# Pannonhalmi Főapátság Szegedi SOB Technikuma

Pannonhalmi Főapátság Szegedi SOB Technikuma
6727 Szeged, Lidicei tér 1.
+36/62 424-927 • titkarsag.sob@szbi.hu • www.sobszeged.hu

# Munkarendszer

*Projektmunka-vizsgaremek*

Szakképzés neve és kódja: Szoftverfejlesztő és -tesztelő (506131203)

Témavezető: Rédai Dávid

Készítők: Adamovits Otília Alexandra

Molnár Szabolcs Szilveszter

Szeged, 2025.

# Tartalomjegyzék

# 1. Bevezetés

A Munkarendszer nevű program létrehozásának célja, egy olyan digitális megoldás megvalósítása volt, ami hatékonyan nyilvántartja a dolgozók munkaidejét, a kilépési és belépési időket, és rögzíthető benne beosztás, majd mindezen adatok megléte esetén különböző kimutatásokat tud elkészíteni. A rendszer fő feladata, hogy három meghatározott jogkör szerint a munkaórák, a ledolgozott és a tervezett munkaidők precízen nyilvántartva legyenek, valamint a havi és napi szintű összesítések és kimutatások pontos tájékoztatást adjanak a rendszer használóinak. A megvalósítás során ASP.NET Core alapú Web API backend készült C# programozási nyelv használatával, amihez egy Vite+SvelteKit frontend felület csatlakozik JavaScript nyelven megírva, a két fő rendszert egy .NET MAUI mobil alkalmazás is kiegészíti.

A projekt témaválasztásának főbb szempontjai több tényezőre vezethető vissza. Elsősorban olyan a mai modern életben olyan könnyen használható és kiemelt jelentőségű rendszert szerettünk volna létrehozni, amely megfelelően támogatja a munkaerőpiacot. A jelenkor vállalkozásai számára kiemelt jelentőségű, hogy hatékony, pontos, precíz munkaidő-nyilvántartó rendszerrel rendelkezzenek, hiszen az automatizált megoldások megkönnyítik a mindennapokat. Másodsorban a közös szakmai munkánk során az is fontos szempontként került mérlegelésre, hogy a backend- és frontend fejlesztés, illetve az adatbáziskezelés mélyebb megismerésével szerettünk volna gyakorlati tapasztalatokat gyűjteni.

A program fejlesztése során célul tűztük ki, hogy egy megfelelő biztonsági rendszerrel ellátott és megbízhatóan működő alkalmazást hozzunk létre, amely támogatja a beléptetési és kijelentkeztetési folyamatokat. Mindezeken kívül lehetőséget teremt az adminisztrátorok és managerek számára a dolgozók adatainak jogkör alapú kezelésére. A projekt során külön figyelmet fordítottunk a jogosultságkezelésre és a felhasználói hitelesítés megvalósítására is, amelyhez JWT (JSON Web Token) alapú autentikációs rendszert alkalmaztunk, továbbá a jelszavak biztonságos mentése érdekében BCrypt alapú jelszó hashelést.

A projektet gyakorlati szempontból is relevánsnak tartjuk, hiszen egy ilyen rendszer alkalmazható lehet akár kis- és középvállalkozások, akár nagyobb szervezetek esetében is, ahol fontos a munkaidő pontos rögzítése és a munkavállalók munkaidejének átlátható nyilvántartása. Ezen kívül a projekt során megszerzett tudás és tapasztalatok hasznos alapot biztosítanak számunkra a jövőbeli backend, frontend és mobilos alkalmazások fejlesztésében is.

A választott projekt jól illeszkedik a szoftverfejlesztői szakma gyakorlati követelményeihez, mivel több, valós környezetben is alkalmazható technológiát és architektúrát ötvöz: REST API használatát, adatbázis-kezelést (MYSQL), platformfüggetlen mobilalkalmazás-fejlesztést (MAUI), frontend fejlesztést (SvelteKit), valamint a frontend- és backend közötti kommunikáció megteremtését.

# 2. Fejlesztői dokumentáció

## 2.1 Fejlesztői környezet ismertetése

A dolgozói beléptetőrendszer fejlesztését egy **ASP.NET Core Web API 8.0** alapú backend és egy **SvelteKit** frontend környezetben valósítottuk meg. A backendhez **C#** nyelvet használtunk, míg a frontend fejlesztés **JavaScript** alapokon történt.

A választásunkat az indokolta, hogy az ASP.NET Core stabil, ipari szintű keretrendszer, amely kiválóan támogatja a REST API-k gyors és biztonságos fejlesztését, beépített autentikációs és jogosultságkezelési lehetőségekkel. A **SvelteKit** pedig modern, gyors és egyszerűen testreszabható frontend keretrendszer, amely jól integrálható REST API-khoz.

A fejlesztés során az alábbi környezetet és eszközöket használtuk:

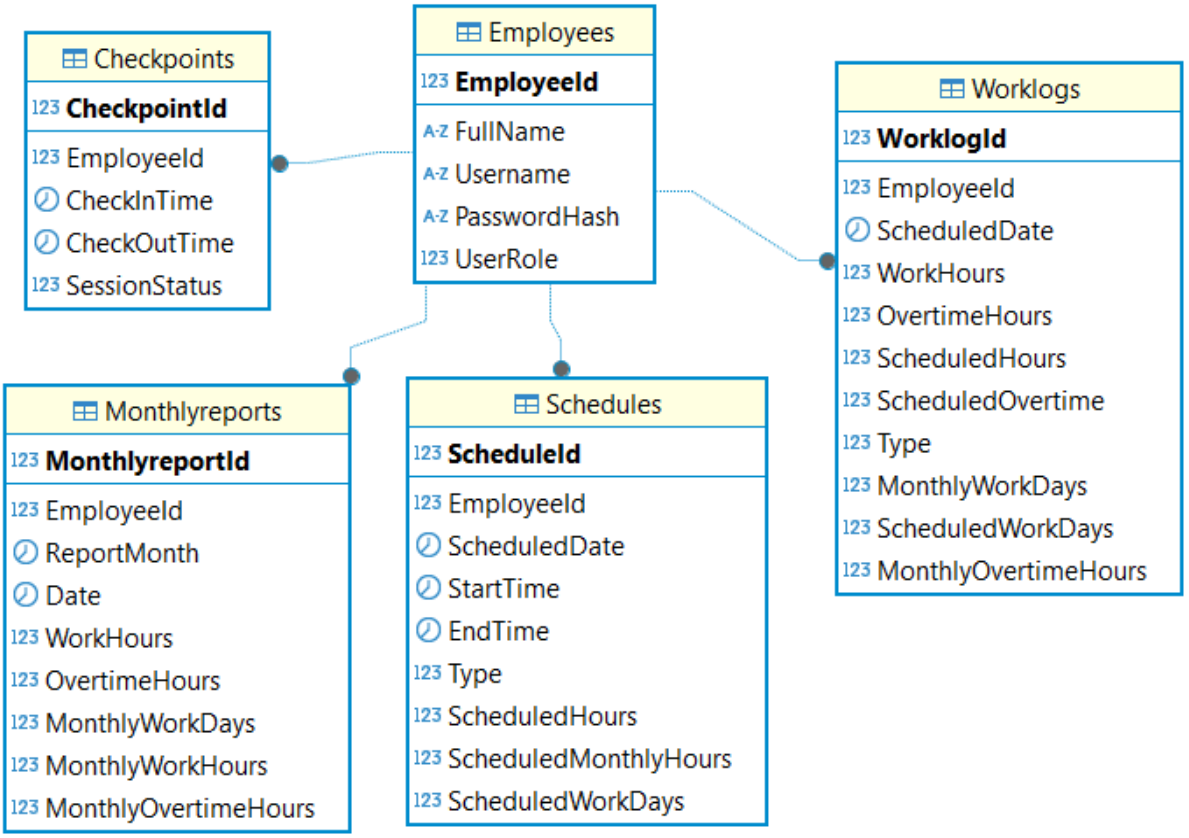
- **Visual Studio 2022** (ASP.NET Core API fejlesztés, .NET Maui mobil fejlesztés)
- **VS Code** (SvelteKit frontend fejlesztés)
- **Swagger** (böngészőn keresztül, API tesztelés)
- **MYSQL** adatbázis (Aiven, felhő alapú, komplex fejlesztési környezethez)
- **Git** verziókezeléshez (GitHub tároló)
- Tesztelés **mobil**készüléken: Android 11/12-es rendszerű telefon

