

# APLICAȚIE WEB PENTRU GESTIONAREA ȘI LOCALIZAREA INSTRUMENTELOR MEDICALE

**Autori: Oana-Cristina GHICAJANU <sup>1</sup>**

[ghicajanuoana@gmail.com](mailto:ghicajanuoana@gmail.com)

**Coordonator: Șef lucr. dr. ing. Vali SÎRB <sup>2</sup>**

<sup>1</sup> *Universitatea din Petroșani, IME, Calculatoare , anul IV*

<sup>2</sup> *Universitatea din Petroșani, IME, Departamentul de Automatică, Calculatoare, Inginerie Electrică și Energetică*

## Rezumat

Lucrarea reprezintă o aplicație web pentru gestionarea și localizarea instrumentelor medicale într-un mod securizat și ușor de folosit. Această aplicație este realizată în .NET pe partea de backend, și în Angular pe partea de frontend.

## Cuvinte cheie

Securitate, localizare, domeniu medical, gestionare, monitorizare.

### 1. Introducere

Aplicația este concepută pentru firmele care dețin dispozitive medicale și au nevoie de monitorizarea și localizarea acestora, asigurând o gestionare optimă. Cu ajutorul acestei platforme, firma beneficiază de numeroase funcționalități: identificarea rapidă a fiecărui dispozitiv la locația respectivă, adăugarea, modificarea și verificarea detaliilor și informațiilor despre acel instrument, se pot înregistra notițe de mentenanță pentru instrumentele care necesită reparații.

### 2. Descrierea aplicației

Proiectul are o pagină principală de log in, sign up și acces admin. La partea de sign up, după ce utilizatorul își creează un cont, administratorul aplicației va primi o notificare pe email cu detaliile despre înregistrarea noului utilizator. Am folosit EmailJS pentru această funcționalitate.

La login admin, doar admin-ul are acces la logare, unde va putea vizualiza lista de utilizatori, având și funcțiile de adăugarea, ștergerea sau modificarea utilizatorilor. De asemenea, din motive de securizare (de exemplu: verificare că nu este un robot care vrea să facă această acțiune), când administratorul dorește să șteargă un anumit utilizator, va fi redirecționat către o pagină unde va trebui să introducă un cod unic de verificare. Acest cod criptat îl va primi pe email. După ce va scrie codul corect, ștergerea utilizatorului este efectuată.

După logare, utilizatorii pot intra în aplicație. Avem un dashboard cu o mapă unde sunt pin-urile cu locațiile în care se află dispozitivele medicale. La partea de dashboard, avem mai multe butoane pentru accesare: Location, Devices, Add Device, Device Types (având incorporat și un modal de add device type), Maintenance, Threshold.

De exemplu, la componenta Device, putem să îi vedem caracteristicile: denumirea, locația în care se află, tipul de dispozitiv, versiunea software, versiunea hardware, email-ul pentru contact, imaginea device-ului și descrierea. După ce avem lista de device-uri, putem să o descărcăm și sub forma unui document CSV.

Un aspect important legat de acest proiect este faptul că aplicația este gândită într-un mod securizat din mai multe puncte de vedere. Câteva exemple:

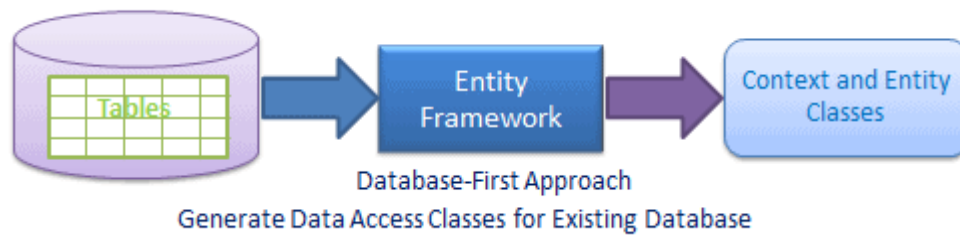
- Nu putem efectua anumite operații care ar pune în pericol datele despre dispozitivele medicale, astfel componentele sunt legate între ele în baza de date; dacă dorim să ștergem locația unui instrument medical care este activ și în folosință, nu vom putea face această acțiune
- Denumirea instrumentelor trebuie să fie unice, astfel avem restricție la duplicitate
- La adăugarea unui dispozitiv, anumite câmpuri sunt obligatoriu de completat, altfel nu se poate efectua această operație

