmalama
- ProtectedRoute ile korumalı sayfalar
- MainLayout ile tutarlı sayfa düzeni
- HashRouter ile GitHub Pages uyumluluğu
```

Amaç: Uygulamanın ana iskeletini oluşturur, tüm sayfaları ve bileşenleri organize eder.

Temel işlevler:

- Kullanıcı kimlik doğrulama kontrolü
- Sayfa yönlendirme yönetimi
- Global state yönetimi
- Layout yapısı sağlama

2. Navbar.jsx - Üst Navigasyon

```
// Özellikler:
- Responsive tasarım (mobil/desktop)
- Kullanıcı profil menüsü
```

- Karanlık mod toggle
- Mobil hamburger menü
- Çıkış yapma fonksiyonu

Amaç: Kullanıcının uygulamada gezinmesini sağlar ve temel işlemlere hızlı erişim sunar.

Kod Yapısı:

- `useState` ile menü durumu yönetimi
- `useAuth` ile kullanıcı bilgisi alma
- Responsive design için Tailwind CSS sınıfları
- Mobile-first yaklaşımı

3. Sidebar.jsx - Yan Menü

```
// Özellikler:  
- Rol tabanlı menü görünürlüğü  
- Aktif sayfa vurgulama  
- Mobil responsive davranış  
- Icon'lu menü öğeleri  
- Daraltılabilir yapı
```

Amaç: Ana navigasyon menüsü olarak çalışır, kullanıcı rolüne göre uygun sayfaları gösterir.

Rol **Tabanlı** Menü Yapısı:

- Admin: Tüm menü öğelerine erişim
- Team Leader: Yönetim ve raporlama sayfaları
- Personnel: Temel işlem sayfaları

4. RecordTable.jsx - Kayıt Tablosu

```
// Özellikler:  
- Sayfalama (pagination)  
- Gelişmiş filtreleme  
- Sıralama işlemleri  
- Kayıt düzenleme/silme  
- Excel export  
- Responsive tablo tasarımı  
- Rol tabanlı veri görünürlüğü
```

Amaç: Satış kayıtlarını tablo formatında gösterir, filtreleme ve düzenleme imkanı sunar.

Teknik Detaylar:

- Firestore real-time listener kullanımı
- Client-side pagination implementasyonu
- Portal modal kullanımı
- Responsive tablo tasarımı

5. RecordForm.jsx - Kayıt Ekleme Formu

```
// Özellikler:  
- Dinamik dropdown menüler  
- Form validasyonu  
- Telefon numarası formatlaması  
- Otomatik tarih ekleme  
- Başarı/hata mesajları
```

Amaç: Yeni satış kayıtlarının sisteme eklenmesini sağlar.

Form Yapısı:

- Controlled components kullanımı
- Real-time validation
- Dropdown seçenekleri Firestore'dan çekme
- Responsive form layout

6. ChatSystem.jsx - Anlık Mesajlaşma

```
// Özellikler:  
- Gerçek zamanlı mesajlaşma  
- Kullanıcı arama  
- Okunmamış mesaj sayacı  
- Online/offline durumu  
- Minimize/maximize özelliği
```

Amaç: Ekip üyeleri arasında anlık iletişim kurulmasını sağlar.

Teknik Özellikler:

- Firestore real-time listeners
- Message threading
- User presence tracking
- Responsive chat interface

7. UserManagement.jsx - Kullanıcı Yönetimi

```
// Özellikler:  
- Kullanıcı ekleme/düzenleme/silme  
- Rol atama (admin/teamLeader/personnel)  
- Şifre sıfırlama  
- Kullanıcı durumu yönetimi  
- Portal modal kullanımı
```

Amaç: Admin kullanıcıların sistem kullanıcılarını yönetmesini sağlar.

Güvenlik Özellikleri:

- Firebase Authentication entegrasyonu
- Rol tabanlı erişim kontrolü

- Güvenli şifre yönetimi

8. RoleGuard.jsx - Rol Tabanlı Erişim Kontrolü

```
// Özellikler:  
- Basit ve etkili rol kontrolü  
- Fallback içerik desteği  
- Context API entegrasyonu
```

Amaç: Belirli bileşenlerin sadece yetkili kullanıcılar tarafından görülmesini sağlar.

Kullanım Örneği:

```
<RoleGuard allowedRoles={['admin', 'teamLeader']}>  
  <AdminPanel />  
</RoleGuard>
```

Sayfalar

1. Dashboard.jsx - Ana Sayfa

```
// Özellikler:  
- Günlük istatistikler  
- Performans kartları  
- Hızlı eylem butonları  
- En iyi performans gösterenleri  
- Hedef takibi  
- Responsive grid layout
```

Amaç: Kullanıcıya genel sistem durumu ve kişisel performansı hakkında özet bilgi verir.

Veri Yapısı:

- Günlük satış sayıları
- Başarı oranları
- Hedef karşılaştırmaları
- Performans grafikleri

2. Analytics.jsx - Analitik Sayfası