A végleges adatbázis üzemeltetését **Aiven** felhőalapú szolgáltatásban terveztük meg, amely biztosítja a skálázhatóságot és a megbízhatóságot. A szerver oldali komponens render.com, a frontend oldali rész pedig netlify,com felhőalapú szolgáltatókon keresztül valósul meg az online térben. Fejlesztői környezetben azonban a böngészőben localhost használatával is elérhetőek a program részei.

## 2.2 Adatszerkezet bemutatása

Az adatbázis az **Entity Framework Core** használatának segítségével jön létre, mely egy széleskörben használt **ORM** (Object-Relational Mapping) keretrendszer. Az adatok úgy kerülnek implementálásra, ahogyan az **AppDbContext.cs** fájlban definiálva van. A fentiek alapján 5 különböző táblát generáltunk a Models osztály létrehozása után.

- **Checpoints** tábla a Checkpoint.cs model-ből
- **Employees** tábla az Employee.cs model-ből
- **Monthlyreports** tábla a Monthlyreport.cs model-ből
- **Schedules** tábla a Schedule.cs model-ből
- **Worklogs** tábla a Worklog.cs model-ből



A táblázatban az elsődleges azonosító, vagyis a **Primary Key** az EmployeeId, amely egy három számjegyű kód, és annak érdekében, hogy ez a teljes projektben egységes lehessen, már a Models mappában található osztályok szintjén beállításra került. A táblák az elsődleges kulcson keresztül kapcsolódnak, hiszen a későbbi metódusokban az EmployeeId gyakran felhasználásra kerül azonosítási célból.

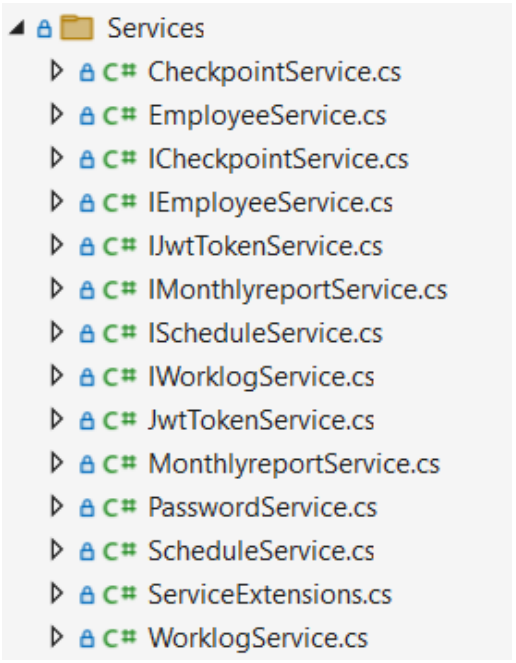
```
[Key]
[Range(100, 999, ErrorMessage = "Az ID-nak 100 és 999 között kell lennie.")]
23 references
public int EmployeeId { get; set; }
```

Az adatszerkezet kialakításához és tárolásához fontos megemlíteni az enum használat a Model osztályokban, a UserRole (Admin, Manager, Employee), SessionStatus (Active, Inactive) és ScheduleType (Shift, Overtime, DayOff, PaidTimeOff, SickLeave) esetében szöveges érték kerül megadásra enum-ként, ám a táblában ez számként tárolódik, az átalakításért a program.cs fájlban a JsonStringEnumConverter() felel.

## 2.3 A backend felépítésének és működésének bemutatása

A backend oldali komponens egy **C# programozási nyelven megírt ASP.NET Core alapú Web API 8.0** alapra épül. A továbbiakban a program felépítését és főbb szerkezeti elemeit szeretnénk ismertetni.

A Models mappában található osztályok az adatszerkezet ismertetése fejezetben megtörtént, így a továbbiakban a **DTO** (Data Transfer Object) illetve az ehhez használt **MappingProfile.cs** jelentőségével szeretném folytatni. Az előbb említett fájlt a projekthez **Automapper** hozzáadása után készítettük el a Models és DTOs mappákban szereplő osztályok közötti kétirányú leképezés megteremtése érdekében. A projektben a Service és Controller rétegben is használatra kerül, ám néhol manuális leképezéssel is történik az adatok konvertálása.




A **Services mappa** rendkívül terjedelmes tartalommal rendelkezik, az üzleti logika rétege, ahol az adatbázis műveletek és specifikus működéshez szükséges logikák vannak elhelyezve.

A Service mappában szereplő fájlok a **CRUD műveleteket** tartalmazzák, melyek a Create-Post, Read-Get, Update-Put, Delete-Delete metódusokat foglalják magukba. Ezeket a metódusokat a Controller rétegben szabályozott módon lehet használni a programban.

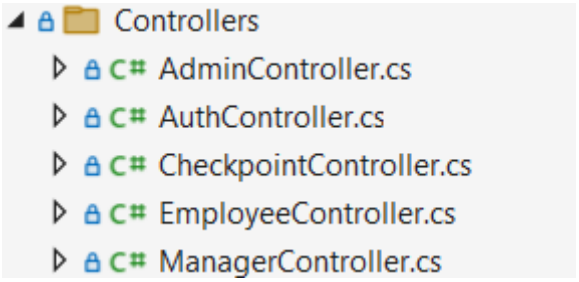
Biztonsági szempontból a **JwtTokenService.cs** fájl rendkívül fontos, hiszen itt történik a tokenek létrehozásának, validálásának, érvénytelenítésének implementálása, melyet az appsettings.json használja a tokenekre. A **PasswordService.cs** fájl a jelszó beállítására vonatkozó biztonsági műveleteket a hash-olást, ellenőrzést és validációt tartalmazza.

```
// Jelszó hash-olása
4 references
public string HashPassword(string password)
{
    ValidatePassword(password);
    return BCrypt.Net.BCrypt.HashPassword(password);
}
```

A Service réteg működését a **Helper** mappában található fájlok segítik, itt különböző Calculator funkciók kerülnek megadásra, melyeket a kimutatások számítására használnak az egyes Service fájlok. A számítási műveletekre a napi és havi munkaórák, a beosztásból a tervezett munkaórák és a túlórák pontos megjelenítésére szolgálnak.

```
// Havi ledolgozott órák Monthlyreportból
0 references
public static decimal CalculateMonthlyWorkHours(IEnumerable<Monthlyreport> reports, DateOnly reportMonth)
{
    if (reports == null)
        throw new ArgumentNullException(nameof(reports));

    decimal totalHours = 0;
    foreach (var report in reports)
    {
        if (report.ReportMonth.Month == reportMonth.Month &&
            report.ReportMonth.Year == reportMonth.Year)
        {
            totalHours += report.WorkHours;
        }
    }
    return totalHours;
```

A fent ismertetett **Service réteg működését** legjobban a Controller-eken keresztül lehet ismertetni, hiszen a gyakorlati használatuk és a metódusok meghívása itt történik. A képen látható a 4 Controller fájl, amelynek a kialakítási logikája követi a jogkörök és jogkör nélküli műveletek kialakítását.