### 3. Tehnologii folosite

Aplicația este concepută în 2 părți: partea de backend (server) și partea de frontend (client). Pe partea de backend am folosit tehnologia ASP.NET Web API, iar pe partea de frontend am folosit Angular.

Cu ajutorul ASP.NET, am putut folosi servicii web RESTful, ce asigură comunicarea între server și client prin intermediul protocolului HTTP (Hyper Text Transfer Protocol). Operațiile folosite sunt de tip GET, POST, PUT și DELETE. Aceste operații sunt de tip request, astfel primim și răspunsurile sub forma de "status code": 200 (OK), 400 (Bad Request), 404 (Not Found), 500 (Internal Server Error). De asemenea, putem folosi status-uri personalizate pentru diferite cazuri aparte.

Am optat pentru utilizarea .NET deoarece avem framework-ul EF Core (Entity Framework) folosit pentru gestionarea interacțiunilor cu baza de date, (fig. 1).



**Fig. 1.** *EF Core*

Arhitectura pe partea de backend este împărțită în 3 părți: Data Access, Business Logic și Web API. În Data Access Layer se află pachete cu clasele specifice:

- În pachetul Models, avem clasele model (fig. 2)
- În pachetul Repository, avem clasele ce conțin metode pentru a accesa datele din baza de date
- În pachetul Interfaces, se află clasele interfață pentru clasele repository

Aici mai avem și clasa DataContext cu câmpurile de tip DbSet, reprezentând tabelele din baza de date, (fig. 3)

```

namespace DataAccessLayer.Models
{
    public class Maintenance
    {
        public int Id { get; set; }
        public Device Device { get; set; }
        public int DeviceId { get; set; }
        public string? Description { get; set; }
        public string? Outcome { get; set; }
        public Status Status { get; set; }
        public DateTime ScheduledDate { get; set; }
        public DateTime? ActualDate { get; set; }
        public DateTime CreatedAt { get; set; }
        public string CreatedBy { get; set; }
    }
}

```

**Fig. 2.** *Clasa model*

```

public DbSet<Device> Devices { get; set; }

public DbSet<DeviceType> DeviceTypes { get; set; }

public DbSet<Location> Locations { get; set; }

public DbSet<DeviceReadingType> DeviceReadingTypes { get; set; }

public DbSet<Threshold> Thresholds { get; set; }

public DbSet<User> Users { get; set; }

public DbSet<Role> Roles { get; set; }

public DbSet<Maintenance> Maintenances { get; set; }

```

**Fig. 3.** *DbSet*

În Business Logic Layer, avem pachetele:

- DTOs, unde folosim doar datele necesare ale tabelor din baza de date
- Services, unde ne folosim de clasele repository pentru a construi partea de logica

În pachetul Web API, avem Controllers unde se gestionează cererile HTTP, se procesează datele, apoi se

returnează răspunsurile.

Pentru partea de frontend, am ales să folosesc Angular deoarece este un framework pentru construirea de aplicații web scalabile. Ca și platformă, Angular include:

- Structură secțională: Angular promovează dezvoltarea bazată pe componente, în care interfața utilizatorului este împărțită în componente reutilizabile și independente. Aceasta facilitează structurarea și organizarea codului, permițând dezvoltatorilor să creeze și să gestioneze interfețe complexe prin compunerea unor componente simple
- Data binding: Angular integrează un sistem puternic de Dependency Injection, care facilitează gestionarea dependențelor între diferitele componente și servicii din aplicație. Acesta promovează modularitatea și reutilizarea codului, permițând dezvoltatorilor să scrie componente și servicii independente și ușor de testat
- Routing: Angular include un modul de rutare integrat, care permite dezvoltatorilor să gestioneze navigarea între diferitele pagini și secțiuni ale aplicației
- HTTP Client: Acest serviciu este disponibil ca o clasă injectabilă, cu metode pentru a efectua solicitări HTTP. Modulul HTTP Client integrat facilitează comunicarea cu serverul, efectuarea cererilor HTTP, gestionarea răspunsurilor și tratarea erorilor