```
// Özellikler:  
- Detaylı satış analizleri  
- Grafik ve çizelgeler  
- Tarih aralığı filtreleme  
- Kanal bazlı analiz  
- Personel performans karşılaştırması  
- PDF/Excel export
```

Amaç: Satış verilerinin detaylı analiz edilmesini ve raporlanmasını sağlar.

Analiz Türleri:

- Zaman bazlı analiz

- Kanal performansı
- Personel karşılaştırması
- Başarı oranları

3. Login.jsx - Giriş Sayfası

```
// Özellikler:  
- E-posta/şifre ile giriş  
- Şifre görünürlük toggle  
- Hata mesajları  
- Responsive form tasarımı  
- Güvenli kimlik doğrulama
```

Amaç: Kullanıcıların sisteme güvenli giriş yapmasını sağlar.

Güvenlik Özellikleri:

- Firebase Authentication
- Input validation
- Error handling
- Session management

4. DataImport.jsx - Veri İçe Aktarma

```
// Özellikler:  
- Excel dosyası yükleme  
- Veri önizleme  
- Hata kontrolü  
- Toplu veri ekleme  
- İlerleme çubuğu
```

Amaç: Mevcut Excel verilerinin sisteme toplu olarak aktarılmasını sağlar.

İşlem Adımları:

1. Excel dosyası seçimi
2. Veri doğrulama
3. Önizleme
4. Toplu ekleme

5. SystemLogs.jsx - Sistem Logları

```
// Özellikler:  
- Sistem aktivite logları  
- Filtreleme ve arama  
- Log seviyesi görünümü  
- Güvenlik logları  
- Export işlemleri
```

Amaç: Sistem aktivitelerinin izlenmesi ve güvenlik denetimi için log kayıtlarını gösterir.

Log Kategorileri:

- Authentication logs
- Security events
- System errors
- User activities

Yardımcı Dosyalar

1. **firebaseConfig.js** - Firebase Yapılandırması

```
// İçerik:  
- Firebase proje ayarları  
- Authentication servisi  
- Firestore veritabanı  
- Güvenlik yapılandırması
```

Yapılandırma Detayları:

```
const firebaseConfig = {  
  apiKey: "AIzaSyB430qFebSz7JqGmSk4ybzQouhkwKLa0o",  
  authDomain: "sati-staki-p-bc5c4.firebaseio.com",  
  projectId: "sati-staki-p-bc5c4",  
  storageBucket: "sati-staki-p-bc5c4.firebaseio.com",  
  messagingSenderId: "931339446084",  
  appId: "1:931339446084:web:d934b0bee4311198ecd911"  
};
```

2. **AuthContext.jsx** - Kimlik Doğrulama Context

```
// Özellikler:  
- Kullanıcı durumu yönetimi  
- Oturum zaman aşımı kontrolü  
- Rol tabanlı yetkilendirme  
- Otomatik çıkış yapma
```

Context **Fonksiyonları:**

- **login()** - Kullanıcı girişi
- **logout()** - Çıkış yapma
- **checkUserStatus()** - Durum kontrolü
- **updateLastActivity()** - Aktivite güncelleme

3. **DarkModeContext.jsx** - Karanlık Mod Context

```
// Özellikler:  
- Tema durumu yönetimi  
- Local storage entegrasyonu  
- Sistem teması algılama
```


4. helpers.js - Yardımcı Fonksiyonlar

```
// Fonksiyonlar:  
- formatDate() - Tarih formatlama  
- formatPhoneNumber() - Telefon formatlama  
- validateEmail() - E-posta doğrulama  
- sanitizeInput() - Girdi temizleme
```

Örnek **Kullanım**:

```
import { formatDate, formatPhoneNumber } from '../utils/helpers';  
  
const formattedDate = formatDate('2024-01-15');  
const formattedPhone = formatPhoneNumber('5551234567');
```

5. logger.js - Sistem Log Yöneticisi

```
// Özellikler:  
- Güvenli log kayıtları  
- Sadece giriş/çıkış logları  
- IP adresi takibi  
- Hata yönetimi  
- Production güvenliği
```

Log **Metodları**:

- `logUserLogin()` - Giriş kaydı
- `logUserLogout()` - Çıkış kaydı
- `logFailedLogin()` - Başarısız giriş
- `logSystemError()` - Sistem hatası

Güvenlik ve Yetkilendirme

Kullanıcı Roller:

1. Admin - Tam erişim, tüm işlemler
 - o Kullanıcı yönetimi
 - o Sistem ayarları
 - o Tüm verilere erişim
 - o Log kayıtları
2. Team Leader - Ekip yönetimi, raporlar
 - o Ekip performansı
 - o Hedef belirleme
 - o Raporlama
 - o Kendi ekibinin verileri

3. Personel - Kendi kayıtları, temel işlemler

- Kendi kayıtları
- Veri ekleme
- Temel raporlar

Güvenlik Özellikleri:

- Firebase Authentication ile güvenli giriş
- Rol **tabanlı erişim** kontrolü (RBAC)
- Oturum zaman **aşımı** (8 saat varsayılan)
- Güvenli veri **aktarımı** (HTTPS)
- Client-side ve server-side validasyon
- Input sanitization
- XSS **koruması**

Firestore Güvenlik Kuralları:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Kullanıcılar sadece kendi verilerine erişebilir
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }

    // Admin kullanıcılar tüm verilere erişebilir
    match /sales_records/{document} {
      allow read, write: if request.auth != null &&
        (resource.data.createdBy == request.auth.uid ||
        get(/databases/{database}/documents/users/{request.auth.uid}).data.role ==
        'admin');
    }
  }
}
```

