

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR,  
INFORMATIKA SZAK**



**SAPIENTIA  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM**

MuresTracker - Városi problémák követési rendszere

**DIPLOMADOLGOZAT**

Témavezető:

Dr. Jánosi-Rancz Katalin Tünde,  
Egyetemi adjunktus

Végzős hallgató:

Sikó Márk-Levente

**2023**

**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,  
SPECIALIZAREA INFORMATICĂ**



# UNIVERSITATEA SAPIENTIA

## LUCRARE DE DIPLOMĂ

Coordonator științific: Absolvent:  
Dr. Jánosi-Rancz Katalin Tünde, Sikó Márk-Levente  
Lector universitar

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF  
TRANSYLVANIA**  
**FACULTY OF TECHNICAL AND HUMAN SCIENCES**  
**COMPUTER SCIENCE SPECIALIZATION**



**SAPIENTIA**  
HUNGARIAN UNIVERSITY  
OF TRANSYLVANIA

MuresTracker - City issue tracker

**BACHELOR THESIS**

Scientific advisor: Dr. Jánosi-Rancz Katalin Tünde,  
Lecturer Student: Sikó Márk-Levente  
2023

### Declarație

Subsemnatul/a .....S.IKÓ MÁRK -LEVANTE....., absolvent(ă) al/a specializării .....INFORMATICA....., promoția.....2023..... cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, TÂRGU MUREŞ  
Data: 15.06.2023

Absolvent

Semnătura.....S.ILÉJ.....

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș  
Programul de studii: Informatică

Viza facultății:

## LUCRARE DE DIPLOMĂ

Coordonator științific:  
**Dr. Jánosi-Rancz Katalin Tünde**

Candidat: Sikó Márk-Levente  
Anul absolvirii: 2023

**a) Tema lucrării de licență:**

Proiectarea și dezvoltarea unei aplicații software pentru urmărirea problemelor orașului - MuresTracker în Java

**b) Problemele principale tratate:**

În lucrare este prezentată o aplicație numită MuresTracker, scrisă în cadrul framework-ului Java Spring&Angular și folosește o bază de date SQL. MuresTracker oferă o platformă pentru călătorii să raporteze problemele și să monitorizeze situația, iar oficialii orașului să se confrunte și să rezolve problemele.

**c) Desene obligatorii:**

Grafice despre structura bazei de date  
Grafice despre structura codului  
Grafica despre Use-Case  
Grafica despre design-ul aplicației  
Diagramă de secvență

**d) Softuri obligatorii:**

Aplicația este bazată pe tehnologii Java care realizează comunicarea între cetăteni și conducerea orașului cu privire la problemele actuale din oraș. Cetătenii pot adăuga problemele cu care se confruntă în oraș. Aplicația va afișa toate problemele, cu detalii și statusurile lor. Pentru afișarea rapoartelor, frontend-ul a fost realizat cu Angular și NgPrime. Software-ul include o bază de date SQL pentru salvarea rapoartelor.

**e) Bibliografia recomandată:**

- Arnold, Ken, James Gosling and David Holmes (2005). The Java Programming Language  
Spring MVC Beginner's Guide - Second Edition by Amuthan Ganeshan  
-AngularJS Web Development Cookbook, Matt Frisbie

**f) Termene obligatorii de consultații:** Studentul a început dezvoltarea aplicației în septembrie 2022 și ne am întâlnit la fiecare 2-3 săptămâni

**g) Locul și durata practicii:** Universitatea „Sapientia” din Cluj-Napoca,  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, sala / laboratorul 327A  
Primit tema la data de: mai 2022  
Termen de predare: 2 iulie 2023

Semnătura Director Departament

Semnătura responsabilului  
programului de studiu

Semnătura coordonatorului

Semnătura candidatului

# Kivonat

Dolgozatom témája a *Mures Tracker*, egy olyan szoftver, amely lehetővé teszi a városlakók számára helyi problémák bejelentését és a városvezetés számára ezek hatékony kezelését.

A városvezetésnek számos kihívással kell szembenéznie nap mint nap, amelyek hatással vannak a lakosok életminőségére. A kátyúk, a meghibásodott lámpaoszlopok, a bűnözés, a biztonsági aggodalmak és hasonlók minden nap jelen vannak a lakosok minden nap életükben, nem tetszésüket pedig nem minden nap megfelelőbb helyeken nyilvánítják ki, ezért a városoknak hatékony eszközökkel van szükségük a problémák kezeléséhez.

A *Mures Tracker* ezt a problémát oly módon kezeli, hogy egy platformot biztosít a lakosok számára a problémák jelentése és a helyzet nyomon követése érdekében, valamint a városi tiszttiselők számára a problémákkal való szembesülésre és ezek megoldására.

A weboldal értékes eszköz a kommunikáció javításához, a polgárok részvételének növeléséhez, a problémák nyomon követéséhez és azok kezeléséhez, valamint a hatékonyabb városi szolgáltatások előmozdításához. Széleskörű, felhasználóbarát jelentéstételi, nyomon követési lehetőségeket biztosít és pozitív hatást gyakorolhat a lakosok életére és a városok hatékonyságára.

Az alkalmazás lehetővé teszi a bejelentkezett egyszerű felhasználóknak, hogy hozzáadjanak és megtekintsék a mások által bejelentett problémákat a városba, azoknak részleteit, helyzetét, helyszínét, valamint hogy ezeket ki és mikor oldotta meg. Az adminisztrátor típusú felhasználók felelnek, hogy egy bizonyos kategóriájú probléma ahhoz a szervhez kerüljön akinek az a hatásköre. Ezenfelül rálátásuk van a felhasználókra, módosítani tudják őket és inaktívvá tenni, példátlan viselkedés következtében.

**Kulcsszavak:** város, problémák, webalkalmazás, spring boot.

# Rezumat

Tema lucrării mele este *Mures Tracker*, un software care permite locuitorilor orașului să raporteze probleme și oferă o gestionare eficientă a acestora pentru autoritățile locale. Administrația orașului se confruntă zilnic cu numeroase provocări care afectează calitatea vieții locuitorilor. Gropi, stâlpi de iluminat defectate, criminalitatea, îngrijorări legate de siguranță și altele sunt prezente în viața de zi cu zi a locuitorilor, iar nemulțumirile lor nu sunt întotdeauna exprimate în locuri potrivite, de aceea orașelor le sunt necesare instrumente eficiente pentru urmărirea și rezolvarea problemelor.