A **három jogkör az Admin, Manager és Employee** más-más műveleteket hajthat végre, míg az Admin-nak teljeskörű hozzáférése van a metódusok használatához, hiszen az AdminController.cs-ben minden szükséges művelet hozzáadásra kerül. A Manager csak korlátozott módosítási és hozzáadási lehetőségeket kapott, az Employee pedig kizárólag a saját adataihoz férhet hozzá. A jogkörön kívül helyezkedik el a Checkpoint, mert a munkaidő rögzítéséhez nem kell bejelentkezés,

hanem az EmployeeId alapján történik az adat regisztrálása az adatbázisba. A jogkör alapú azonosítási logikát a program.cs deklarálja.

```
//Authorization beállítása.
builder.Services.AddAuthorization(options =>
{
    // Alap authentikációs policy
    options.AddPolicy("Authenticated", new AuthorizationPolicyBuilder()
        .AddAuthenticationSchemes(JwtBearerDefaults.AuthenticationScheme)
        .RequireAuthenticatedUser()
        .Build());

    // Szerepkör alapú policy-k
    options.AddPolicy("AdminOnly", policy =>
        policy.RequireAuthenticatedUser()
              .RequireRole("Admin"));

    options.AddPolicy("ManagerOnly", policy =>
        policy.RequireAuthenticatedUser()
              .RequireRole("Manager"));

    options.AddPolicy("ManagerOrAbove", policy =>
        policy.RequireAuthenticatedUser()
              .RequireRole("Admin", "Manager"));

    options.AddPolicy("EmployeeOrAbove", policy =>
        policy.RequireAuthenticatedUser()
              .RequireRole("Admin", "Manager", "Employee"));
});
```

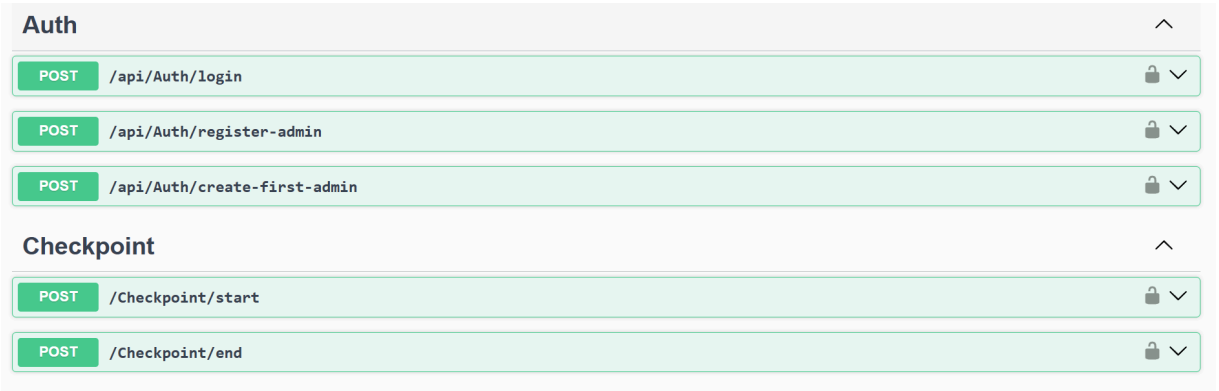
A projekt két különböző **környezet**ben is képes működni, így két apsettings.json fájlt tartalmaz az egyik a fejlesztői a másik a termék oldali komponenseket tárolja. A projektben emiatt a **CORS** (Cross-Origin Resource Sharing) konfigurációja is kétféle frontend elérést biztosít, localhost-on és online Netlifyon.

