Smart Home System
Kapsamlı Teknik Proje Dökümantasyonu

Nazlıcan Aka

23 Şubat 2026

İçindekiler

1 Tech Stack **3**
- 1.1 Backend . . . 3
- 1.2 Frontend . . . 3

2 Kimlik Doğrulama ve Yetkilendirme Mimarisi **3**
- 2.1 A. Veri Modeli ve Kullanıcı Rolleri . . . 3
- 2.2 B. JWT Yapılandırması (Backend) . . . 3
- 2.3 C. Giriş ve Kayıt Süreci (Authentication) . . . 4

3 Yetkilendirme Mekanizması (Authorization) **4**

4 İstemci Tarafı Yönetimi (WPF) **4**

5 Cihaz Yönetimi (Device Management) **4**

6 Senaryolar ve Otomasyon (Scenarios) **5**

7 Geçmiş ve İstatistikler (History/Statistics) **5**

8 Olay Tabanlı Mimari (Event-Driven Architecture) **5**
- 8.1 1. Domain Events (Alan Olayları) . . . 5
- 8.2 2. Event Dispatcher (Merkezi Mesaj Dağıtıcı) . . . 6
- 8.3 3. Event Handlers . . . 6
- 8.4 4. Gerçek Zamanlı Bildirim Altyapısı (SignalR) . . . 6
- 8.5 5. Mimari Avantajlar . . . 6

9 Adaptör (Adapter/Driver) Yaklaşımı ve Mimari Yapı **7**
- 9.1 Cihaz Eşleştirme (Pairing) Mekanizması . . . 7
- 9.2 Esneklik ve Genişletilebilirlik (Scalability) . . . 7
- 9.3 Teknik Özet . . . 7

10 WPF Masaüstü Uygulama Mimarisi (Client) **7**
- 10.1 Cihaz Yönetimi ve Kullanıcı Arayüzü . . . 8
- 10.2 Senaryo Yönetimi ve Geçmiş Görüntüleme . . . 8
- 10.3 Backend Entegrasyonu ve REST İletişimi . . . 8
- 10.4 Gerçek Zamanlı Güncelleme (SignalR Entegrasyonu) . . . 8
  - 10.4.1 A. Mimari Akış ve Operasyonel Süreç . . . 8
  - 10.4.2 B. Bileşen Detayları . . . 9

1 Tech Stack

1.1 Backend

- .NET 10 (ASP.NET Core Web API)
- Entity Framework Core + SQLite
- SignalR (WebSocket)
- JWT Authentication

1.2 Frontend

- WPF (.NET 10)
- MVVM Pattern (CommunityToolkit.Mvvm)
- SignalR Client

2 Kimlik Doğrulama ve Yetkilendirme Mimarisi

Sistemde güvenlik, JSON Web Token (JWT) tabanlı bir kimlik doğrulama ve Rol Tabanlı Yetkilendirme (RBAC) modeli üzerine inşa edilmiştir. Bu yapı, kullanıcıların sisteme giriş yaparak bir bilet (token) almasını ve bu biletle yetkileri dahilindeki işlemleri gerçekleştirmesini sağlar.

2.1 A. Veri Modeli ve Kullanıcı Rolleri

Sistemde iki temel kullanıcı rolü tanımlanmıştır:

- **Parent (Ebeveyn):** Tam yetkiye sahip kullanıcı. Cihaz ekleme, silme ve geçmişi temizleme gibi kritik işlemleri yapabilir.
- **Child (Çocuk):** Kısıtlı yetkiye sahip kullanıcı. Sadece cihazları açıp kapatabilir ve durumlarını izleyebilir.

Kullanıcı bilgileri veritabanında `UserEntity` sınıfı ile temsil edilir ve `Id`, `Username`, `Password` ve `Role` alanlarını içerir.

