SharingMezzi

Sistema IoT per Sharing Mobility

Relazione Tecnica di Progetto

Souhail Nmili 20044037 - Ayoub Mahfoud 20044074 - Mathias Costantino 20043922

16 giugno 2025

Indice

1 Introduzione

SharingMezzi è un sistema completo di sharing mobility che integra tecnologie IoT, comunicazione MQTT, interfacce real-time e un'architettura modulare con Clean Architecture. Il sistema permette la gestione di biciclette e monopattini elettrici e muscolari attraverso un ecosistema di componenti interconnessi.

Il progetto implementa un sistema end-to-end, dalla simulazione di dispositivi IoT fisici fino all'interfaccia web per l'amministrazione, passando per API REST, comunicazione real-time via SignalR e un'applicazione console per utenti finali.

2 Stack Tecnologico

2.1 Backend Framework

- **ASP.NET Core 9.0** - Framework web principale
- **Entity Framework Core** - ORM per persistenza dati
- **SQLite3** - Database embedded per sviluppo
- **JWT Bearer Authentication** - Autenticazione stateless

2.2 Comunicazione Real-time

- **SignalR** - WebSocket per notifiche real-time
- **MQTTnet** - Broker MQTT integrato per IoT
- **System.Text.Json** - Serializzazione messaggi

2.3 Frontend e UI

- **HTML5/CSS3/JavaScript** - Dashboard amministrativa
- **Font Awesome** - Iconografia
- **Console Application** - Interfaccia utente terminale

2.4 Architettura e Pattern

- **Clean Architecture** - Separazione responsabilità
- **Repository Pattern** - Astrazione accesso dati
- **Dependency Injection** - Inversione controllo
- **CQRS Pattern** - Separazione command/query

3 Analisi dei Requisiti

3.1 Requisiti Funzionali

3.1.1 RF01 - Gestione Utenti

- Registrazione e autenticazione utenti
- Gestione profili (Admin, User)
- Sistema di credito
- Storico transazioni e ricariche

3.1.2 RF02 - Gestione Mezzi

- Catalogazione mezzi (biciclette, monopattini)
- Distinzione mezzi elettrici/muscolari
- Monitoraggio stato batteria per mezzi elettrici
- Gestione stati: Disponibile, In Uso, Manutenzione

3.1.3 RF03 - Sistema di Noleggio

- Prenotazione e sblocco mezzi
- Calcolo tariffe (fissa + per minuto)
- Gestione corse con inizio/fine
- Addebito automatico crediti

3.1.4 RF04 - Gestione Parcheggi

- Mapping parcheggi con geolocalizzazione
- Monitoraggio occupazione slot
- Gestione capienza e disponibilità

3.1.5 RF05 - Sistema Manutenzione

- Segnalazione problemi da utenti
- Programmazione manutenzioni
- Workflow: Programmata → In Corso → Completata
- Cambio automatico stato mezzo

3.1.6 RF06 - Comunicazione IoT

- Integrazione dispositivi tramite MQTT
- Monitoraggio real-time batterie
- Comandi sblocco/blocco remoto
- Telemetria dispositivi

3.2 Requisiti Non Funzionali

3.2.1 RNF01 - Performance

- Tempo risposta API
- Supporto più dispositivi IoT simultanei
- Aggiornamenti real-time ¡ 1 secondo

3.2.2 RNF02 - Scalabilità

- Architettura modulare espandibile
- Database ottimizzato per crescita dati
- Pattern asinconi per I/O operations

3.2.3 RNF03 - Sicurezza

- Autenticazione JWT con refresh token
- Autorizzazione basata su ruoli
- Validazione input e protezione injection

3.2.4 RNF04 - Usabilità

- Interfacce responsive multi-device
- Console user-friendly per operatori
- Dashboard amministrativa intuitiva

4 Architettura del Sistema

4.1 Clean Architecture

Il sistema implementa Clean Architecture con 4 layer principali:

4.1.1 Core Layer

- **Entities**: Modelli di dominio (Utente, Mezzo, Corsa, Parcheggio)
- **Interfaces**: Repository e servizi
- **DTOs**: Data transfer objects
- **Services**: Business logic e use cases

4.1.2 Infrastructure Layer

- **Database**: Repository implementations con EF Core
- **MQTT**: Broker e client per comunicazione IoT
- **External APIs**: Integrazioni servizi esterni

4.1.3 API Layer

- **Controllers**: Endpoint REST per operazioni CRUD
- **Hubs**: SignalR per comunicazione real-time
- **Middleware**: Autenticazione, autorizzazione, logging

4.1.4 Presentation Layer

- **Console Client**: Interfaccia terminale per utenti
- **Admin Dashboard**: Interfaccia web per amministratori

5 Architettura MQTT e IoT

5.1 Topologia MQTT

Il sistema implementa un broker MQTT integrato che gestisce la comunicazione con i dispositivi IoT simulati.

5.1.1 Topic Structure

```
sharingmezzi/
          devices/
                {mezzoId}/
                      status          # Stato dispositivo (online/
   offline)
                      battery         # Livello batteria (0-100%)
                      location        # Coordinate GPS
                      diagnostics     # Dati diagnostici
          commands/
                {mezzoId}/
                      unlock          # Comando sblocco
                      lock            # Comando blocco
                      maintenance     # Modalit  manutenzione
```

```
          notifications/
              battery_low         # Batteria sotto soglia
              maintenance_required # Richiesta manutenzione
              device_offline      # Dispositivo disconnesso
```

5.1.2 Messaggi MQTT

Battery Update Message:

```
{
    "mezzoId": "BIC001",
    "batteryLevel": 85,
    "timestamp": "2025-06-16T14:30:00Z",
    "voltage": 12.4,
    "temperature": 25.5
}
```

Status Update Message:

```
{
    "mezzoId": "SCO004",
    "status": "in_use",
    "location": {
        "latitude": 45.0703,
        "longitude": 7.6869
    },
    "timestamp": "2025-06-16T14:30:00Z"
}
```

Command Message:

```
{
    "command": "unlock",
    "mezzoId": "BIC002",
    "userId": "user123",
    "timestamp": "2025-06-16T14:30:00Z",
    "parameters": {
        "duration": 7200
    }
}
```

5.2 Simulazione Dispositivi IoT

Il sistema include un servizio che simula 6 dispositivi IoT corrispondenti ai mezzi nel database:

```
public class IoTDeviceSimulator
{
    private readonly Dictionary<string, DeviceState> _devices;

    // Simula batteria che si scarica nel tempo
    private async Task SimulateBatteryDrain(string mezzoId)
    {
        var device = _devices[mezzoId];
        if (device.IsElectric && device.Status == "in_use")
        {
            device.BatteryLevel -= Random.Next(1, 5);
```

```
            await PublishBatteryUpdate(mezzoId, device.BatteryLevel);
        }
    }

    // Simula posizione GPS durante corsa
    private async Task SimulateLocationUpdate(string mezzoId)
    {
        var device = _devices[mezzoId];
        if (device.Status == "in_use")
        {
            // Simula movimento casuale nell'area di Torino
            device.Location.Latitude += (Random.NextDouble() - 0.5) *
               0.001;
            device.Location.Longitude += (Random.NextDouble() - 0.5) *
               0.001;
            await PublishLocationUpdate(mezzoId, device.Location);
        }
    }
}
```

6 Implementazione

6.1 Setup e Avvio Sistema

6.1.1 Prerequisiti

- .NET 9.0 SDK
- Visual Studio Code
- Git per cloning repository

6.1.2 Installazione

```
# Clone repository
git clone [repository-url]
cd SharingMezzi

# Restore dependencies
dotnet restore

# Setup database
cd SharingMezzi.Api
dotnet ef database update

# Build solution
dotnet build

# Run API server
dotnet run --project SharingMezzi.Api
```