## 2.4 Backend tesztelés Swagger használatával

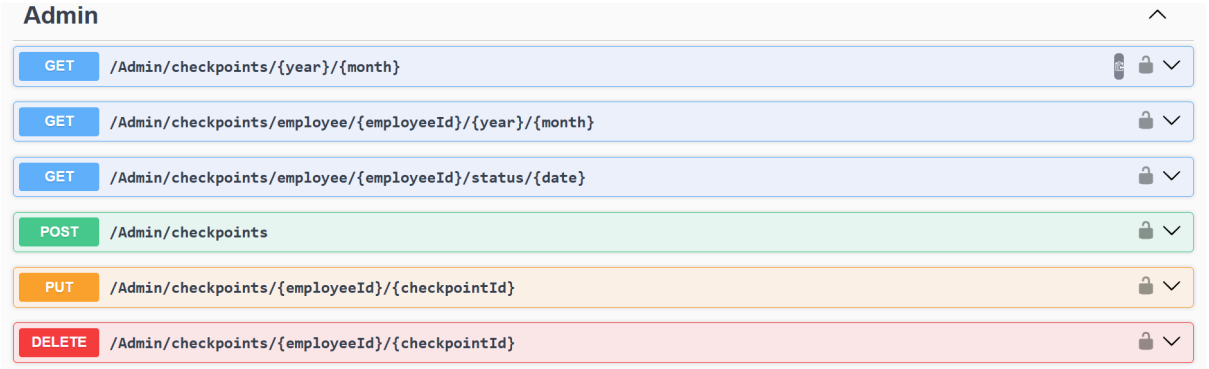
A **Swagger** tesztelői környezet kialakítása a backend oldali komponensen történt, ahol a megfelelő NuGet package telepítése után két részletben kellett elvégezni a beállítását.

```
// Swagger konfiguráció
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c => {
    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme {
        Description = "JWT Authorization header using the Bearer scheme",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer"
    });

    c.AddSecurityRequirement(new OpenApiSecurityRequirement {
        {
            new OpenApiSecurityScheme {
                Reference = new OpenApiReference {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            Array.Empty<string>()
        }
    });
});
var app = builder.Build();

// Swagger engedélyezése Production környezetben is
if (app.Environment.IsDevelopment() || app.Environment.IsProduction())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

Az **Auth** rész 3 metódust tartalmaz, az első a bejelentkezéshez szükséges, a második regisztrációhoz, a harmadik pedig a fejlesztői környezetben lehetővé teszi az első admin létrehozását. A **Swagger** használatával az első Admin jogkörrel rendelkező dolgozó hozzáadása sikeres volt, és második Admin jogkörrel rendelkező dolgozót nem enged hozzáadni. A /login végponton a bejelentkezés sikeres, ha a megadott adatok megfelelnek annak, amivel regisztrálva lett a dolgozó, és helyesen visszaadja a tokent is a Response body részen.

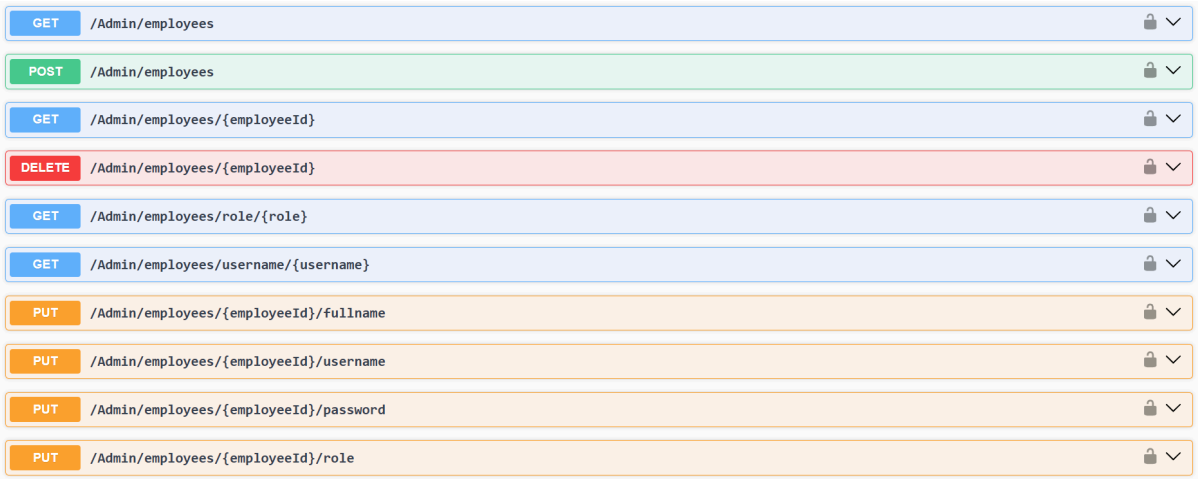



A **Checkpoint** végponton a /start és /end a munka megkezdésére és befejezésére szolgál. Tesztelve kizárólag a három számjegyű EmployeeId helyes megadása esetén kezdődik el a munkaidő számítása, kétszer nem lehetséges elkezdeni a munkát, és a munka befejezéséhez szükséges a kezdési időponttal rendelkeznie az adott dolgozónak.

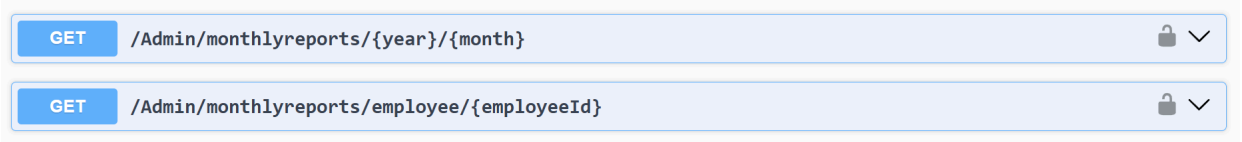



Az Admin végponton a Checkpoint-okhoz tartozó metódusokat a Service rétegben meghatározott módon tudja elvégezni kizárólag Admin jogkörrel rendelkező dolgozó. A Checkpoint lekérési lehetőségei a megadott adatokkal sikeresen lefut, azonban helytelen adatok megadása esetén hibaüzenetet ad válaszként. A létrehozás, módosítás és törlés is a várt módon hibátlanul működik.

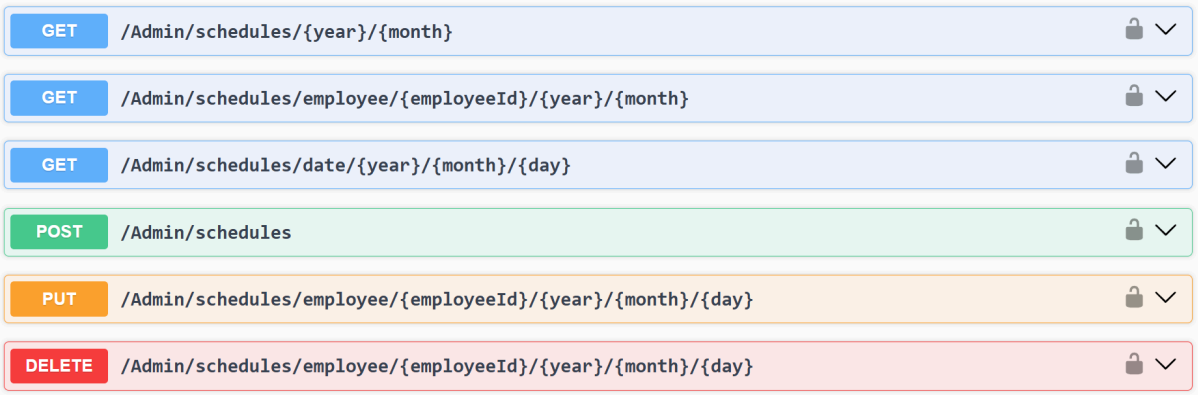
A következőkben a dolgozókkal kapcsolatos műveletek érhetőek el, tesztelés során az összes lekérés, módosítás, létrehozás és törlés metódus a várt módon jól működött.



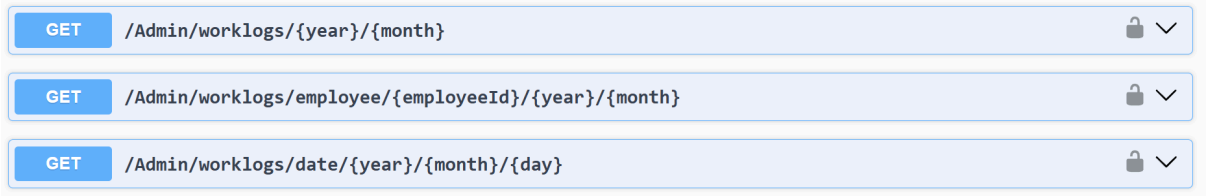

A **havi kimutatások**, lekérések esetén a visszakapott és a számolt értékek is megfelelnek a metódusok által elvárt eredményekkel, tehát a napi és havi összegek is megjelennek, az EmployeeId-ra való szűrés is jól működik, nem megfelelő adatok esetén pedig hibaüzenet érkezik.




A **beosztás lekérései**, létrehozása, módosítása, törlése is jól működik, van lehetőség kezdési és befejezési időpont esetén 00:00 értéket megadni és ilyen módon pl DayOff típust kiválasztani, a kötelezően kitöltendő elemeket elvárja a metódus és hibaüzenetet küld hibás kitöltés esetén.



A **Worklog** egy munkaidőket és beosztás adatokat tartalmazó kimutatás, amely megfelelően lekéri az adatokat a MonthlyReport és Schedule táblákból, ha helyes a beviteli érték, és hibaüzenetet ír ki hibás kitöltés esetén.