2.2 B. JWT Yapılandırması (Backend)

Sistemin güvenliği `Program.cs` içerisinde `AddAuthentication` ve `AddJwtBearer` servisleri ile yapılandırılmıştır. Token doğrulaması için şu parametreler kullanılır:

- **Issuer/Audience:** Token'ı dağıtan ve kabul eden tarafların doğrulanması.
- **IssuerSigningKey:** Token'ın imzalanması için `appsettings.json` içerisindeki gizli anahtar (Key) kullanılır.

2.3 C. Giriş ve Kayıt Süreci (Authentication)

`AuthController`, kullanıcı giriş ve kayıt işlemlerini yöneten merkezdir:

- **Login:** Kullanıcı adı ve şifre kontrol edildikten sonra, kullanıcıya özel bir JWT üretilir. Bu token içerisine kullanıcının `Name` ve `Role` bilgileri *Claim* olarak gömülür.
- **Register:** Yeni kullanıcılar sisteme belirli bir rolle (Parent/Child) kaydedilir.

3 Yetkilendirme Mekanizması (Authorization)

Sistemdeki API uç noktaları, kullanıcıların rollerine göre `[Authorize]` attribute ile korunmaktadır:

- **Genel Erişim:** `GetAllDevices` veya `ToggleDevice` gibi metodlar sadece `[Authorize]` özniteliği taşır; yani geçerli bir token'ı olan her kullanıcı bu işlemleri yapabilir.
- **Rol Kısıtlaması:** `AddDevice`, `DeleteDevice` ve `ClearHistory` gibi kritik metodlar `[Authorize(Roles = "Parent")]` özniteliği ile korunur. Bu metodlara "Child" rolündeki bir kullanıcı erişmeye çalıştığında sistem otomatik olarak erişimi engeller.

4 İstemci Tarafı Yönetimi (WPF)

WPF tarafında kimlik yönetimi `ApiService` ve `LoginViewModel` üzerinden yürütülür:

- **Token Saklama:** Kullanıcı başarıyla giriş yaptığında API'den dönen JWT, `ApiService` içerisinde statik bir değişkende saklanır.
- **Header Entegrasyonu:** Giriş yapıldıktan sonra gönderilen tüm HTTP isteklerinin başlığına (Header) `Authorization: Bearer <token>` formatında bu bilet eklenir.
- **UI Tepkisi:** Eğer kullanıcı bir işlem yapmaya yetkili değilse (örneğin Child rolüyle cihaz silmeye çalışırsa), API'den gelen hata kodu yakalanır ve kullanıcıya "Yetkiniz yok" mesajı gösterilir.

5 Cihaz Yönetimi (Device Management)

Cihazların sisteme dahil edilmesi, kontrolü ve takibi için uçtan uca bir altyapı kurulmuştur.

- **Dinamik Cihaz Ekleme ve Silme:** "Parent" rolüne sahip kullanıcılar, sistem üzerinden cihaz adı, tipi (Işık, Termostat vb.) ve iletişim protokolünü (Wi-Fi, Bluetooth) seçerek yeni cihazlar tanımlayabilir veya mevcut cihazları silebilir.
- **Protokol Adaptör Yapısı:** Farklı donanım protokolleri için *Adapter Pattern* kullanılmıştır. Bu sayede cihazın Wi-Fi veya Bluetooth üzerinden bağlanması fark etmeksizin, sistem `IDeviceProtocolAdapter` arayüzü üzerinden standart bir komut seti kullanır.
- **Gerçek Zamanlı Durum Kontrolü:** Cihazların açma/kapama işlemleri `ToggleDeviceAsync` metodu ile yönetilir. SignalR entegrasyonu sayesinde, bir cihazın durumu değiştiğinde tüm bağlı arayüzler anında güncellenir.

6 Senaryolar ve Otomasyon (Scenarios)

Sistemde cihazlar arası etkileşimi sağlayan akıllı senaryolar ve kullanıcı merkezli modlar bulunmaktadır.