*Mures Tracker* abordează această problemă prin furnizarea unei platforme prin care locuitorii pot raporta probleme și pot urmări starea acestora, iar oficialităților orașului li se oferă instrumente pentru a se confrunta și rezolva aceste probleme. Acest site web reprezintă o unealtă valoroasă pentru îmbunătățirea comunicării, creșterea participării cetățenilor, urmărirea și gestionarea problemelor, precum și promovarea serviciilor urbane mai eficiente. Oferă o gamă largă de opțiuni de raportare și urmărire prietenosă pentru utilizatori și poate avea un impact pozitiv asupra vieții locuitorilor și eficienței orașelor.

Aplicația permite utilizatorilor simpli înregistrați să adauge și să vizualizeze problemele raportate de alții în oraș, detalii despre acestea, locația și cine, când le-a rezolvat. Utilizatorii cu rol de administrator sunt responsabili de direcționarea problemelor dintr-o anumită categorie către departamentul potrivit. În plus, ei au acces la informații despre utilizatori și pot efectua modificări sau dezactivări ale acestora în caz de comportament neadecvat.

**Cuvinte cheie:** oraș, probleme, aplicație web, Spring Boot.

# Abstract

The topic of my paper is *MuresTracker*, a software that enables city residents to report issues and facilitates efficient management of these problems by city authorities. City administrations face numerous challenges on a daily basis that impact the quality of life for residents. Potholes, malfunctioning streetlights, crime, safety concerns, and similar issues are present in the daily lives of residents, and their dissatisfaction is not always expressed in the most appropriate places. Therefore, cities require effective tools for tracking and addressing problems.

*MuresTracker* addresses this problem by providing a platform for residents to report issues and track their status, while also enabling city officials to confront and resolve these problems. The website serves as a valuable tool for improving communication, increasing civic engagement, monitoring and managing problems, and promoting more efficient urban services. It offers a wide range of user-friendly reporting and tracking capabilities and can have a positive impact on residents' lives and the efficiency of cities.

The application allows registered regular users to add and view problems reported by others in the city, including details, location, and who and when they were resolved. Users with administrator roles are responsible for directing problems of a certain category to the appropriate department. Additionally, they have insight into user information and can modify or deactivate users in case of inappropriate behavior.

**Keywords:** city, issues, web application, spring boot.

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>10</b>
<b>2. Célkitűzések</b>	<b>12</b>
<b>3. Felhasznált technológiák</b>	<b>13</b>
3.1. Szervert oldalon használt technológiák . . . . .	13
3.1.1. Keretrendszer és könyvtár összehasonlítása . . . . .	13
3.1.2. Inversion of Control . . . . .	14
3.1.3. Dependency Injection . . . . .	14
3.1.4. Bean . . . . .	16
3.1.5. API . . . . .	16
3.1.6. REST API . . . . .	16
3.1.7. Spring és Spring boot összehasonlítása . . . . .	17
3.1.8. Project Lombok . . . . .	19
3.1.9. Entity osztályok . . . . .	20
3.1.9.1. DTO osztályok és konverterek . . . . .	21
3.1.10. Spring Boot alkalmazás felépítése . . . . .	21
3.1.11. JPA Repository . . . . .	23
3.1.12. Kivételkezelés . . . . .	24
3.1.13. Spring security . . . . .	25
3.2. Kliens oldalon használt technológiák . . . . .	26
3.2.1. Angular alkalmazás felépítése . . . . .	26
3.2.2. TypeScript . . . . .	26
3.2.3. Angular Routing . . . . .	26
3.2.4. Guardok . . . . .	28
3.2.5. PrimeNG . . . . .	28
3.2.6. Nemzetköziesítés és az i18n . . . . .	29
<b>4. A rendszer specifikációja</b>	<b>30</b>
4.1. Felhasználói követelmények . . . . .	30
4.1.1. Fontosabb használati esetek . . . . .	32
4.2. Rendszerkövetelmények . . . . .	35
4.2.1. Funkcionális követelmények . . . . .	35
4.2.2. Nem funkcionális követelmények . . . . .	36
<b>5. Tervezés</b>	<b>39</b>
5.1. A rendszer architektúrája . . . . .	39

5.2. Adatbázis . . . . .	40
5.3. Felhasználói interfész tervezése . . . . .	40
<b>6. Kivitelezés</b>	<b>41</b>
6.1. Kliens oldali megvalósítás . . . . .	41
6.1.1. Szerver oldallal való kommunikáció . . . . .	41
6.1.2. Service hívások eredményének feldolgozása . . . . .	42
6.1.3. Angular routing . . . . .	43
6.2. Szerver oldali megvalósítás . . . . .	44
6.3. Az alkalmazás működése . . . . .	44
6.3.1. Bejelentkezés . . . . .	44
6.3.2. Regisztráció . . . . .	44
6.3.3. Kezdőlap . . . . .	44
6.3.4. Értesítések . . . . .	44
6.3.5. Felhasználók lista, szerkesztése és hozzáadása . . . . .	47
6.3.6. Bejelentések lista, szerkesztése és hozzáadása . . . . .	47
6.3.7. Szerepkörök . . . . .	47
6.3.8. Statisztikák . . . . .	47
6.4. Segédeszközök . . . . .	47
<b>Összefoglaló</b>	<b>50</b>
<b>Ábrák jegyzéke</b>	<b>51</b>
<b>Irodalomjegyzék</b>	<b>53</b>

# 1. fejezet

## Bevezető

A fejezet célja a dolgozat témájának és motivációjának ismertetése, annak való életbeli használati eseteinek bemutatása.

A folyamatosan növekvő városi népesség, a korlátozott erőforrások és a hatóságok kommunikációs nehézségei miatt gyakran nehéz a problémák azonosítása és azok hatékony kezelése, ezért a lakosok nagyon hamar elégedetlenek lesznek a városvezetéssel. Az emberek nem tudják, hogy hogyan és hol jelentsenek bizonyos problémákat, hogy megfelelően eljussanak azok az illetékesekhez. Ez hamar eredményezhet elégedetlenséget, bizalmatlanságot és csökkent életminőséget.

Életünknek egyik legfontosabb szakasza az, amikor eljön a pillanat, hogy döntést kell hoznunk arról, hol szeretnénk letelepedni és családot alapítani. Ez egy döntés, amely hosszú távú hatással lesz minden napra életünkre és jól-létünkre. Az egyik legfontosabb szempont, amit figyelembe kell vennünk, az az, hogy egy tiszta és élhető városba költözünk, ahol kevés probléma és kihívás nehezíti minden napjainkat.