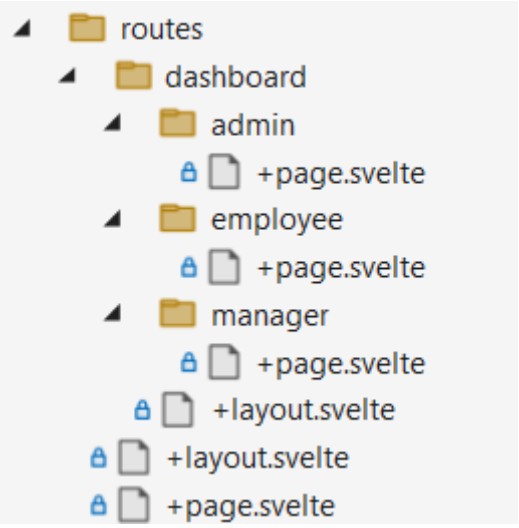

A **Manager jogkör**höz tartozó végpontok átfednek az Admin-nal, hiszen ugyanazokat a metódusokat használja. A /schedules végpontok teljesen egyezőek, a többi végpont esetében módosításra, létrehozásra és törlésre nincs lehetősége. A jelszó módosítására szigorúbb szabályok mellett van lehetősége, de kizárólag a sajátját módosíthatja. A végpontok egyesével történő tesztelése során hibát nem tapasztaltunk.

Az **Employee jogkör**höz tartozó metódusok csak a saját EmployeeId-hoz tartozó lekéréseket tartalmazzák, illetve a saját jelszavát szigorú szabályok mentén van lehetősége módosítani.

## 2.5 A frontend felépítésének és működésének bemutatása

A frontend oldali komponens **Vite+SvelteKit fejlesztői környezetben JavaScript nyelv**et alkalmazva lett elkészítve. A továbbiakban a program szerkezeti elemeit és működését szeretnénk bemutatni.

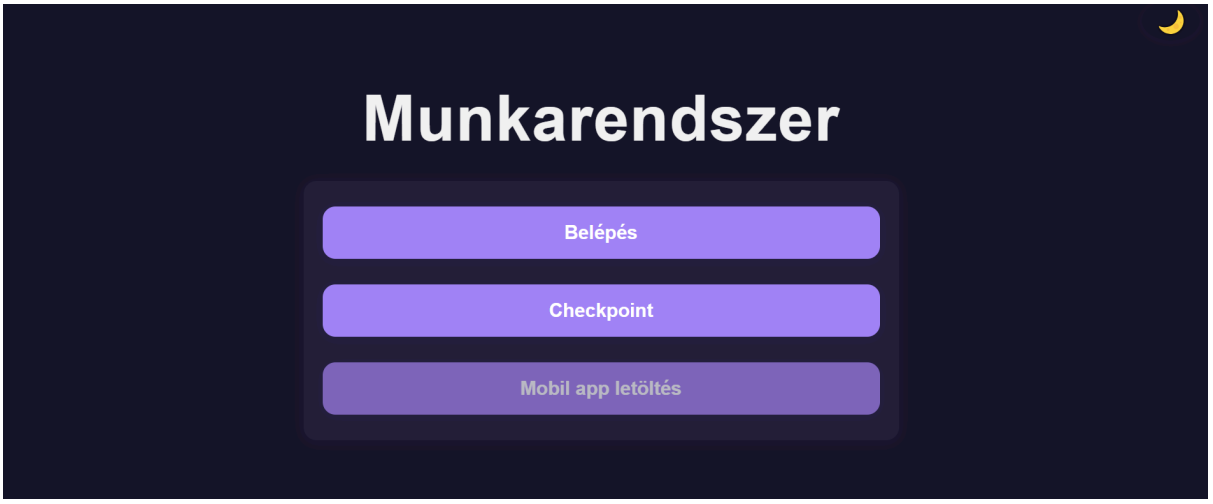
A projekt megalkotásakor az elsődleges szempont a backend oldali logika tükrözése volt a frontend oldali struktúrális felépítésben, hogy a jogkör alapú azonosítási és működési metodika megfelelően megvalósulhasson. Mindezekből kiindulva egy főoldal és három aloldal került kialakításra a /routes mappában a képen látható módon.

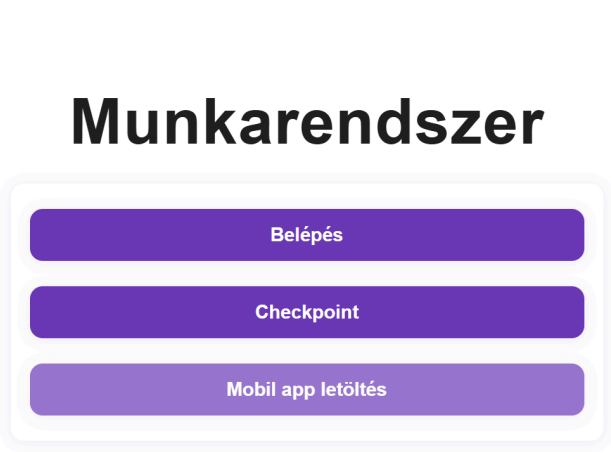



A külső megjelenés kialakítása során egy közös **+layout.svelte** és egy a /dashboard mappa tartalmára vonatkozó +layout.svelte fájl került létrehozásra, továbbá a stíluselemek egységesítése érdekében egy **global.css** fájlt is készítettünk.

Az oldal megjelenítésekor szerettünk volna **sötét-világos témát** kialakítani, hogy a felhasználói élmény minél jobb lehessen, ehhez először a themeStore.js-ben a téma inicializálását kellett elvégezni, ami meghatározza az elsődleges téma kiválasztását. A tárolása a themeStore.js-ben történik, és így a local storage alapján tudja a módosítási szándékot tárolni. A stílusát, a css változókat, a színeket a global.css

szelektorok használatával dark/light határozza meg. A ThemeToggle.svelte pedig a témaváltó gombot kezeli.



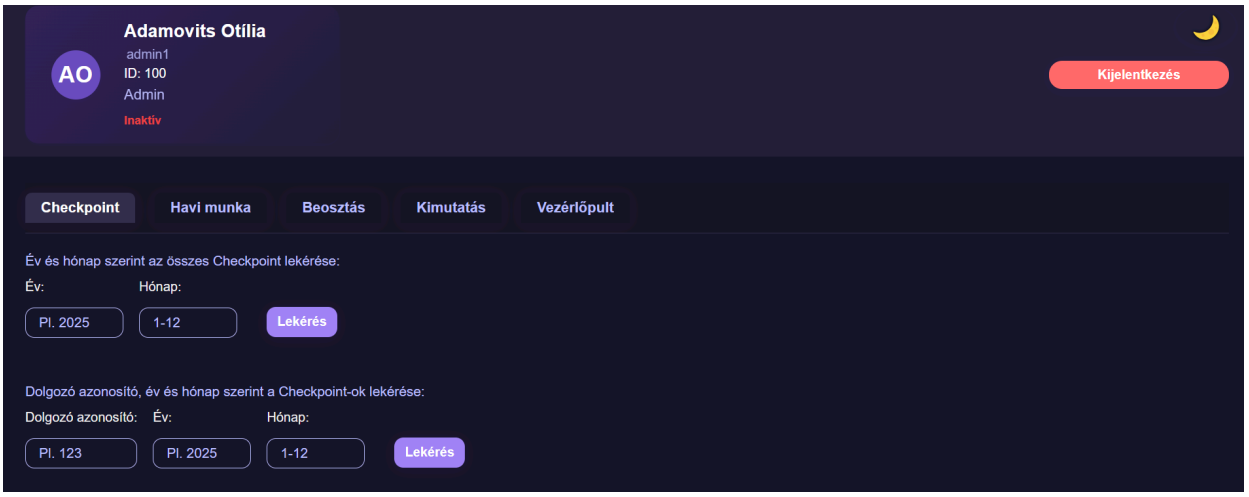

A fenti képen a téma váltó gomb megfelelő működése látható, illetve a főoldal megjelenése, a /routes page.svelte fájlnak felel meg.

A **főoldalon** két funkció érhető el, a Bejelentkezés és a Checkpoint használata, a dolgozók ha munkaidőt szeretnének megkezdeni vagy befejezni akkor a Checkpointot tudják használni, ha viszont a saját oldalukra szeretnének belépni, hogy műveleteket hajtsanak végre akkor a Bejelentkezést kell használni. A harmadik nem működő gomb a Mobil app letöltés, fejlesztési céllal maradt bent, ha később lesz link az apphoz.

A projektben szereplő műveletek megfelelő működésének megteremtése érdekében a szerver oldali komponenssel össze kellett kötni a frontendet a **.env** fájlokban, a backend logikájához hasonlóan itt is van egy fejlesztői és egy termék oldali komponens, ami biztosítja, hogy a végpontok localhost-ról és a render.com-ról is megérkezzenek a frontend felé. A vite.config.js fájlban pedig a környezeti változók szerepelnek, illetve a proxy beállításokat tartalmazza a belső API útvonalak beállítására. A képen a vite.config.js egy részlete látszik a fent ismeretett elemekkel.