Veritabanı Yapısı

Firestore Collections:

1. users - Kullanıcı Bilgileri

```
{
  uid: "string",           // Firebase UID
  email: "string",         // E-posta adresi
  name: "string",          // Kullanıcı adı
  role: "admin|teamLeader|personnel", // Kullanıcı rolü
}
```

```

    isActive: boolean,           // Aktif durum
    createdAt: timestamp,        // Oluşturulma tarihi
    lastLogin: timestamp,        // Son giriş
    phoneNumber: "string",       // Telefon numarası
    department: "string",        // Departman
    sessionTimeout: number       // Oturma süresi (dakika)
}

```

2. sales_records - Satış Kayıtları

```

{
  personel: "string",           // Personel adı
  tarih: "string",              // Arama tarihi
  telefon: "string",            // Müşteri telefonu
  kanal: "string",              // İletişim kanalı
  durum: "string",              // Arama durumu
  detay: "string",              // Sonuç detayı
  abonelikDurum: "string",      // Abonelik durumu
  aboneNo: "string",            // Abone numarası
  not: "string",                // Ek notlar
  createdBy: "string",          // Oluşturan kullanıcı UID
  createdAt: timestamp,         // Oluşturulma tarihi
  updatedAt: timestamp,         // Güncelleme tarihi
  updatedBy: "string"           // Güncelleyen kullanıcı
}

```

3. targets - Hedefler

```

{
  userId: "string",             // Kullanıcı UID
  userName: "string",           // Kullanıcı adı
  month: "string",              // Hedef ayı (YYYY-MM)
  year: number,                 // Hedef yılı
  callTarget: number,           // Arama hedefi
  salesTarget: number,          // Satış hedefi
  actualCalls: number,          // Gerçekleşen aramalar
  actualSales: number,          // Gerçekleşen satışlar
  createdAt: timestamp,         // Oluşturulma tarihi
  createdBy: "string"           // Oluşturan admin
}

```

4. system_logs - Sistem Logları

```

{
  level: "info|warning|error|success|security", // Log seviyesi
  category: "authentication|security|system",    // Kategori
  action: "string",                               // Yapılan işlem
  details: {                                       // Detay bilgileri
    email: "string",
    ip: "string",
    userAgent: "string",
    timestamp: "string"
  },
  userId: "string",                               // Kullanıcı UID
  userName: "string",                             // Kullanıcı adı
  timestamp: timestamp,                           // Log zamanı
}

```

```
ip: "string",           // IP adresi
userAgent: "string",    // Tarayıcı bilgisi
url: "string"           // Sayfa URL'i
}
```

5. dropdown_settings - Dropdown Ayarları

```
{
  kanalList: [           // İletişim kanalları
    "Telefon",
    "WhatsApp",
    "E-posta",
    "Yüz Yüze",
    "Web Site"
  ],
  durumList: [           // Arama durumları
    "Arandı",
    "Meşgul",
    "Ulaşılamadı",
    "Geri Arama",
    "Reddetti"
  ],
  detayList: [           // Sonuç detayları
    "İlgilendi",
    "Düşünecek",
    "Fiyat Sordu",
    "Bilgi İstedi",
    "İlgilenmedi",
    "Satış Sağlandı"
  ],
  abonelikDurumList: [   // Abonelik durumları
    "Yeni Abone",
    "Mevcut Abone",
    "İptal",
    "Askıya Alındı",
    "Bekl emede"
  ]
}
```

6. chat_messages - Chat Mesajları

```
{
  senderId: "string",    // Gönderen UID
  senderName: "string",  // Gönderen adı
  receiverId: "string",  // Alıcı UID
  receiverName: "string", // Alıcı adı
  message: "string",     // Mesaj içeriği
  timestamp: timestamp,  // Gönderim zamanı
  isRead: boolean,       // Okundu mu
  readAt: timestamp,     // Okunma zamanı
  chatId: "string"       // Chat ID (senderId_receiverId)
}
```

✨ Özellikler



Veri Yönetimi

- CRUD **İşlemleri**: Satış kayıtları ekleme, düzenleme, silme
- Excel **Import/Export**: Toplu veri işlemleri
- Veri Validasyonu: Girdi kontrolleri ve temizleme
- Duplicate Detection: Tekrarlanan kayıt tespiti
- Bulk Operations: Toplu güncelleme işlemleri