A MuresTracker egy hatékony megoldást kínál a városi problémák kezelésére és nyomon követésére. Az alkalmazás lehetővé teszi a felhasználók számára, hogy egyszerűen és gyorsan bejelentsék a problémákat a város illetékesinek. A bejelentések tartalmazhatnak fényképeket, leírást és helymeghatározást, amelyek segítik az azonosítást és a problémák prioritás szerinti kezelését. A városi hatóságok pedig nyomon követhetik a beérkező jelentéseket, és hatékonyan kezelhetik azokat.

Példák és esettanulmányok: Hasonló megoldásokat már alkalmaztak világszerte, amelyek javították a városi szolgáltatások minőségét és növelték a polgárok részvételét. Például Boston városa bevezette a 311 szolgáltatást, amely lehetővé teszi a polgárok számára a problémák jelentését és a városi szolgáltatások igénybevételét, ami gyorsabb reakcióidőt és hatékonyabb városi szolgáltatásokat eredményezett.

Marosvásárhelyen is létezik egy ehhez hasonló mobilos alkalmazás, de nem olyan könnyű használni mint az én weboldalamat. Emelett nem lehet követni olyan tisztán a bejelentések státuszát és kategóriáját. Nem beszélve a borzasztó felhasználói felületről, ami ellehetetleníti az alkalmazás gond nélküli használatát. Zavaros elrendezéssel, hiányzó és nehezen elérhető funkciókkal és hiányos visszajelzéssel találkoztam használat során. A könnyű használat mellett az én weboldalam a bejelentett problémáról és azoknak megoldásáról részletes statisztikát kínál. A dolgozat további részében ismertetem a weboldal elkészítése előtt kitűzött célokot. Ezt követi a felhasznált technológiák ismertetése úgy felhasználó, mint szerver oldalon, továbbá fogalmak és függőségek elméleti meghatározása.

A következő fejezet tartalmazza a szoftver funkcionális és nem-funkcionális követelményeit és használati esetek bemutatását. Utána a rendszer architektúrájáról lesz szó, valamint az adatbázis és felhasználói felület terveiről.

Végül egy összefoglalóval zárom, amelyben következtetéseket és továbbfejlesztési lehetőségeket fejtek ki.

## 2. fejezet

# Célkitűzések

Célok kitűzése minden napjaink állandó szereplője, legyen szó személyes vagy üzleti életünkről. A célok meghatározása segít a nagy, lehetetlennek tűnő ötletekből kézzelfogható és mérhető célokat formálni. A fejezet tartalmazza a *Mures Tracker* megvalósításával kapcsolatosan kitűzött célokat a felépítésével kapcsolatosan és a főbb funkcionálisaival kapcsolatosan.

A szoftver elkészítéséhez kitűzött célok a következők:

- mysql adatbázis megtervezése az adatok tárolása érdekében,
- entitások definiálása és létrehozása,
- felhasználó kategóriák definiálása,
- projektstruktúra előkészítése, projekttervezési technikák meghatározása,
- GUI megtervezése<sup>1</sup> és összekötése a szerverrel,
- kliens-szerver architektúra megalkotása egy REST<sup>2</sup> API segítségével.

Fő funkcionálisok:

- regisztráció,
- bejelentkezés,
- problémák bejelentése, listázása és azokkal kapcsolatos műveletek végrehajtása,
- felhasználók listázása és azokkal kapcsolatos műveletek végrehajtása,
- szerepkörök és jogok figyelembevételével történő betöltés,
- értesítés különböző folyamatokról,
- keresési lehetőség és filterezés,
- többnyelvűség,
- statisztika.

---

<sup>1</sup>Graphical user interface

<sup>2</sup>Representational State Transfer

## 3. fejezet

# Felhasznált technológiák

### 3.1. Szerver oldalon használt technológiák

#### 3.1.1. Keretrendszer és könyvtár összehasonlítása

A modern programozás világában a keretrendszerök és könyvtárak kulcsfontosságú eszközök, amelyek segítenek a fejlesztőknek strukturáltabb módon dolgozni és segítenek előidézni a kód-újrafelhasználást ami által gyorsabb és hatékonyabb kódolást érhetünk el. A programozók gyakran egy kalap alá veszik a két fogalmat, de különböznek egymástól a funkcionalitásuk és a felhasználásuk módja szempontjából.

*A könyvtár:* olyan gyűjtemények, amelyek egy bizonyos funkcionalitásokra fókusztálnak. Előre írt kódrészleteket, modulokat és osztályokat tartalmaznak, amelyek könnyen felhasználhatók a programozási projektekben. A fejlesztő hozzátudja adni őket és ott használja az adott könyvtár függvényeit, amikor csak akarja. Ez azt jelenti, hogy a könyvtárak általában nem határozzák meg a fejlesztési mintákat vagy a projekt struktúráját.

*A keretrendszer:*<sup>[1]</sup> lényege, hogy a különböző alkalmazásokban leggyakrabban használt elemeket egyetlen helyre gyűjtik össze, és készen kínálják a fejlesztők számára, amelyek így rengeteg elvégzendő munkától mentesülnek. Bizonyos tipikus feladatok elvégzését nagy mértékben segítő, egységes módon megszerkesztett komponenseket tartalmazó halmaz és a fejlesztő ezekre az adott "építőkockákra" építi fel a saját implementációját betartva a keretrendszer megkötéseit. Tartalmaz előre definiált sablonokat, szabályokat és struktúrákat, amelyek felgyorsítják a fejlesztési folyamatot és tartalmaz olyan alapvető modulokat és könyvtárakat, amelyek megkönnyítik például adatbázis-kezelést vagy felhasználói felület készítését. Érdemes belemerülni és elsajátítani különböző frameworkokat mivel jelentősen növelik a produktivitást. Alkalmazásomba két fő keretrendszer használtam: *Springet* és *PrimeFonest*, melynek a hangsúlya a webes megjelenésen alapszik. A Spring sok komponenst integrál magába, amelyeket a következő alpontokba fogok bemutatni.

Összegezve, mind a keretrendszerök és a könyvtárak hasznosak a fejlesztés során és segíthetnek időt megtakarítani. Viszont a framework átveszi az irányítást a program felett, a programozónak követnie kell a keretrendszer megkötéseit, ellenben a könyvtár nincsenek megkötései és a fejlesztő dönti el mikor van szüksége az adott könyvtár függvényeire, és hogy mikor hívja meg azt <sup>[2]</sup>.

### 3.1.2. Inversion of Control

A kontroll megfordítása egy olyan technika, ami egy keretrendszer keretén belül kerül megvalósításra és lényegében az objektumok irányítását ruházza át, az én programom esetébe a Spring frameworkra. A hagyományos szoftverfejlesztési modellben az alkalmazás irányítja a komponensek létrehozását és összekapcsolását. Az *IoC* azonban ezt az irányítást a keretrendszerre bízza, amely felelős a komponensek életciklusának kezeléséért és a kapcsolatok létrehozásáért.