```
export default defineConfig(({ mode }) => {
  // Betöltjük a környezeti változókat (pl. .env, .env.production)
  const env = loadEnv(mode, process.cwd(), '');
  return {
    plugins: [sveltekit()],
    server: {
      port: 5050,
      strictPort: true,
      proxy: {
        '/api': {
          target: env.VITE_API_URL,
          changeOrigin: true,
          secure: false,
        },
```

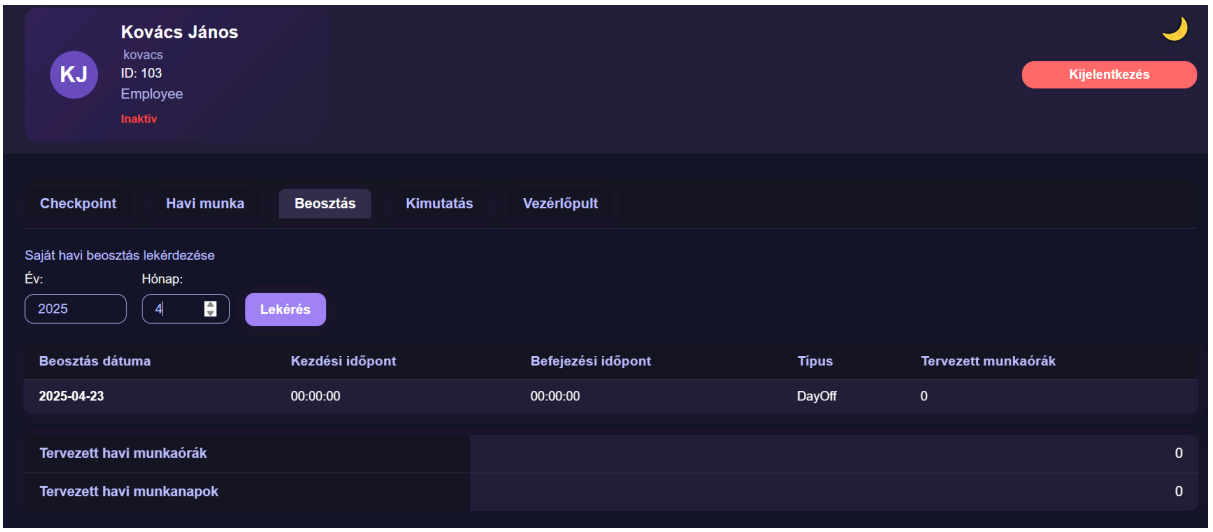
Az Admin, Manager és Employee jogkörök szerint az aloldalak egyforma külső megjelenéssel, de eltérő tartalommal jelennek meg.



A **jogkör ellenőrzése** összetett módon történik. Az authStore.js tárolja a tokent, felhasználói szerepkört és alkalmazotti azonosítót. Az authFetch.js automatikusan kezeli a bejelentkezéskor kapott tokent, tehát a “Bearer” előtegot hozzáteszi. Az átirányítás goto() metódussal történik a megfelelő aloldalra a bejelentkezési logikában, és a jogosultság alapján a megfelelő oldalra irányítja a bejelentkezett dolgozót.