- **Cihazlar Arası Otomasyon (Automation Rule Handler):** Belirli cihaz hareketleri diğerlerini tetikleyecek şekilde kurgulanmıştır. Örneğin; Robot Süpürge çalışmaya başladığında evdeki Hava Temizleyicilerin otomatik olarak kapanması, süpürge şarj istasyonuna döndüğünde ise tekrar açılması sağlanmıştır.
- **Varlık Modu (Presence Mode):** Kullanıcının eve gelme veya evden ayrılma durumuna göre toplu cihaz aksiyonları tanımlanmıştır. "Eve Gelindi" işaretlendiğinde ışıklar ve termostatlar otomatik olarak açılır.
- **Enerji Tasarrufu Modu:** Sistem, açık unutulan ışıkları tespit edip kapatabilen bir enerji tasarrufu mekanizmasına sahiptir.

7 Geçmiş ve İstatistikler (History/Statistics)

Sistemdeki tüm hareketler denetim ve analiz amacıyla kayıt altına alınmaktadır.

- **Detaylı İşlem Kaydı (Logging):** Her cihaz hareketi (ekleme, silme, açılma, kapanma); işlemin zamanı, hangi cihaz üzerinde yapıldığı ve işlemi kimin (Kullanıcı adı, Otomasyon veya Sistem) tetiklediği bilgileriyle birlikte `DeviceHistory` tablosunda saklanır.
- **Cihaz Bazlı Geçmiş Sorgulama:** Kullanıcılar ister tüm evin geçmişini, isterlerse sadece belirli bir cihazın kullanım geçmişini arayüz üzerinden filtreleyerek görüntüleyebilir.
- **Görsel Durum Göstergeleri:** WPF arayüzündeki `DataGrid` üzerinde cihazların güncel durumları (AÇIK/KAPALI) ve geçmiş aksiyonlar, renk kodlu görselleştirme ile sunulur.

8 Olay Tabanlı Mimari (Event-Driven Architecture)

8.1 1. Domain Events (Alan Olayları)

Sistemde meydana gelen her önemli iş değişikliği bir "Olay" olarak tanımlanmıştır. `DomainEvent` soyut sınıfından türetilen bu yapılar, olayın ne zaman ve hangi benzersiz ID ile gerçekleştiğini takip eder.

- **Cihaz Durum Değişikliği:** `DeviceStateChangedEvent`, bir cihazın durumunun kim tarafından ve neden değiştirildiğini taşır.
- **Sistem Hareketleri:** Cihaz ekleme (`DeviceAddedEvent`) ve silme (`DeviceRemovedEvent`) işlemleri için özel olaylar tanımlanmıştır.
- **Akıllı Senaryo Olayları:** Otomasyonların tetiklenmesi, kullanıcı varlık durumu ve enerji tasarrufu modları için özel alan olayları mevcuttur.

8.2 2. Event Dispatcher (Merkezi Mesaj Dağıtıcı)

Sistem, loosely coupled bir yapı sağlamak için bir *In-Process Message Bus* (Event Dispatcher) kullanır.

- **Pub/Sub Mekanizması:** `IDeviceService` bir olayı yayınladığında, içeriğini bilmesine gerek duymadan `IEventDispatcher` üzerinden sisteme duyurur.
- **Scope Yönetimi:** `EventDispatcher`, her bir olay işleme süreci için yeni bir servis kapsamı (scope) oluşturarak scoped servislerin güvenli kullanımını sağlar.
- **Çoklu Dinleyici Desteği:** Tek bir olay, birbirinden bağımsız birden fazla "Handler" tarafından aynı anda işlenebilir.

8.3 3. Event Handlers

Yayınlanan olaylar, uzmanlaşmış işleyiciler tarafından asenkron olarak ele alınır:

- **AutomationRuleHandler:** Cihazlar arası etkileşimi yönetir (örn. robot süpürge çalışınca hava temizleyiciyi kapatma).
- **SignalRNotificationHandler:** Backend'deki olaylardan sonra bağlı tüm istemcilere gerçek zamanlı WebSocket mesajı gönderir.