Ennek az architektúrának az előnyei a következők:

- a komponensek közötti szoros összekapcsolódást csökkenti
- modulárisabbá válik és ezáltal könnyen újrafelhasználhatók
- könnyebb és hatékonyabb tesztelés
- skálázhatóság, karbantarthatóság és rugalmasság
- Keretrendszer-kompatibilitás: a *Spring* vagy az *Angular* könnyedén használható

Az *IoC* megvalósítása többféle módon történhet:

- *Függőségi befecskendezés*<sup>1</sup>, ahol a függőségeket a komponens konstruktőrén keresztül kapja meg
- *IoC konténer*<sup>2</sup>, a konténer automatikusan létrehozza és összekapcsolja a komponenseket, és elérhetővé teszi őket az alkalmazás számára.
- Annotációk használata, például a `@Component` és `@Autowired`

### 3.1.3. Dependency Injection

A függőség befecskendezés az *IoC* [3] egyik esete, amely lehetővé teszi a függőségek kezelését osztályok között. A függőség befecskendezés során egy osztály nem maga hozza létre a szükséges függőségeit, hanem azokat kívülről kapja meg, amelyeket használni szeretne. Technikailag ez úgy történik, hogy attribútumként deklarálja azt, hogy milyen típusú objektumra van szüksége, de azt nem állítja be és egy külső keretrendszer biztosítja azt, hogy mire az első hívás bekövetkezik, az rendelkezésre álljon.

A Java nyelvben az osztályfüggőség azt jelenti, hogy egy osztály használja vagy hivatkozik egy másik osztályra. Az osztályfüggőségek kialakítása és kezelése alapvető fontosságú a hatékony és jól strukturált Java alkalmazások létrehozásához.

Ezekre az osztályfüggőségekre a 3.1 és a 3.2 ábrákon mutatók példát.

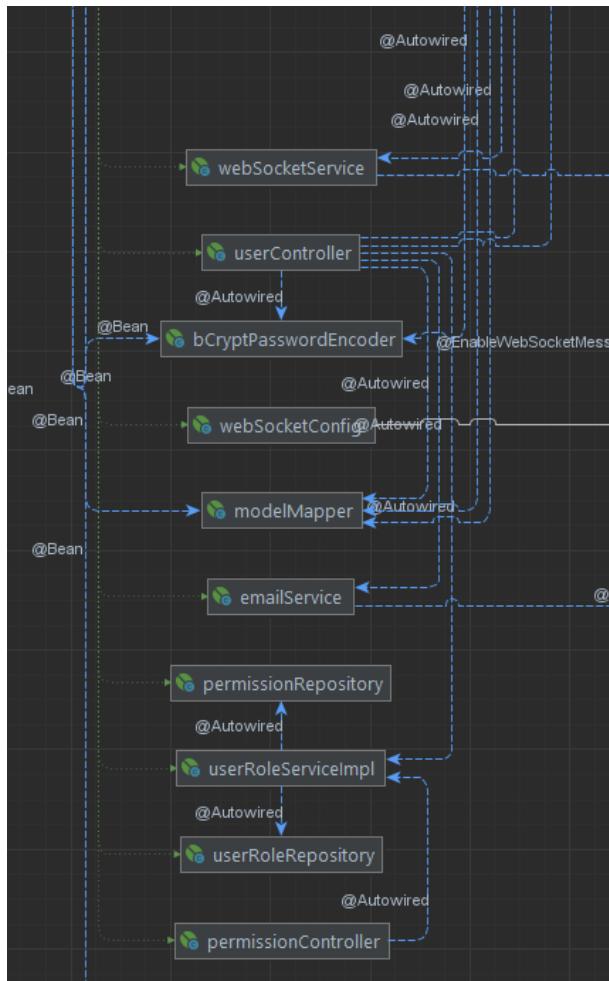
A Reflection lehetőséget ad az osztályok, metódusok, mezők és annotációk tulajdon-ságainak lekérdezésére, hozzáférésére és módosítására anélkül, hogy ismernénk a forráskódot.

A Spring boot esetében három típusú függőségbefecskendezés ismert:

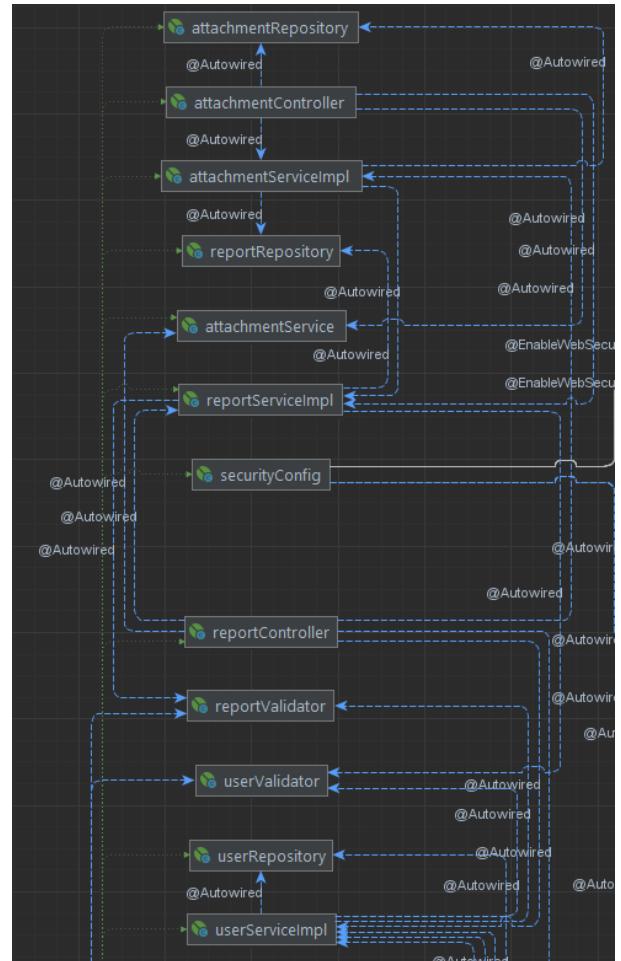
---

<sup>1</sup>Dependency Injection (DI)

<sup>2</sup>Inversion of Control Container



**3.1. ábra.** Felhasználó vezérlő függőségei



**3.2. ábra.** Bejelentések vezérlő függőségei

1. *Konstruktur alapú:* leggyakoribb, a függőségeket a komponens konstruktorán keresztül adjuk át. A Spring az alkalmazás kontextusába lévő komponenseket automatikusan összekapcsolja és a megfelelő konstruktorokon keresztül injektálja a függőségeket. példa a 3.1 kódrészletben megfigyelhető.