```
  try {
    const response = await fetch('/api/Auth/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ username, password })
    });
    if (!response.ok) {
      const errorMsg = await handleBackendMessage(response);
      throw new Error(errorMsg);
    }
    const data = await response.json();
    if (data.token && data.userRole && data.employeeId) {
      auth.login({ token: data.token, userRole: data.userRole, employeeId: data.employeeId });
      loginOpen = false;
      username = '';
      password = '';
      if (data.userRole === 'Admin') {
        goto('/dashboard/admin', { replaceState: true, reload: true });
      } else if (data.userRole === 'Manager') {
        goto('/dashboard/manager', { replaceState: true, reload: true });
      } else if (data.userRole === 'Employee') {
        goto('/dashboard/employee', { replaceState: true, reload: true });
      } else {
        goto('/', { replaceState: true, reload: true });
      }
    } else {
      throw new Error('Hiányzó adat a válaszból!');
    }
  } catch (error) {
    loginErrorMessage = error.message || 'Ismeretlen hiba';
  } finally {
    loginLoading = false;
  }
}
```

Az egyes oldalakon a backend oldali logika szerint szerepelnek a **metódusok** külön tabon találhatóak az egyes API végpontokhoz, illetve Service réteghez tartozó lekérések. A frontend oldalon gombok, mezők találhatóak és egyszerű leírások az egyes metódusok felett, hogy milyen működéssel rendelkeznek. Az Employee oldalon a következő eredmény látszik egy lekérés esetén:



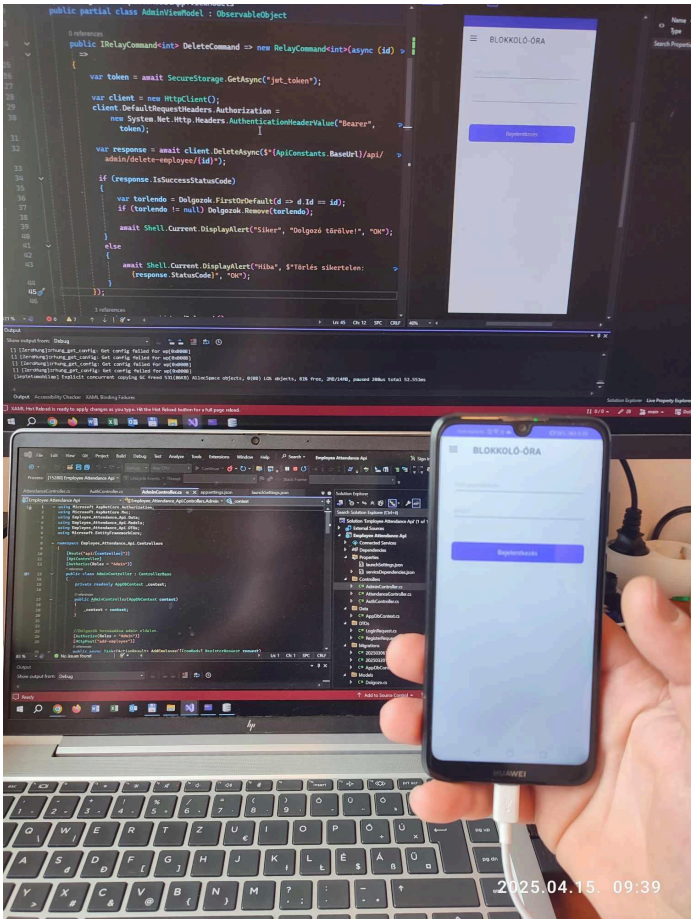

A frontend oldalon a **fetch** mechanizmusával történik a lekérés. A megjelenő táblák és esetleges kiegészítő táblák stílusa is egységes a projekt szerkezetében.

Az **üzenetek és hibaüzenetek kezelése** is egységesített, a messageHandler.js fájl megvizsgálja a választ content-type-ját, ha JSON kinyeri az üzenet mezőt, szöveges válasz esetén megjeleníti az üzenetet és minden más esetben visszaadja az “Ismeretlen hiba” üzenetet.