6.2 Accesso alle Interfacce

6.2.1 Admin Dashboard

- **URL**: http://localhost:5000/admin/dashboard.html
- **Login**: admin@test.com
- **Password**: admin123
- **Demo Mode**: http://localhost:5000/admin/dashboard.html#demo

6.2.2 Console Application

```
# Avvia console client
dotnet run --project SharingMezzi.Client.Console

# Credenziali utente standard
Email: mario@test.com
Password: user123

# Credenziali amministratore
Email: admin@test.com
Password: admin123
```

6.3 API Endpoints Principali

6.3.1 Autenticazione

```
# Login
POST /api/auth/login
Content-Type: application/json

{
    "email": "mario@test.com",
    "password": "user123"
}
```

6.3.2 Gestione Corse

```
# Inizia corsa
POST /api/corse/inizia
Authorization: Bearer {token}
Content-Type: application/json

{
    "mezzoId": 1
}

# Termina corsa
PUT /api/corse/{id}/termina
Authorization: Bearer {token}
Content-Type: application/json

```

```
{
    "parcheggioDestinazioneId": 2,
    "segnalazioneManutenzione": "Freni␣rumorosi"
}
```

7 Struttura Progetto

7.1 SharingMezzi.Core

Responsabilità: Business logic e entità di dominio

7.1.1 Entities

- **Utente**: Gestione account, crediti, ruoli
- **Mezzo**: Biciclette/monopattini con stato e batteria
- **Corsa**: Sessioni noleggio con calcolo costi
- **Parcheggio**: Stazioni con slot e capienza
- **SegnalazioneManutenzione**: Sistema ticket manutenzione

7.1.2 Services

- **CorsaService**: Business logic gestione corse
- **MezzoService**: Operazioni sui mezzi
- **UtenteService**: Gestione profili e crediti
- **ParcheggioService**: Gestione parcheggi e slot

7.1.3 DTOs

- Input/Output objects per API
- Validazione dati automatica
- Mappatura entità-DTO

7.2 SharingMezzi.Infrastructure

Responsabilità: Implementazioni tecniche e persistenza

7.2.1 Database

- **SharingMezziContext**: DbContext principale
- **Repositories**: Implementazioni pattern Repository
- **Migrations**: Versioning schema database
- **Seed Data**: Dati iniziali per testing

7.2.2 External Services

- Integrazioni future (payment, mapping, etc.)
- Adapter pattern per servizi esterni

7.3 SharingMezzi.Api

Responsabilità: Layer presentazione e comunicazione

7.3.1 Controllers

- **AuthController**: Login/logout/refresh
- **CorseController**: CRUD corse
- **MezziController**: Gestione mezzi
- **UserController**: Profili e crediti
- **AdminController**: Funzioni amministrative

7.3.2 Hubs SignalR

- **CorseHub**: Notifiche corse real-time
- **MezziHub**: Aggiornamenti stato mezzi
- **ParcheggiHub**: Monitoraggio parcheggi
- **IoTHub**: Comunicazione dispositivi

7.3.3 wwwroot

- **admin/**: Dashboard amministrativa completa
- File statici (CSS, JS, HTML)
- Assets e risorse

7.4 SharingMezzi.IoT

Responsabilità: Comunicazione e simulazione IoT

7.4.1 Services

- **MqttBrokerService**: Broker MQTT integrato
- **ConnectedIoTClientsService**: Gestione client connessi
- **IoTDeviceSimulator**: Simulazione dispositivi

7.4.2 Models

- **IoTMessage**: Strutture messaggi MQTT
- **DeviceState**: Stato dispositivi simulati
- **CommandMessage**: Comandi ai dispositivi