---

```

private final NotificationRepository notificationRepository;
private final UserRepository userRepository;
private final WebSocketService webSocketService;

@Autowired
public NotificationServiceImpl(
    NotificationRepository notificationRepository,
    UserRepository userRepository,
    WebSocketService webSocketService) {
    this.notificationRepository = notificationRepository;
    this.userRepository = userRepository;
    this.webSocketService = webSocketService;}

```

---

**3.1. kódrészlet.** A konstruktur alapú függőségbefecskendezés

2. *Setter alapú*: A Spring a setter metódusokat használva automatikusan injektálja a függőségeket a megfelelő komponensekbe.
3. *Mező alapú*: a függőségeket közvetlenül a komponens mezőin keresztül injektáljuk. A Spring az `@Autowired` annotációval jelölt mezőket automatikusan injektálja.

Fontos megjegyezni, hogy minden típus esetén a Spring Boot az IoC konténerében található bean-eket kezeli és injektálja a megfelelő helyekre. Az alkalmazás kontextusának konfigurálása és a megfelelő annotációk (`@Service`, `@Autowired`) használata segít a Spring Bootnak az automatikus függőségi befelecskendezés végrehajtásában.

### 3.1.4. Bean

A *Bean* [4] fogalma a Spring keretrendszerben használt kifejezés, amely egy komponenst vagy egy objektumot jelent, amelyet a Spring IoC konténer kezel.

A *Bean* egy Java objektum, amelyet a Spring konténer létrehoz, konfigurál és kezel. A Spring IoC konténerben definiáljuk és regisztráljuk a bean-eket, és a konténer felelős a bean-ek példányosításáért, konfigurálásáért és függőségi befelecskendezéséért. A bean-eknek van hatókörük (scope), ami meghatározza, hogy a keretrendszer mi módon példányosítsa. Ezt a `@Bean` annotáció után a `@Scope` annotációval tudjuk megadni. Lehetőséges értékek:

- Singleton, alapértelmezett
- Prototype, minden esetben új példányt hoz létre
- Request, HTTP kérésenként jön létre
- Session, munkamenetenként hoz létre új példányt
- Application, alkalmazásonként

### 3.1.5. API

Az API<sup>3</sup> egy olyan szoftveres felület, amely lehetővé teszi az alkalmazások közötti kommunikációt és adatcserét. Az API definiálja, hogy hogyan lehet más programoknak vagy komponenseknek kommunikálni egy adott szoftverrel vagy rendszerrel. Az API lehet belső vagy külső. A belső API-kat egy adott szoftver vagy rendszer komponensei közötti kommunikációhoz használják, míg a külső API-k a szoftver vagy rendszer más alkalmazásokkal, szolgáltatásokkal vagy fejlesztőkkel való kommunikációját teszik lehetővé.

### 3.1.6. REST API

A REST<sup>4</sup> API [5] [6] egy olyan "módszer", amellyel alkalmazások kommunikálhatnak egymással a weben keresztül. Egyfajta szabványosított megközelítés, amely lehetővé teszi az alkalmazásoknak, hogy kéréseket küldjenek és válaszokat kapjanak egymástól.

---

<sup>3</sup>Application Programming Interface

<sup>4</sup>Representational State Transfer

A REST API alapelvei egyszerűek és jól meghatározottak. Egy egységes és konzisz-tens interfész biztosít az alkalmazások közötti kommunikációhoz. Az alkalmazásoknak csak az erőforrások azonosítóját kell ismerniük, és a kérésekben át kell adniuk a szükséges információkat. A válaszokat általában formázott adatformátumban, például JSON<sup>5</sup>vagy XML<sup>6</sup> formájában kapjuk vissza.

A REST API nagyon elterjedt a webes alkalmazások fejlesztésében, mivel egyszerű és hatékony módja a különböző alkalmazások integrálásának. Lehetővé teszi az adatok és szolgáltatások könnyű elérését és manipulálását más alkalmazásokban, például mobilalkalmazásokban vagy harmadik fél szoftverekben.

*Metódus:* A HTTP metódus közli a szerverrel, hogy mit kell tennie az erőforrással. A HTTP-ben a négy legáltalánosabb metódusok, amelyet a REST-alapú architektúrában használnak, azok a GET, POST, PUT és DELETE. Ezek megfelelnek a létrehozási, olvasási, frissítési és törlési (vagy Spring Bootba elterjed megnevezése: CRUD<sup>7</sup>) műveleteknek.

- *GET:* A kliensek a GET metódus segítségével érik el az erőforrásokat, amelyek a kiszolgáló megadott URL-címén találhatók. Küldhetnek paramétereket a kérésbe, hogy elérjék az adatok szűrését a paraméter szerint. Egy JSON formátumú választ és egy 200-as (OK) HTTP válaszkódot ad vissza. Amennyiben hibás a kérés a 404 (NOT FOUND) vagy 400 (BAD REQUEST) kódot adja vissza.
- *POST:* A kliensek a POST metódus segítségével küldik el az adatokat a szerver-nek. Sikeres adat küldés esetén 200-ast ad vissza.
- *PUT:* A szerveren egy meglévő entitás módosítására szolgál. Sikeres frissítés esetén 200-as kóddal tér vissza.
- *DELETE:* A kliensek a DELETE metódus segítségével törölhetnek egy erőforrást. Sikeres törlés esetén 200-ast térit vissza.

*HTTP fejlécek:* A kérések fejlécei a kliens és a szerver közötti metaadatok

### 3.1.7. Spring és Spring boot összehasonlítása

A Spring:[7] egy Java alapú keretrendszer, amelyet az alkalmazások fejlesztésének megkönnyítésére és az alkalmazások rugalmasabbá tételere használnak. Különböző új technikákat használ, mint például az aspektus-orientált programozás, a POJO<sup>8</sup> és a függőség befelekendezés. A Springben a konfiguráció lehet XML-alapú vagy Java alapú annotációkkal történik. A fejlesztőnek részletesen meghatározna kell a beaneket és a függőségeket, és ezek kezeléséhez általában a Maven vagy a Gradle build eszközöket használják. Az alkalmazásokat külső webszerverrel pld Tomcat kell telepíteni és üzemeltetni. Számos modult tartalmaz

---

<sup>5</sup>(JavaScript Object Notation) A JSON egy kulcs-érték párosokból felépülő objektum.