```
/**
 * Kezeli a háttérhívások válaszait.
 * @param {Response} response - A háttérhívás válasza
 * @returns {Promise<string>} - Felhasználóbarát üzenet
 */
export async function handleBackendMessage(response) {
  const unknownError = 'Ismeretlen hiba';
  const contentType = response.headers.get('content-type') || '';

  try {
    if (contentType.includes('application/json')) {
      const data = await response.json();
      if (typeof data === 'object' && data !== null && data.message) {
        return data.message;
      }
      return JSON.stringify(data);
    } else if (contentType.includes('text/')) {
      const text = await response.text();
      return text || unknownError;
    } else {
      return unknownError;
    }
  } catch (e) {
    return unknownError;
  }
}
```

## 2.6 A mobil komponens bemutatása

A **.NET MAUI** választása lehetővé teszi a multiplatformos mobilalkalmazás-fejlesztést (Android, iOS), a Visual Studio pedig beépített támogatást kínál ehhez. A MYSQL Server könnyen integrálható az ASP.NET API-val, és jól támogatja a relációs adatmodellezést.




Főbb változók a MAUI ViewModel-ben:

- Jogosultság alapján történő felület beállítása Admin, Manager, Employee /IsAdmin
- Állapotváltozó a munkavégzéshez IsWorking/ SessionStatus
- MVVM parancsok a munkavégzéshez- StartWorkCommand, EndWorkCommand

- UI megjelenítéséhez használt ObservablePoverty-k - DailyHours, MonthlyStats

## 2.7 Fejlesztési tervezet 16 hétre 2 fős csoportban

**Munkamegosztás**:

Adamovits Otília: backend, frontend, tesztelés, dokumentáció

Molnár Szabolcs: backend, mobil, tesztelés, dokumentáció

**16 hetes tervezet:**

**1. hét** - Tervezés és előkészületek:
Rendszer követelményeinek pontosítása
Felhasználói szerepek és jogosultságok meghatározása
Alapvető projektszerkezet kialakítása ( alapbeállítások)

**2. hét** - Adatbázis tervezés
Adatbázis struktúra tervezése
MySQL séma létrehozása (5 tábla: checkpoint, employee, monthlyreport, schedules, worklog)
Aiven regisztráció és Dbeaver beállítás
Alap adatok összegyűjtése, előkészítése, model-ek tervezése

**3. hét** - Backend fejlesztés kezdete
C# backend projekt beállítása
Alap entitások, modellek és DTO-k létrehozása
Automapper használata
Adatbázis kapcsolat kialakítása
Service réteg elkezdése

**4. hét** - Backend API fejlesztés

Service réteg folytatása

Helperek létrehozása

Hitelesítés és jogosultságkezelés beállítása

**5. hét** - Backend API befejezés

Controllerek létrehozása

Controllerek metódusainak kialakítása

Adatbázis tesztadatokkal való feltöltése

Swagger tesztelés

**6. hét** - Frontend fejlesztés kezdete

Vite + SvelteKit projekt beállítása

Alap oldalstruktúra létrehozása (főoldal és aloldak)

Szerver oldali kommunikációs komponensek elkészítése

**7. hét** - Frontend stíluselemek kialakítása

Sötét és világos téma kialakítása

Global.css, +layout.svelte létrehozása

A főoldal és az aloldalak stílusának kialakítása

**8. hét** - Frontend fejlesztés

A hitelesítés alapú átirányítás beállítása

Hibakezelések és validációk a frontendben

Tab-ok stílusának kialakítása

**9. hét** – Frontend véglegesítése

API végpontok fetchelése

Stílus és funkcióelemek tisztázása

Tesztelés

**10. hét** - Mobil fejlesztés
.NET MAUI projekt létrehozása
Bejelentkezési képernyő és hitelesítés fejlesztése
Backend API csatlakoztatása a mobil apphoz

**11. hét** - Mobil alkalmazás folytatása
API végpontok tisztázása
Kijelentkezési lehetőség
Stíluselemek tisztázása

**12. hét** - Tesztelés, hibakezelés
Backend, frontend és mobil alkalmazás tesztelése
Fejlesztői és termék környezet kialakítása
render.com és netlify deploy

**13. hét** - Hibajavítások és optimalizálás
Rendszer teljes átnézése és hibajavítások
Tesztelési eredmények alapján módosítások
API végpontok finomhangolása

**14. hét** - Dokumentáció elkészítése
Backend és frontend dokumentáció írása
Mobilalkalmazás dokumentációja
Teszt dokumentálás

**15. hét** – Prezentáció előkészítése
PPT elkészítése
Próba bemutatás, gyakorlás

**16. hét** – Végső ellenőrzések
Konzultáció a témavezetővel
Végső simítások, ellenőrzések

# 3. Fejlesztési lehetőségek

A jelenlegi rendszer stabil alapot biztosít, azonban a jövőben további funkciókkal szeretnénk bővíteni:

1. **Automatizált havi riportok**: PDF export funkció, amely havi jelentéseket generál a dolgozók ledolgozott munkaidejéről.

2. **E-mail értesítések**: beosztás létrehozásáról automatikus e-mail küldése a dolgozóknak.

3. **Többnyelvűség támogatása**: angol-magyar felület közötti váltás a frontend oldalon.

4. **RFID vagy QR kód alapú beléptetés:** gyors munkába való megérkezés és még inkább modernizált működés.

# 4. Kiegészítő felhasználói dokumentáció

## 4.1 A program céljának és lényegesebb funkcióinak összefoglalása

A szoftver egy **webalapú dolgozói beléptető és munkaidő-nyilvántartó rendszer, beosztás készítési lehetőséggel**. Célja, hogy digitálisan nyilvántartsa a vállalkozás munkavállalóinak munkaidejét: mikor léptek be és ki a munkahelyről, illetve mennyi időt töltöttek el munkával egy adott napon vagy hónapban és beosztást lehessen létrehozni, ami alapján tervezhető a munkaidő, és összehasonlítható a belépésekkel.

A rendszer **három fő szereplőt** különböztet meg:

- **Dolgozó**: beléphet, kiléphet és megtekintheti havi ledolgozott óráit, beosztását, kimutatását.
- **Manager:** hozzáfér a manager felülethez, ahol tud beosztást készíteni és megtekintheti a dolgozók kimutatásait.
- **Adminisztrátor**: hozzáfér az admin felülethez, ahol dolgozókat vehet fel, törölhet, és után láthatja az összes dolgozó minden kimutatását.

A program a böngészőben használható, nem szükséges telepíteni semmilyen további alkalmazást a számítógépre. Localhost-on történő használat esetén érdemes Visual Studio és Visual Studio Code alkalmazásokból indítani a projektet.

Elérési pontok:
https://munkarendszer.netlify.app/

https://worksystem.onrender.com/swagger/index.html

https://github.com/adaotilia/projektmunka_2025

## 4.3 Telepítés és indítás lépéseinek ismertetése

**Backend telepítése:**

1. Töltse le vagy klónozza a projektet GitHub-ról.
2. Nyissa meg a projektet Visual Studio 2022-ben.
3. Ellenőrizze az appsettings.json fájlt, hogy az adatbázis elérési út helyes legyen (pl. MYSQLite helyi fájl vagy MYSQL Server).
4. Nyomjon F5-öt, a zöld nyilat vagy indítsa el dotnet run paranccsal.
5. A rendszer alapértelmezés szerint a http://localhost:8080 címen fog futni.

**Frontend telepítése:**

1. Navigáljunk a frontend mappába parancssorban vagy terminálban.
2. Futtassuk: npm install
3. Indítsuk el a fejlesztői szervert: npm run dev
4. Az alkalmazás a http://localhost:5050 címen lesz elérhető böngészőben. (Lehetséges, hogy adapter telepítése szükséges a megfelelő működéshez.)

**Felhasználói oldal megnyitása:**

- A dolgozó vagy adminisztrátor egyszerűen böngészőből megnyitja a frontend URL-t: http://localhost:5050
- Belépéshez szükséges adatok:
    - admin1 teszt123
    - manager1 cica1234
    - kovacs teszt123

**Swagger elérés:**

- A backend API a https://localhost:8080/swagger címen dokumentálva van, ahol tesztelhetők a végpontok.

**Telepítés és indítás lépéseinek ismertetése MAUI APP**

1. Töltsük le az APK fájlt a fejlesztőtől (pl. e-mail, fájlmegosztó link).
2. A telefon beállításaiban engedélyezzük az **ismeretlen forrásból származó alkalmazások** telepítését.
3. Keressük meg a letöltött fájlt, és kattintsunk rá a telepítéshez.
4. A telepítés után megjelenik az alkalmazás ikonja a telefon főmenüjében.
5. Kattintsunk az ikonra az alkalmazás elindításához.
6. Jelenleg teszt környezetben működik az app verzió tehát közös Wifi hálózaton kell legyen a backend és a tesztelésre használt mobil készülék, ez azonban (ha ez a kívánság) előnyt is jelenthet mivel például csak a munkahelyen tudnak a dolgozók becsekkolni.

## 4.2 A program bemutatása

### Munkarendszer:

**Főoldal:**

- Bejelentkezés: Felhasználónév és jelszó megadásával működik
- Checkpoint: Az EmployeeId megadása után Belépés vagy Kilépés gomb megnyomása.

**Oldalak: Admin, Manager, Employee**

- Header: Az oldal felső részében a dolgozói adatokat lehet látni, Kijelentkezés gombot, és a témaváltó gombot a jobb felső sarokban.
- 5 tab: Checkpoint, Havi munka, Beosztás, Kimutatás, Vezérlőpult
- A tabok alatt eltérő műveletek találhatóak, a műveletek felett rövid leírás olvasható, hogy mire jó az adott művelet. A mezőkben példa szerepel arra,

hogy mit és milyen formátumban kell megadni. A frontend által visszaadott üzenetek és hibaüzenetek támogatják a helyes használatot.

**MAUI APP mobil alkalmazás bemutatása**

- Bejelentkezés: Az indító képernyőn be kell írni a felhasználónevet és jelszót. Sikeres bejelentkezés esetén a megfelelő aloldalra jutnak a dolgozók.
- Kijelentkezés: A hamburger menüben található „Kijelentkezés” gombbal visszatérhetünk a bejelentkező képernyőre.
- Bejelentkezés után az aloldalakon megjelenítési lehetőségek találhatóak.

# 5. SWOT-analízis a munkarendszerhez

**Erősségek (Strengths)**

- Biztonságos bejelentkezés JWT token alapú hitelesítéssel és Bcrypt jelszó hash-eléssel.
- Külön admin, manager és dolgozó szerepkörök kezelése.
- Automatikus munkaóra nyilvántartás (belépés és kilépés rögzítése).
- API alapú backend, amely könnyen integrálható a Svelte frontenddel.
- Swagger integráció, amely megkönnyíti az API tesztelését és dokumentálását.
- Platformfüggetlen futtathatóság (ASP.NET Core backend, MYSQL Server adatbázis).

**Gyengeségek (Weaknesses)**

- A rendszer skálázhatósága korlátozott lehet nagyobb céges környezetekben (pl. több száz dolgozó esetén).

**Lehetőségek (Opportunities)**

- Integráció RFID/NFC olvasókkal a gyorsabb beléptetés érdekében. Könnyen adaptálható más, hasonló célú rendszerekhez is, és amelyet tovább lehet fejleszteni – például RFID vagy NFC-alapú beléptetés integrálásával, vagy felhőalapú adattárolással.

**Veszélyek (Threats)**

- Biztonsági kockázatok: amennyiben nem megfelelően frissíted a függőségeket és a rendszer sérülékenységeit (pl. JWT token kezelése).
- Adatvédelmi kihívások (GDPR megfelelés), különösen ha személyes adatokat kezelsz (pl. név, munkaidő).
- Nagyvállalati környezetben versenyhelyzet más, kész, kereskedelmi rendszerekkel szemben (pl. SAP vagy munkaidő-nyilvántartó szoftverek).
- Technikai elavulás, ha a rendszer hosszabb távon nem kap folyamatos karbantartást vagy fejlesztést.

# 6. Összefoglalás, köszönetnyilvánítás

A munkarendszer projekt fejlesztése során sikerült elérnünk azokat a szakmai célokat, amelyeket a vizsgaremek megalkotása előtt megfogalmaztunk. A rendszer teljeskörűen lefedi az alapvető munkaidő nyilvántartási, kezelési folyamatokat, a beosztás létrehozása és kimutatások lekérdezése funkciókat. Biztosítja a jogkörök szerinti funkciók használatát. Fontos célunk volt, hogy a rendszer egy könnyen használható, webes felületen keresztül legyen elérhető, ami az adminisztrátorok, managerek és a dolgozók számára egyszerű és átlátható élményt nyújt.

A projekt során rengeteg hasznos tapasztalattal gazdagodtunk. Kiemelnénk az ASP.NET Core Web API használatának mélyebb megismerését, különös tekintettel a JWT token-alapú autentikációra és a REST API biztonsági megoldásaira. Szintén sokat tanultunk a SvelteKit frontend keretrendszer alkalmazásából, amely segített abban, hogy modern és gyors webes felületet tudjunk kialakítani.

A legnagyobb kihívást talán az jelentette, hogy az API és a frontend oldal közötti kommunikációt biztonságosan és hatékonyan alakítsuk ki, figyelve a CORS problémákra, a tokenek kezelésére és az adminisztrátori jogosultságok megfelelő elkülönítésére.

A projekt utóéletét illetően a célunk az, hogy a rendszert tovább bővítsük új funkciókkal. Szeretnénk bevezetni a PDF formátumú havi riportok exportálását, és integrálni egy e-mail értesítési rendszert is, továbbá RFID alapú munka bejelentkezési lehetőséget biztosítani.

Zárásképpen szeretnénk megköszönni mindazoknak a támogatását, akik közvetve vagy közvetlenül hozzájárultak a projekt sikeres megvalósításához. Külön köszönettel tartozunk témavezetőnknek, Rédai Dávidnak, oktatóinknak a szakmai iránymutatásért, valamint családjainknak és barátainknak a türelemért és támogatásért, amelyet a fejlesztési időszak során nyújtottak számunkra.

## 7. Irodalomjegyzék

https://learn.microsoft.com/hu-hu/dotnet/csharp/

https://www.c-sharpcorner.com/article/cors-in-dotnet-core/

https://dev.mysql.com/doc/refman/8.4/en/

https://dbeaver.com/docs/dbeaver/

https://svelte.dev/docs/kit/introduction

https://javascript.info/

https://render.com/docs

https://docs.netlify.com/

https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-9.0

https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-9.0&WT.mc_id=dotnet-35129-website

https://chatgpt.com/