Analitik ve Raporlama

- Real-time **İstatistikler**: Anlık performans verileri
- Grafik **Görselleştirme**: Chart.js entegrasyonu
- **Karşılaştırmalı** Analiz: Dönemsel karşılaştırmalar
- Export **İşlemleri**: PDF/Excel rapor oluşturma
- Filtreleme ve **Sıralama**: Gelişmiş veri filtreleme



Kullanıcı Yönetimi

- Role-Based Access Control: Rol tabanlı erişim
- User Profile Management: Kullanıcı profil yönetimi
- Session Management: Oturum kontrolü ve zaman aşımı
- Security Logging: Güvenlik log kayıtları
- Password Management: Şifre yönetimi ve sıfırlama



İletişim Sistemi

- Real-time Chat: Anlık mesajlaşma
- User Presence: Online/offline durumu
- Message Threading: Mesaj zincirleme
- Notification System: Bildirim sistemi
- File Sharing: Dosya paylaşımı (gelecek özellik)



Kullanıcı Arayüzü

- Responsive Design: Mobil uyumlu tasarım
- Dark/Light Mode: Tema değiştirme
- Accessibility: Erişilebilir tasarım
- Progressive Web App: PWA özellikleri

- Touch Optimization: Dokunmatik optimizasyon

Sistem Yönetimi

- Configuration Management: Sistem ayarları
- System Monitoring: Performans izleme
- Error Handling: Hata yönetimi
- Backup & Recovery: Yedekleme (gelecek özellik)
- Performance Optimization: Performans optimizasyonu

Kurulum ve Çalıştırma

Sistem Gereksinimleri:

- Node.js 18.0.0 veya üzeri
- npm 8.0.0 veya üzeri
- Firebase Projesi (Authentication + Firestore)
- Modern Web Browser (Chrome, Firefox, Safari, Edge)

Kurulum Adımları:

1. Projeyi Klonlama

```
git clone https://github.com/wupani/satis-crm.git
cd satis-crm
```

2. Bağımlılıkları Yükleme

```
npm install
```

3. Firebase Yapılandırması

`src/auth/firebaseConfig.js` dosyasını kendi Firebase proje bilgilerinizle güncelleyin:

```
const firebaseConfig = {
  apiKey: "your-api-key",
  authDomain: "your-project.firebaseio.com",
  projectId: "your-project-id",
  storageBucket: "your-project.appspot.com",
  messagingSenderId: "your-sender-id",
  appId: "your-app-id"
};
```

4. Firestore Güvenlik Kuralları

`firebase.rules` dosyasını Firebase Console'dan yükleyin.

5. Geliştirme Sunucusu

```
npm run dev
```

Uygulama <http://localhost:5173> adresinde çalışacaktır.

6. Production Build

```
npm run build
```

7. Deploy İşlemi

```
npm run deploy
```

Çevre Değişkenleri:

`.env` dosyası oluşturun:

```
VI TE_FI REBASE_API_KEY=your_api_key  
VI TE_FI REBASE_AUTH_DOMAIN=your_auth_domain  
VI TE_FI REBASE_PROJECT_ID=your_project_id  
VI TE_FI REBASE_STORAGE_BUCKET=your_storage_bucket  
VI TE_FI REBASE_MESSAGING_SENDER_ID=your_sender_id  
VI TE_FI REBASE_APP_ID=your_app_id
```

İlk Kurulum Sonrası:

1. Firebase Console'dan ilk admin kullanıcıyı oluşturun
2. Firestore'da `dropdown_settings` koleksiyonunu oluşturun
3. Sistem ayarlarını yapılandırın
4. Test verilerini ekleyin



Geliştirme Notları

Kod Yapısı ve Standartlar:

- Functional Components: Tüm bileşenler functional olarak yazılmıştır
- React Hooks: useState, useEffect, useContext kullanımı
- Custom Hooks: Tekrar eden logic için özel hook'lar
- ESLint: Kod kalitesi ve standart kontrolü
- Prettier: Kod formatlama (opsiyonel)

State Management:

- React Context API: Global state yönetimi
- Local State: Bileşen bazlı state yönetimi
- Firebase Real-time: Veritabanı senkronizasyonu

Styling Yaklaşımı:

- Tailwind CSS: Utility-first CSS framework
- Mobile-First: Mobil öncelikli responsive tasarım
- CSS Custom Properties: Tema değişkenleri
- Component Styling: Bileşen bazlı stil yönetimi

Performance Optimizasyonları:

- Code Splitting: Lazy loading ile sayfa bazlı kod bölme
- Memoization: React.memo ve useMemo kullanımı
- Debouncing: Arama ve filtreleme optimizasyonu
- Bundle Optimization: Vite build optimizasyonları

Güvenlik Önlemleri:

- Input Sanitization: Kullanıcı girdilerinin temizlenmesi
- XSS Protection: Cross-site scripting koruması
- CSRF Protection: Cross-site request forgery koruması
- Authentication Checks: Her route için kimlik kontrolü
- Role-based Access: Rol tabanlı erişim kontrolü