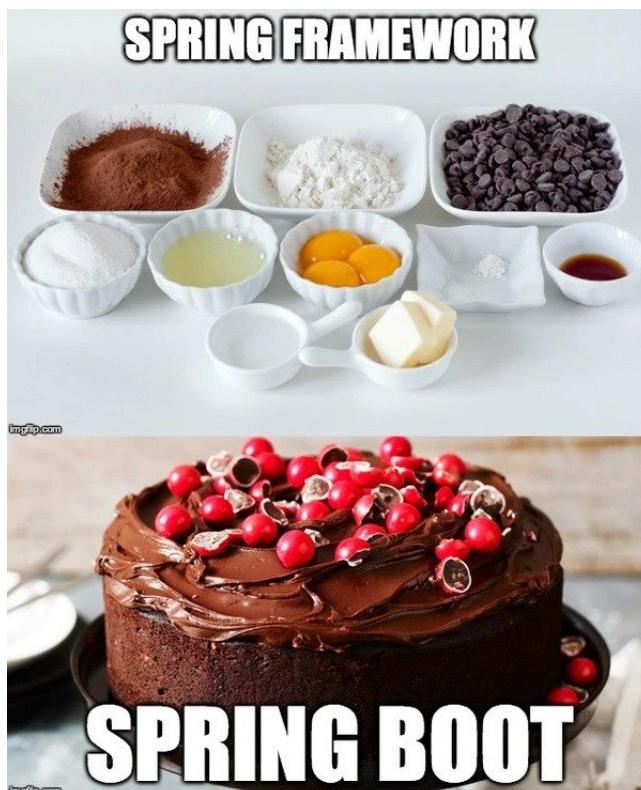
<sup>6</sup>(Extensible Markup Language) egy leíró nyelv, amelyet strukturált adatok tárolására és átvitelére használnak.

<sup>7</sup>create, read, update, delete

<sup>8</sup>Plain Old Java Object

- *Spring Core*: Az alapvető modul, amely a keretrendszer alapjait nyújtja. Tartalmazza a függőségi befelektetés mechanizmusát
- *Spring MVC*<sup>9</sup>: lehetővé teszi az alkalmazások felhasználói felületének tervezését és a kérések feldolgozását a kliensektől.
- *Spring Security*: Lehetővé teszi a felhasználókezelést, az autentikációt és az autorizációt.

*Spring boot*: A Spring Boot a Spring egyfajta tovább gondolása, amely az alkalmazásfejlesztést tovább egyszerűsíti és gyorsítja. minden automatikusan konfigurálódik, ez azt jelenti, hogy az alapértelmezett beállításokkal rendelkező alkalmazásokat könnyen lárthatunk, és nem kell sok időt tölteni a konfigurációval. A Spring Boot projektben előre definiált függőségek és modulok találhatók, amelyek segítenek az adatbázisokhoz, a webalkalmazásokhoz, a biztonsághoz és más komponensekhez történő könnyű hozzáférésben. Az alkalmazások indítása szintén egyszerű a Spring Boot segítségével és nem szükséges külön webszerver telepítése és konfigurálása. Ez a kép elég jól illusztrálja a kettő viszonyát:



A Spring Boot a torta, amiben a Spring keretrendszer különböző összetevőit (alapanyagok) már összerakták. Tehát ha jól ismerjük a Springes technológiákat, akkor a Boot csak megkönnyíti a dolgunkat.

---

<sup>9</sup>(Model View Controller)

### 3.1.8. Project Lombok

A *Project Lombok* [8] egy annotációalapú nyílt forráskódú Java-könyvtár, amely segít a fejlesztőknek egyszerűsíteni a kód írását és növelni a produktivitást ezzel lehetővé teszi a forráskód csökkentését. A fejlesztők számára automatikus kódgenerálást és különböző annotációkat biztosítson, hogy csökkentse a felesleges boilerplate kódot és ismétlődő munkát. A Lombok használatával a programozók egyszerűen hozhatnak létre gettereket, settereket, konstruktorokat, a *toString()*, az *equals()* és a *hashCode()* metódusokat, vagy éppen a *builder* tervezési minta implementálását annotációk hozzáadásával, ahogy ezt a 3.3 kódrészlet szemlélteti. A Lombok az annotációkat használja a kódgeneráláshoz, amelyeket a Java forráskódba helyeznek. Az annotációk hatására előfeldolgozza a forráskódot és a szükséges kódokat automatikusan generálja. Ez csökkenti a redundáns kód mennyiségét, javítja a kód olvashatóságát és karbantarthatóságát. A lombok használatához egy Maven függőséget kell hozzáadni a projekthez:

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

**3.2. kódrészlet.** Használatához a következő sorokat kell beírni a pom.xml-be

```
package cityissue.tracker.murestrack.persistence.model;

import lombok.*;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.validator.constraints.Length;
import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "'NotificationType'")
@Data
@ToString(exclude = "notifications")
@Builder(toBuilder = true)
@NoArgsConstructor
@AllArgsConstructor
public class NotificationType {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "native")
    @GenericGenerator(name = "native", strategy = "native")
    @Column(name = "ID")
    private Long id;

    @Column(name = "name", length = 30)
    @Length(max = 30)
    private String name;
```

```

    @OneToMany(
        mappedBy = "notificationType",
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    private List<Notification> notifications = new ArrayList<>();

```

### 3.3. kódrészlet. Példa a Lombok annotációra

#### 3.1.9. Entity osztályok

Az *Entity* osztályok összekapcsolódnak az adatbázis tábláival, és a táblák sorai reprezentálják az adott entitás példányait. Az Entity osztályok általában adatműveleteket is tartalmazhatnak, például az adatok mentését, frissítését, törlését vagy lekérdezését végző metódusokat.