#### 4. Descrierea aplicației practice

Limbajele de programare folosite sunt:

- În backend: C#
- În frontend: TypeScript, HTML, CSS

Fiind un proiect complex, voi alege o singură componentă și câteva clase ca și exemplificare.

Clasa model Device are câmpurile:

```
public class Device
{
    public int DeviceId { get; set; }
    public string SerialNumber { get; set; }
    public string Name { get; set; }
    public string? SoftwareVersion { get; set; }
    public string? FirmwareVersion { get; set; }
    public DateTime? LastUploadTime { get; set; }
    public string? Description { get; set; }
    public string? Alias { get; set; }
    public int LocationId { get; set; }
    public Location Location { get; set; }
    public int DeviceTypeId { get; set; }
    public DeviceType DeviceType { get; set; }
    public byte[]? ImageBytes { get; set; }
    public string? Emails { get; set; }
    public DateTime? LastPingedTs { get; set; }
}
```

În clasa DeviceRepository, avem operațiile CRUD: Get, Create, Update și Delete. Voi alege una dintre metode, aceasta fiind cea de afișare a tuturor instrumentelor medicale:

```
public async Task<IEnumerable<Device>> GetAllDevicesAsync()
{
    var devices = await _dataContext.Devices
        .Include(d => d.Location)
        .Include(d => d.DeviceType)
        .ToListAsync();
    return devices;
}
```

Vom observa că în această metodă se creează legătura între tabelele Devices, Location și DeviceType, folosind metoda Include de tip LINQ.

Clasa DeviceService apelează metoda de get din clasa repository:

```
public async Task<IEnumerable<DeviceDto>> GetAllDevicesAsync()
{
    var allDevicesFromDb = await _deviceRepository.GetAllDevicesAsync();
    var deviceDtos = new List<DeviceDto>();
}
```

```

        foreach (var device in allDevicesFromDb)
        {
            var deviceDto = ConvertDeviceToDto(device);
            deviceDtos.Add(deviceDto);
        }
        return deviceDtos;
    }
}

```

În clasa DeviceController, declarăm metoda HTTP de tip GET, și atributul Route pentru a specifica ruta "getAllDevices" atunci când va fi accesată această metoda:

```

[HttpGet]
[Route("getAllDevices")]
public async Task<IActionResult> GetAllDevicesAsync()
{
    try
    {
        _logger.LogInformation($"GetAllDevices");
        var deviceDtos = await _deviceService.GetAllDevicesAsync();

        return Ok(deviceDtos);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex.Message);
        return BadRequest(ex);
    }
}

```

La partea de frontend, serviciile sunt cele mai importante de menționat deoarece comunică cu server-ul și sunt folosite în componentele specifice.

În clasa device service, avem metoda pentru a obține lista de dispozitive de la server, după care va fi apelată în componenta device:

```

getAllDevices() {
    return this.http.get<Devices[]>(`${this.apiUrl}/getAllDevices`, header)
}

```

## 5. Concluzii

Aplicația este dedicată gestionării și localizării instrumentelor medicale, reprezentând o soluție inovatoare și sigură. Folosind tehnologia .NET, am dezvoltat o platformă optimă ce conține numeroase funcționalități eficiente de tip GET, POST, PUT și DELETE. Cu ajutorul framework-ului Angular, interfața este ușor de utilizat și user-friendly, astfel încât utilizatorii pot accesa fără dificultate platforma.

## 6. Bibliografie

1. <https://developer.mozilla.org/en-US/docs/Web/HTTP>
2. <https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>
3. <https://angular.io/guide/what-is-angular>
4. <https://www.cybersuccess.biz/angular-features/>