Testing Stratejisi:

- Unit Tests: Bileşen bazlı testler (gelecek)
- Integration Tests: API entegrasyon testleri (gelecek)
- E2E Tests: Uçtan uca testler (gelecek)
- Manual Testing: Manuel test senaryoları

Güncellemeler ve Versiyon Geçmişi

v1.3.2 (Mevcut - Ocak 2024)

✨ Yeni Özellikler:

- Mobil responsive tasarım tamamen yenilendi
- Touch interaction optimizasyonları eklendi
- iOS uyumluluğu artırıldı
- Mobile-first yaklaşımı uygulandı



Düzeltilmeler:

- Z-index sorunları çözüldü
- Sidebar mobil erişim problemi giderildi
- Touch target boyutları optimize edildi
- Mobile browser compatibility iyileştirildi



Teknik İyileştirmeler:

- CSS grid ve flexbox optimizasyonları
- Responsive breakpoint'ler güncellendi
- Mobile navigation iyileştirmeleri
- Performance optimizasyonları

v1.2.0 (Aralık 2023)



Yeni Özellikler:

- Chat sistemi eklendi
- Real-time mesajlaşma
- User presence tracking
- Notification system



İyileştirmeler:

- Performance optimizasyonları
- Memory leak düzeltmeleri
- Error handling iyileştirmeleri

v1.1.0 (Kasım 2023)



Yeni Özellikler:

- Analytics sayfası eklendi
- PDF/Excel export işlemleri
- Advanced filtering options
- Data visualization charts



İyileştirmeler:

- Role management system
- Security enhancements

- UI/UX improvements

v1.0.0 (Ekim 2023)



İlk Kararlı Sürüm:

- Temel CRUD işlemleri
- User authentication
- Role-based access control
- Basic reporting
- Responsive design
- Firebase integration



Gelecek Özellikler (Roadmap)

v1.4.0 (Planlanan - Şubat 2024)

- ☐ Advanced analytics dashboard
- ☐ Custom report builder
- ☐ Email notifications
- ☐ API endpoints
- ☐ Third-party integrations

v1.5.0 (Planlanan - Mart 2024)

- ☐ Mobile app (React Native)
- ☐ Offline support
- ☐ Advanced user permissions
- ☐ Audit trail
- ☐ Data backup/restore

v2.0.0 (Planlanan - Q2 2024)

- ☐ Microservices architecture
- ☐ GraphQL API
- ☐ Advanced AI features
- ☐ Multi-tenant support
- ☐ Enterprise features

Bilinen Sorunlar ve Çözümler

Yaygın Sorunlar:

1. Firebase Connection Issues

Sorun: Firebase bağlantı hataları Çözüm:

```
// Network retry logic
const retryConnection = async (fn, retries = 3) => {
  try {
    return await fn();
  } catch (error) {
    if (retries > 0) {
      await new Promise(resolve => setTimeout(resolve, 1000));
      return retryConnection(fn, retries - 1);
    }
    throw error;
  }
};
```

2. Mobile Safari Issues

Sorun: iOS Safari'de scroll problemi Çözüm:

```
/* iOS scroll fix */
.scroll-container {
  -webkit-overflow-scrolling: touch;
  overflow-y: auto;
}
```

3. Memory Leaks

Sorun: Component unmount sonrası memory leak Çözüm:

```
useEffect(() => {
  const unsubscribe = onSnapshot(query, callback);
  return () => unsubscribe(); // Cleanup
}, []);
```

Performans İyileştirmeleri:

1. Bundle Size Optimization

```
// Dynamic imports
const AnalyticsPage = lazy(() => import('./pages/Analytics'));
```

2. Image Optimization

```
// Lazy loading images
<img loading="lazy" src={imageUrl} alt="description" />
```

3. Database Query Optimization

```
// Efficient queries
const q = query(
  collection(db, 'sales_records'),
  where('createdBy', '==', userId),
  orderBy('createdAt', 'desc'),
  limit(10)
);
```

Destek ve İletişim

Teknik Destek:

- GitHub Issues: [Proje Issues](#)
- E-posta: wupaniyazilim@gmail.com
- Dokümantasyon: Bu dosya ve kod içi yorumlar

Katkıda Bulunma:

1. Fork the repository
2. Create feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

Lisans:

Bu proje özel lisans altında geliştirilmiştir. Ticari kullanım için iletişime geçiniz.