Az Entity osztályok nagyon fontos szerepet játszanak az alkalmazások adatkezelésében és az adatbázisokkal való kommunikációban. A Java nyelvben a *JPA*<sup>10</sup> specifikáció segítségével, valamint a Spring keretrendszerben a *Spring Data* modul használatával könnyedén lehet kezeln az Entity osztályokat és az adatbázis műveleteket. *@Entity* annotációval vannak ellátva. Ezen kívül található még a *@Table* annotáció is, amivel tábla nevét is meg tudjuk adni.

```

@Entity
@Table(name = "Permission")
@Data
@Builder(toBuilder = true)
@NoArgsConstructor
@AllArgsConstructor
public class Permission {
    ...
}

```

### 3.4. kódrészlet. Példa az entitás definiálására

A *@Column* és *@Id* annotációk a *JPA* részét képezik, és a relációs adatbázisban tárolt entitások osztályainak mezőit vagy tulajdonságait jelölik. A *@Column* annotációval jelölt mezők azt jelzik, hogy az adott osztályban lévő adattagot egy adatbázis oszlop reprezentálja. Az annotáció segítségével a fejlesztő beállíthatja az oszlop nevét, típusát, hosszát, egyedülállóságát és más tulajdonságait. Az *@Id* annotáció segít az adott entitás egyedi azonosításában és az adatbázisban való hivatkozásában. Az automatikus id generálás érdekében több féle annotáció létezik: *@GeneratedValue*, *@SequenceGenerator* és *@TableGenerator*.

```

@Id
@GeneratedValue(strategy = GenerationType.AUTO, generator = "native")
@GenericGenerator(name = "native", strategy = "native")

```

<sup>10</sup>Java Persistence API

```
@Column(name = "ID")
private Long id;
```

### 3.5. kódrészlet. Példa a @GenericGenerator használatára

#### 3.1.9.1. DTO osztályok és konverterek

*@DTO*<sup>11</sup> egy olyan osztály, amelyet az adatok továbbítására és megosztására használnak különböző rétegek vagy komponensek között az alkalmazásban, mivel a táblákba beszúrt adatok és a lekért adatok nem mindenkor egyeznek meg az Entity osztály struktúrájával. Ezekben csak az adatok vannak és nincsenek azonosítók és jelszavak.

Ahhoz, hogy a Repositoryba található függvények helyesen tudjanak működni, szükséges egy *@Converter* a DTO és Entity osztályok között. A konverterek felelősek az adatok átváltásáért, például egy entitás osztályból vagy adatbázis objektumból egy DTO osztályba, vagy fordítva. Ebbe a konverter osztályba statikus függvényeket használunk. A Lombok biztosítja számunkra a *Builder* tervezési minta egy implementációját, ezért egyszerűen és átlathatóan tudjuk az objektumainkat konvertálni, ahogy ezt a 3.6 kódblokkban is láthatjuk.

```
public static ReportDto convertToDTO(Report report){
    return ReportDto.builder()
        .id(report.getId())
        .title(report.getTitle())
        .description(report.getDescription())
        .category(report.getCategory())
        .location(report.getLocation())
        .targetDate(report.getTargetDate())
        .severity(report.getSeverity())
        .creator(report.getCreator().getUsername())
        .assignee(report.getAssignee().getUsername())
        .status(report.getStatus())
        .attachmentDTO(attachmentDTO)
        .build();
}
```

### 3.6. kódrészlet. Példa a builder használatára

#### 3.1.10. Spring Boot alkalmazás felépítése

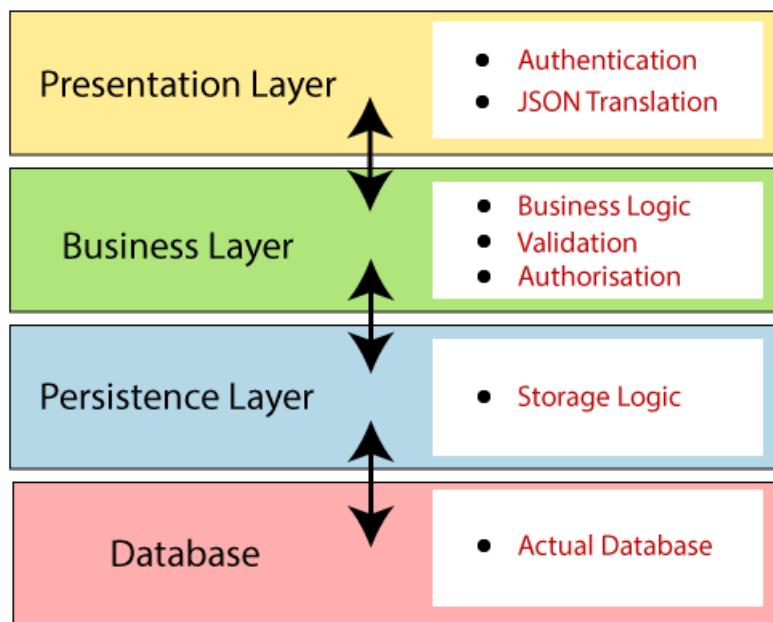
A Spring Boot architektúrája [9] hierarchikus felépítésű, ahol minden réteg közvetlenül alatta vagy fölötté lévő réteggel kommunikál. A **Presentation-Business-Persistence-Database** rétegstruktúra általános megközelítése:

1. **Presentation Layer:** Ez a réteg felelős a felhasználói felület megjelenítéséért. Ide tartoznak a vezérlők (controllers), a nézetek (views) és a felhasználói interfések

<sup>11</sup>Data Transfer Object

(UI). Továbbá ez a réteg értelmezi a JSON-t<sup>12</sup> és kezeli az autentikációt és az HTTP kéréseket. Az autentikációt követően továbbítja az üzleti rétegbe a további feldolgozás céljából.

- Business Layer:** Ez a réteg tartalmazza az üzleti logikát és a feldolgozási szabályokat. Itt valósulnak meg az üzleti folyamatok, a műveletek és az adatmanipulációk. Ide tartoznak a szolgáltatások (services), amelyek végzik az üzleti logika végrehajtását.
  - Persistence Layer:** Ez a réteg kapcsolódik az adatbázishoz vagy más adatforrásokhoz, és felelős az adatok tárolásáért és lekérdezéséért. Ide tartoznak az adatelérési réteghez (repository) tartozó osztályok.
  - Database Layer:** Ez a réteg az adatok fizikailag tárolja és kezeli az adatbázisban. Ide tartoznak az adatbázis rendszerek, például az SQL alapú adatbázisok vagy NoSQL adatbázisok.