7.5 SharingMezzi.Client.Console

Responsabilità: Interfaccia utente terminale

7.5.1 Features

- Menu interattivo con 9 opzioni
- Autenticazione e gestione sessione
- Operazioni CRUD complete
- Output formattato con tabelle ASCII
- Error handling user-friendly

8 Testing

8.1 Account di Test

8.1.1 Utenti Predefiniti

Email	Password	Ruolo	Credito
admin@test.com	admin123	Admin	€50.00
mario@test.com	user123	User	€25.00
lucia@test.com	user123	User	€15.00

Tabella 1: Account di Test Predefiniti

8.1.2 Dati Test

- **6 Mezzi**: 2 bici muscolari, 2 bici elettriche, 2 monopattini
- **3 Parcheggi**: Centro Storico, Politecnico, Porta Nuova
- **IoT Devices**: 6 dispositivi simulati connessi
- **Slot Parcheggio**: 75 slot totali distribuiti

8.2 Scenari di Test

8.2.1 Console Client

1. Login con mario@test.com
2. Visualizza mezzi disponibili
3. Inizia corsa con bici elettrica
4. Termina corsa con segnalazione manutenzione
5. Verifica addebito credito
6. Ricarica credito €10
7. Visualizza storico corse

8.2.2 Admin Dashboard

1. Accesso con admin@test.com
2. Monitoraggio statistiche real-time
3. Verifica connessione 6 dispositivi IoT
4. Gestione manutenzioni programmate
5. Monitoraggio batterie mezzi elettrici
6. Test notifiche real-time

9 Sviluppi Futuri

9.1 Miglioramenti Tecnici

9.1.1 Scalabilità

- **Database**: Migrazione a PostgreSQL/SQL Server
- **Caching**: Implementazione Redis per performance

9.1.2 IoT e Hardware

- **Device Reali**: Integrazione con hardware fisico
- **GPS Tracking**: Localizzazione real-time avanzata

9.2 Funzionalità Business

9.2.1 Mobile App

- **React Native/Flutter**: App cross-platform
- **QR Code Scanner**: Sblocco mezzi tramite QR
- **Maps Integration**: Google Maps

9.2.2 Payment Gateway

- **Stripe/PayPal**: Integrazione pagamenti reali
- **Wallet**: Sistema wallet digitale interno
- **Subscription**: Piani abbonamento mensili

A Appendice A - Struttura File Completa

```
SharingMezzi/
        SharingMezzi.Api/
            Controllers/
                AdminController.cs
                AuthController.cs
                CorseController.cs
                MezziController.cs
                UserController.cs
            Hubs/
                CorseHub.cs
                IoTHub.cs
                MezziHub.cs
                ParcheggiHub.cs
            wwwroot/admin/
                dashboard.html
                css/admin-dashboard.css
                js/admin-dashboard.js
            Program.cs
        SharingMezzi.Core/
            Entities/
                Utente.cs
                Mezzo.cs
                Corsa.cs
                Parcheggio.cs
                SegnalazioneManutenzione.cs
            Services/
                CorsaService.cs
                MezzoService.cs
                UtenteService.cs
            DTOs/
            Interfaces/
        SharingMezzi.Infrastructure/
            Database/
                SharingMezziContext.cs
                Repositories/
        SharingMezzi.IoT/
            Services/
                MqttBrokerService.cs
                ConnectedIoTClientsService.cs
        SharingMezzi.Client.Console/
            Program.cs
```

B Appendice B - Comandi Utili

```
# Build completo
dotnet build

# Test API
curl -X POST http://localhost:5000/api/auth/login \
     -H "Content-Type: application/json" \
     -d '{"email":"mario@test.com","password":"user123"}'

# Monitoraggio logs
tail -f SharingMezzi.Api/logs/app.log

# Database reset
dotnet ef database drop
dotnet ef database update

# Console client
dotnet run --project SharingMezzi.Client.Console
```