Kaynaklar ve Referanslar

Teknoloji Dokümantasyonları:

- [React Documentation](#)
- [Firebase Documentation](#)
- [Tailwind CSS Documentation](#)
- [Vite Documentation](#)

Kullanılan Kütüphaneler:

- [Lucide React Icons](#)
- [React Router](#)

- [jsPDF](#)
- [SheetJS](#)

Design Resources:

- [Tailwind UI Components](#)
 - [Heroicons](#)
 - [Headless UI](#)
-

API Referansı ve Fonksiyonlar

Firestore İşlemleri

1. Veri Okuma **Fonksiyonları**

```
// Tüm kayıtları getir
const getAllRecords = async () => {
  const q = query(collection(db, 'sales_records'));
  const snapshot = await getDocs(q);
  return snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
};

// Kullanıcıya göre kayıtları getir
const getUserRecords = async (userId) => {
  const q = query(
    collection(db, 'sales_records'),
    where('createdBy', '==', userId)
  );
  const snapshot = await getDocs(q);
  return snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
};

// Tarih aralığına göre kayıtları getir
const getRecordsByDateRange = async (startDate, endDate) => {
  const q = query(
    collection(db, 'sales_records'),
    where('createdAt', '>=', startDate),
    where('createdAt', '<=', endDate),
    orderBy('createdAt', 'desc')
  );
  const snapshot = await getDocs(q);
  return snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
};
```

2. Veri Yazma **Fonksiyonları**

```
// Yeni kayıt ekle
const addRecord = async (recordData) => {
  const docRef = await addDoc(collection(db, 'sales_records'), {
    ...recordData,
  });
};
```

```

    createdAt: serverTimestamp(),
    refId: generateRefId()
  });
  return docRef.id;
};

// Kayıt güncelle
const updateRecord = async (recordId, updateData) => {
  const docRef = doc(db, 'sales_records', recordId);
  await updateDoc(docRef, {
    ...updateData,
    updatedAt: serverTimestamp()
  });
};

// Kayıt sil
const deleteRecord = async (recordId) => {
  const docRef = doc(db, 'sales_records', recordId);
  await deleteDoc(docRef);
};

```

3. Real-time Listeners

```

// Gerçek zamanlı veri dinleme
const listenToRecords = (callback) => {
  const q = query(
    collection(db, 'sales_records'),
    orderBy('createdAt', 'desc')
  );

  return onSnapshot(q, (snapshot) => {
    const records = snapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
    callback(records);
  });
};

// Chat mesajlarını dinle
const listenToMessages = (chatId, callback) => {
  const q = query(
    collection(db, 'chat_messages'),
    where('chatId', '==', chatId),
    orderBy('timestamp', 'asc')
  );

  return onSnapshot(q, (snapshot) => {
    const messages = snapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
    callback(messages);
  });
};

```

Yardımcı Fonksiyonlar (helpers.js)

1. Veri **Formatlaması**

```
// Benzersiz referans ID oluştur
export const generateRefId = () => {
  const timestamp = Date.now();
  const random = Math.random().toString(36).substring(2, 8);
  return `REF-${timestamp}-${random.toUpperCase()}`;
};

// Tarih formatla (DD/MM/YYYY)
export const formatDate = (date) => {
  if (!date) return '';
  const d = new Date(date);
  const day = String(d.getDate()).padStart(2, '0');
  const month = String(d.getMonth() + 1).padStart(2, '0');
  const year = d.getFullYear();
  return `${day}/${month}/${year}`;
};

// Telefon numarası formatla
export const formatPhoneNumber = (phone) => {
  if (!phone) return '';
  const cleaned = phone.replace(/\D/g, '');
  if (cleaned.length === 11 && cleaned.startsWith('0')) {
    return cleaned.replace(/(\d{4})(\d{3})(\d{2})(\d{2})/, '$1 $2 $3 $4');
  }
  return phone;
};
```

2. Validasyon **Fonksiyonları**

```
// E-posta validasyonu
export const validateEmail = (email) => {
  const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return re.test(email);
};

// Telefon numarası validasyonu
export const validatePhone = (phone) => {
  const cleaned = phone.replace(/\D/g, '');
  return cleaned.length >= 10 && cleaned.length <= 11;
};

// Güçlü şifre kontrolü
export const validatePassword = (password) => {
  return password.length >= 6;
};
```

UI/UX Tasarım Sistemi

Renk Paleti

1. Ana Renkler

```
/* Mor Tonları */
--purple-50: #faf7ff;
--purple-100: #f3e8ff;
--purple-500: #a855f7;
--purple-600: #9333ea;
--purple-700: #7c3aed;

/* Lavanta Tonları */
--lavender-50: #fdfcff;
--lavender-100: #f8f4ff;
--lavender-500: #c084fc;

/* Periwinkle Tonları */
--periwinkle-50: #f0f0ff;
--periwinkle-500: #7a7aff;
```

2. Gradyanlar

```
/* Ana Gradyanlar */
.gradient-primary {
  background: linear-gradient(135deg, #a855f7 0%, #9333ea 100%);
}

.gradient-soft {
  background: linear-gradient(135deg, #faf7ff 0%, #f3e8ff 100%);
}

.gradient-glass {
  background: rgba(168, 85, 247, 0.1);
  backdrop-filter: blur(10px);
}
```