8.4 4. Gerçek Zamanlı Bildirim Altyapısı (SignalR)

Kullanıcı deneyimini artırmak için *Pub/Sub* modeli SignalR ile desteklenmiştir:

- **Broadcasting (Yayınlama):** `SignalRNotificationHandler` üzerinden gelen olaylar tüm istemcilere yayınlanır.
- **WPF Entegrasyonu:** `SignalRService` sınıfı arka planda bağlı kalarak arayüzün anında güncellenmesini sağlar.
- **Bağlantı Yönetimi:** İstemci tarafında otomatik yeniden bağlanma ve durum takibi mevcuttur.

8.5 5. Mimari Avantajlar

- **Genişletilebilirlik:** Yeni bir bildirim türü eklemek için sadece yeni bir `IEventHandler` yazmak yeterlidir.
- **İzlenebilirlik:** Her olay `DeviceHistory` tablosuna loglanarak tam bir denetim izi oluşturulur.

9 Adaptör (Adapter/Driver) Yaklaşımı ve Mimari Yapı

Sistemde farklı iletişim protokollerine sahip cihazların tek bir merkezden yönetilebilmesi için "Loose Coupling" prensibi uygulanmıştır. Bu sayede iş mantığı, cihazın fiziksel dilinden bağımsız hale getirilmiştir.

- **Soyutlama Katmanı:** Tüm protokollerin uyması gereken kurallar `IDeviceProtocolAdapter` arayüzü ile tanımlanmıştır.
- **Protokol Bazlı Özelleşme:** `WiFiAdapter` ve `BluetoothAdapter` sınıfları protokol spesifik mantıkları yönetir.

9.1 Cihaz Eşleştirme (Pairing) Mekanizması

- **Dinamik Adaptör Seçimi:** Sistem, *Dependency Injection* üzerinden ilgili protokolü destekleyen adaptörü otomatik olarak bulur.
- **Eşleşme Simülasyonu:** `PairDeviceAsync` metodu, asenkron bir gecikme ile gerçek bir donanım tarama sürecini taklit eder.
- **Doğrulama ve Kayıt:** Adaptörden başarılı yanıt gelmeden cihaz veritabanına kaydedilmez.

9.2 Esneklik ve Genişletilebilirlik (Scalability)

- **Yeni Protokol Desteği:** Yeni bir protokol eklemek için mevcut kodlarda değişiklik yapmadan sadece yeni bir adaptör sınıfı yazılması yeterlidir.
- **Hata Yönetimi:** `DeviceService`, cihazın protokolüne bakmadan standart komutu adaptör üzerinden yayınlar.

9.3 Teknik Özet

- **Interface:** `IDeviceProtocolAdapter`.
- **Somut Sınıflar:** `WiFiAdapter`, `BluetoothAdapter`.
- **Pattern:** Adapter Design Pattern.
- **Yönetim:** `IEnumerable<IDeviceProtocolAdapter>` olarak enjekte edilir.

10 WPF Masaüstü Uygulama Mimarisi (Client)

Kullanıcı arayüzü WPF (.NET 10) ve MVVM tasarımı ile geliştirilmiştir.

- **View (Arayüz):** XAML dosyalarıyla (MainWindow, LoginWindow) kurgulanmıştır.
- **ViewModel:** `MainViewModel` ve `LoginViewModel` sınıfları etkileşimi ve servis iletişimini yönetir.
- **Model:** API'den dönen JSON verileri C# sınıflarına dönüştürülerek kullanılır.

10.1 Cihaz Yönetimi ve Kullanıcı Arayüzü

- **Listeleme ve Kontrol:** `DataGrid` üzerinde gerçek zamanlı durum takibi ve kontrol butonları sunulur.
- **Eşleştirme ve Ekleme:** Form üzerinden yeni donanım ekleme işlemi adaptörleri tetikler.
- **Görsel Bildirimler:** İşlem sonuçları `MessageBox` ile kullanıcıya iletilir.