Tipografi

1. Font Ailesi

```
/* Ana Font */
font-family: 'Poppins', system-ui, sans-serif;

/* Boyutlar */
.text-2xs: 0.625rem; /* 10px */
.text-xs: 0.75rem; /* 12px */
.text-sm: 0.875rem; /* 14px */
.text-base: 1rem; /* 16px */
.text-lg: 1.125rem; /* 18px */
.text-xl: 1.25rem; /* 20px */
.text-2xl: 1.5rem; /* 24px */
.text-3xl: 1.875rem; /* 30px */
```


Animasyonlar

1. Özel Animasyonlar

```
/* Float Animasyonu */
@keyframes float {
  0%, 100% { transform: translateY(0px); }
  50% { transform: translateY(-10px); }
}

/* Fade In */
@keyframes fadeIn {
  0% { opacity: 0; }
  100% { opacity: 1; }
}

/* Scale In */
@keyframes scaleIn {
  0% { transform: scale(0.95); opacity: 0; }
  100% { transform: scale(1); opacity: 1; }
}
```

Responsive Breakpoints

```
/* Ekran Boyutları */
xs: 475px; /* Extra Small */
sm: 640px; /* Small */
md: 768px; /* Medium */
lg: 1024px; /* Large */
xl: 1280px; /* Extra Large */
2xl: 1536px; /* 2X Large */
```

Yapılandırma Dosyaları

1. Vite Yapılandırması (vite.config.js)

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig(({ command }) => {
  return {
    plugins: [react()],
    base: command === 'build' ? '/satis-crm/' : '/',
    build: {
      outDir: 'dist',
      rollupOptions: {
        output: {
          manualChunks: {
            vendor: ['react', 'react-dom'],
            firebase: ['firebase/app', 'firebase/auth', 'firebase/firestore'],
            utils: ['lucide-react', 'xlsx', 'jspdf']
          }
        }
      }
    }
  }
})
```

```

    }
  },
  server: {
    port: 5173,
    host: true,
    open: true
  },
  preview: {
    port: 4173,
    host: true
  }
}
})

```

2. ESLint Yapılandırması (eslint.config.js)

```

import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'

export default [
  { ignores: ['dist'] },
  {
    files: ['**/*.jsx'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
    },
    rules: {
      ...reactHooks.configs.recommended.rules,
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
    },
  },
]

```

3. PostCSS Yapılandırması (postcss.config.js)

```

export default {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
}

```

Performans ve Optimizasyon

1. Bundle Analizi

```
# Bundle boyutunu analiz et
npm run build
npx vite-bundle-analyzer dist

# Performans metrikleri
npm run preview
# Lighthouse audit çalıştır
```

2. Lazy Loading Implementasyonu

```
// Sayfa bazlı lazy loading
import { lazy, Suspense } from 'react';

const Analytics = lazy(() => import('./pages/Analytics'));
const Dashboard = lazy(() => import('./pages/Dashboard'));

// Loading component
const PageLoader = () => (
  <div className="flex justify-center items-center h-64">
    <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-purple-600"></div>
  </div>
);

// Kullanım
<Suspense fallback={<PageLoader />}>
  <Analytics />
</Suspense>
```

3. Memory Management

```
// Component cleanup
useEffect(() => {
  const unsubscribe = onSnapshot(query, callback);

  // Cleanup function
  return () => {
    unsubscribe();
  };
}, []);

// Event listener cleanup
useEffect(() => {
  const handleResize = () => {
    // Handle resize
  };

  window.addEventListener('resize', handleResize);

  return () => {
    window.removeEventListener('resize', handleResize);
  };
}, []);
```

```
};  
}, []);
```

Test Senaryoları

1. Kullanıcı Testleri

```
// Login testi  
describe('User Authentication', () => {  
  test('should login with valid credentials', async () => {  
    // Test implementation  
  });  
  
  test('should show error with invalid credentials', async () => {  
    // Test implementation  
  });  
});  
  
// CRUD işlemleri testi  
describe('Record Management', () => {  
  test('should create new record', async () => {  
    // Test implementation  
  });  
  
  test('should update existing record', async () => {  
    // Test implementation  
  });  
});
```

2. Manuel Test Checklist

- ☐ Kullanıcı girişi ve çıkışı
 - ☐ Kayıt ekleme, düzenleme, silme
 - ☐ Filtreleme ve arama
 - ☐ Export işlemleri
 - ☐ Responsive design
 - ☐ Chat sistemi
 - ☐ Rol tabanlı erişim
 - ☐ Performans testleri
-

Deployment ve DevOps

1. GitHub Pages Deployment

```
# Build ve deploy  
npm run build
```

```
npm run deploy

# Otomatik deployment için GitHub Actions
# .github/workflows/deploy.yml
name: Deploy to GitHub Pages
on:
  push:
    branches: [ main ]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Setup Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '18'
      - name: Install dependencies
        run: npm install
      - name: Build
        run: npm run build
      - name: Deploy
        run: npm run deploy
```

2. Environment Variables

```
# Development
VITE_NODE_ENV=development
VITE_FIREBASE_API_KEY=dev_api_key

# Production
VITE_NODE_ENV=production
VITE_FIREBASE_API_KEY=prod_api_key
```

Progressive Web App (PWA) Özellikleri

1. Manifest Dosyası

```
{
  "name": "Satış Takip CRM",
  "short_name": "SatışCRM",
  "description": "Satış takip ve CRM sistemi",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#a855f7",
  "theme_color": "#9333ea",
  "icons": [
    {
      "src": "/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
```

```

      "src": "/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}

```

2. Service Worker (Gelecek Özellik)

```

// sw.js
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open('crm-v1').then((cache) => {
      return cache.addAll([
        '/',
        '/static/js/bundle.js',
        '/static/css/main.css'
      ]);
    })
  );
});

```

Güvenlik Best Practices

1. Input Sanitization

```

// XSS koruması
const sanitizeInput = (input) => {
  return input
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;')
    .replace(/"/g, '&quot;')
    .replace(/'/g, '&#x27;')
    .replace(/\\/g, '&#x2F;');
};

// SQL Injection koruması (Firestore otomatik korur)
const safeQuery = (userInput) => {
  // Firestore parametrelili sorgular kullan
  return query(
    collection(db, 'records'),
    where('field', '==', userInput) // Güvenli
  );
};

```

2. Authentication Security

```

// Güçlü şifre politikası
const passwordPolicy = {
  minLength: 8,
  requireUppercase: true,
  requireLowercase: true,
  requireNumbers: true,
  requireSymbols: false
};

```

```
};

// Session timeout
const SESSION_TIMEOUT = 8 * 60 * 60 * 1000; // 8 saat
```

Analytics ve Monitoring

1. Performance Monitoring

```
// Performance metrikleri
const measurePerformance = (name, fn) => {
  const start = performance.now();
  const result = fn();
  const end = performance.now();
  console.log(`${name} took ${end - start} milliseconds`);
  return result;
};

// Error tracking
window.addEventListener('error', (event) => {
  // Error logging
  console.error('Global error:', event.error);
});
```

2. User Analytics (Gelecek Özellik)

```
// Google Analytics entegrasyonu
import { gtag } from 'ga-gtag';

const trackEvent = (action, category, label) => {
  gtag('event', action, {
    event_category: category,
    event_label: label
  });
};
```

Backup ve Recovery

1. Veri Yedekleme Stratejisi

```
// Firestore export (Admin SDK gerekli)
const backupFirestore = async () => {
  // Cloud Functions ile implement edilecek
  const backup = await admin.firestore().export({
    outputUriPrefix: 'gs://backup-bucket/backups',
    collectionIds: ['sales_records', 'users', 'system_logs']
  });
  return backup;
};
```

2. Disaster Recovery Plan

1. Günlük otomatik yedekleme
 2. Multi-region deployment
 3. Data replication
 4. Recovery testing
-

Maintenance Checklist

Günlük Kontroller:

- ☐ Sistem logları kontrolü
- ☐ Performance metrikleri
- ☐ Error rate monitoring
- ☐ User activity logs

Haftalık Kontroller:

- ☐ Database backup verification
- ☐ Security updates
- ☐ Performance optimization
- ☐ User feedback review

Aylık Kontroller:

- ☐ Dependency updates
 - ☐ Security audit
 - ☐ Performance analysis
 - ☐ Feature usage analytics
-

Eğitim ve Dokümantasyon

1. Kullanıcı Kılavuzları

- Admin **Kullanıcı Kılavuzu**: Sistem yönetimi
- Personel **Kullanıcı Kılavuzu**: Günlük kullanım
- API Dokümantasyonu: Geliştirici referansı

2. Video Eğitimleri (Planlanan)

- Sistem kurulumu
 - Temel kullanım
 - Gelişmiş özellikler
 - Sorun giderme
-

En İyi Uygulamalar

1. Kod Kalitesi

```
// Consistent naming
const getUserRecords = async (userId) => { ... };
const updateUserRecord = async (recordId, data) => { ... };

// Error handling
try {
  const result = await apiCall();
  return result;
} catch (error) {
  logger.error('API call failed:', error);
  throw new Error('İşlem başarısız oldu');
}

// Type checking (JSDoc)
/**
 * @param {string} userId - Kullanıcı ID
 * @param {Object} recordData - Kayıt verisi
 * @returns {Promise<string>} Kayıt ID
 */
const createRecord = async (userId, recordData) => { ... };
```

2. Performance Best Practices

- Lazy loading kullan
 - Memoization uygula
 - Bundle size'ı optimize et
 - Image optimization
 - Caching strategies
-

Bu dokümantasyon, **Satış** Takip CRM Sistemi'nin tüm teknik **detaylarını**, **kullanım kılavuzlarını**, API **referanslarını**, güvenlik önlemlerini, performans **optimizasyonlarını** ve **bakım** süreçlerini içermektedir. Proje sürekli **geliştirilmekte** ve güncellemeler **yapılmaktadır**.

Son güncelleme: Ocak 2024 - v1.3.2 (Kapsamlı Versiyon)