10.2 Senaryo Yönetimi ve Geçmiş Görüntüleme

- **Senaryo Tetikleme:** Varlık modu manuel olarak tetiklenebilir.
- **Detaylı Geçmiş Ekranı:** `DeviceHistoryWindow` üzerinden sistem hareketleri izlenir.
- **Filtreleme:** Cihaz bazlı geçmiş verileri görüntülenebilir.

10.3 Backend Entegrasyonu ve REST İletişimi

- **RESTful Bağlantı:** `ApiService` üzerinden GET, POST ve DELETE istekleri yönetilir.
- **JWT Yetkilendirme:** Token, isteklere "Authorization: Bearer" başlığı ile eklenir.

10.4 Gerçek Zamanlı Güncelleme (SignalR Entegrasyonu)

Bu sistem, sunucu tarafında (Backend) gerçekleşen bir değişikliğin, kullanıcı arayüzünde (Frontend) sayfa yenilemeye gerek kalmadan anında görünmesini sağlayan **Olay Tabanlı (Event-Driven)** bir iletişim altyapısı üzerine kuruludur.

10.4.1 A. Mimari Akış ve Operasyonel Süreç

Sistemin gerçek zamanlı çalışma döngüsü şu adımları izler:

1. **Olayın Tetiklenmesi (Backend):** `DeviceService` üzerinde bir cihazın durumu değiştirildiğinde (açma/kapama, yeni cihaz ekleme vb.), sistem bir *Domain Event* (örneğin `DeviceStateChangedEvent`) yayınlar.
2. **Olayın Yakalanması ve Dağıtımı:** `EventDispatcher`, bu olayı dinleyen `SignalRNotificationHandler` sınıfını tetikler.
3. **Broadcasting (Yayınlama):** `SignalRNotificationHandler`, gelen olay verisini istemcinin anlayacağı bir JSON mesaj formatına dönüştürür ve `DeviceNotificationHub` üzerinden bağlı olan tüm istemcilere (WPF uygulamalarına) yayınlar.
4. **İstemci Tarafında Karşılama (WPF):** `SignalRService`, WebSocket üzerinden gelen bu mesajı yakalar ve ilgili C# event'ini (örneğin `DeviceStateChanged`) tetikler.
5. **Arayüzün Güncellenmesi:** `MainViewModel`, bu event'i yakaladığında asenkron olarak `LoadDevicesAsync` metodunu çağırır ve `ObservableCollection` yapısını güncelleyerek `DataGrid` üzerindeki bilgilerin anında değişmesini sağlar.

10.4.2 B. Bileşen Detayları

- **SignalRService:** Uygulama başlatıldığında `https://localhost:7106/hubs/notifications` adresine sürekli bir WebSocket bağlantısı kurar. `WithAutomaticReconnect()` özelliği sayesinde bağlantı koptuğunda otomatik olarak yeniden bağlanmaya çalışır.
- **Anlık Güncelleme:** Sunucudaki değişiklikler (bir cihazın kullanıcı tarafından açılması, bir otomasyonun tetiklenmesi veya enerji tasarrufu modunun ışıkları kapatması) yakalanarak ekran anında güncellenir. Bu sayede birden fazla kullanıcı aynı anda sistemi izliyorsa, birinin yaptığı değişiklik diğerinin ekranında saniyeler içinde belirir.
- **Durum Takibi:** İstemci, sunucu ile olan bağlantı durumunu (Bağlı, Bağlanıyor, Yeniden Bağlanıyor, Bağlantı Kesildi) anlık olarak izler. Bu durum bilgisi, `MainViewModel` üzerindeki `ConnectionStatus` özelliği aracılığıyla ana ekrandaki bildirim çubuğunda görsel olarak kullanıcıya sunulur.