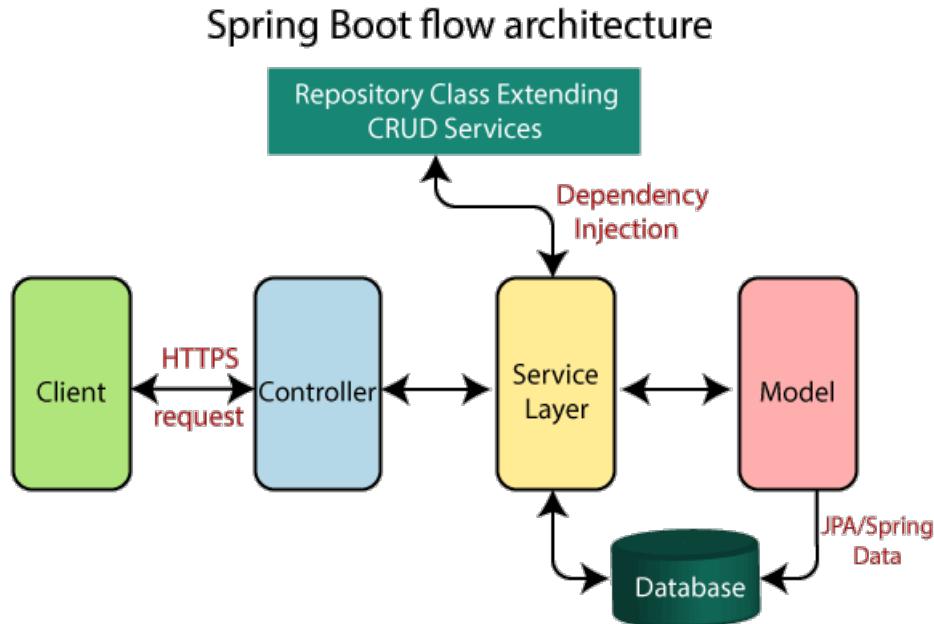
**3.3. ábra.** A Presentation-Business-Persistence-Database megközelítés

A Repository-Service-Controller rétegstruktúra a Spring keretrendszer specifikus megközelítése:

1. **Repository:** Ez a legalacsonyabb réteg, és felelős az adatbázissal vagy más adatforrásokkal történő kommunikációért. Spring Data JPA használata leegyszerűsít a dolgunkat.
  2. **Service:** Ez a köztes réteg az üzleti logika végrehajtásáért felelős. A Service réteg kapcsolódik a Repository réteghez, és tartalmazza a fő logikát, amelyet az alkalmazás működtetéséhez szükséges. Itt valósul meg az adatok feldolgozása

## <sup>12</sup> JavaScript Object Notation

3. **Controller:** Ez a legfelső réteg, amely közvetlenül kommunikál a klienssel HTTP kérések formájában. Befogadja a bejövő kéréseket, meghívja a Service rétegen megírt függvényeket majd visszatéríti a választ a kliensnek.



**3.4. ábra.** A Repository-Service-Controller rétegstruktúra megközelítés

### 3.1.11. JPA Repository

A *JPA Repository* [10] egy interfész, amely megkönnyíti az adatbázis-műveletek végrehajtását. Különböző adatbázis-műveletek, mint például adatok lekérdezése, beszúrása, frissítése vagy törlése végrehajtására szolgál, tehát a CRUD műveletekre. A JPA Repository-nak számos beépített metódusa van, amelyek lehetővé teszik az adatok lekérdezését, rendezését és korlátozását. Emellett lehetőség van egyedi lekérdezések definiálására is, mint például a findByUsername a fenti példában. Ahhoz, hogy használni tudjuk:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

**3.7. kódrészlet.** spring-boot-starter-data-jpa pom.xml-be

Az alkalmazás Repository rétegébe használjuk, amely ugye az adatbázissal kommunikál. Ehhez *@Repository* ellátott interfésekkel hozunk létre és minden entitásnak fog kelleni egy repository interfész. Ez pedig származtatva lesz a JpaRepository osztályból, és ezt paraméterezzük az entitás osztály nevével, és az entitás elsődleges kulcsának típusával.

A következő példa szemléltet egy ilyen Repository-t a User entitásra, melynek elsődleges kulcsa egy Long típusú érték:

```
@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
    Optional<User> findByUsername(String uname);  
}
```

### 3.8. kódrészlet. JPA Repository kiterjesztése

Amennyiben a JPARespositoryt használjuk a Repositoryba elég függvény fejlécét megadjuk és nem szükséges implementálnunk se.

#### 3.1.12. Kivételkezelés

A kivételkezelés nagyon fontos része egy Spring Boot alkalmazásnak, és a Spring keretrendszer számos beépített lehetőséget kínál a kivételkezelésre. Lehetséges a fent említett rétegekbe külön kezelní a hibákat, azoknak különböző fajtáit. Viszont a *try catch* struktúra helyett, lehet globális kivételkezelőt használni mint pld *@ExceptionHandler, @ControllerAdvice*. TODO Alkalmazásom fejlesztése során a *ResponseEntity*t használtam, amely lehetővé teszi a HTTP válasz testreszabását és a kliens számára visszaküldött adatok kezelését. Az ResponseEntity objektum segítségével definiálhatjuk a válasz HTTP státuszkódját, fejlécét és body részét. Amikor a válasz sikeres volt, a *ResponseEntity.status(ResponseEntity.OK)* metódusával hozhatjuk létre az OK (200) státuszkódot, amennyiben valami hiba lépett fel, akkor a *ResponseEntity.status(HttpStatus.UNAUTHORIZED)* metódust a megfelelő HTTP státuszkódot küldtem. Az ResponseEntity objektum továbbá lehetővé teszi az adatok, például objektumok, listák vagy üzenetek visszaküldésé, tetszőleges üzenetet(ResponseMessage), amit a kliensnek küldtem vissza:

```
public ResponseEntity<ResponseMessage> upload(@RequestParam("file")  
        MultipartFile file) {  
    String message = "";  
    try{  
        Long id = attachmentService.saveFile(file);  
        return ResponseEntity.status(HttpStatus.OK).body(new  
            ResponseMessage(id + ""));  
    } catch (Exception e) {  
        message = "Could not upload the file: " +  
            file.getOriginalFilename() + "!";  
        return ResponseEntity.status(HttpStatus.EXPECTATION_FAILED)  
            .body(new ResponseMessage(message));  
    }  
}
```

### 3.9. kódrészlet. Kivételkezelés a MuresTracker szoftvernél

### 3.1.13. Spring security

A *SecurityConfig* [11] osztály egy Spring Security konfigurációs osztály, amelyet a *WebSecurityConfigurerAdapter* osztályból származtatunk. Ennek a feladata, hogy letiltsa vagy engedélyezzen bizonyos hívásokat, attól függően hogy a felhasználó milyen jogokkal rendelkezik, vagy be van-e jelentkezve. Ezeket a tokenból állapítja meg. A példában jól látható, hogy a *configure* metódus felülírása révén konfiguráljuk a *HttpSecurity* objektumot, amely az HTTP kéréseket kezeli a biztonsági szempontból.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity security) throws Exception
    {
        security.formLogin().disable();
        security.csrf().disable()
            .addFilterAfter(new JWTAuthorizationFilter(),
                UsernamePasswordAuthenticationFilter.class)
            .authorizeRequests()
            .antMatchers(HttpMethod.POST, "/login").permitAll()
            .antMatchers(HttpMethod.GET, "/websocket/**").permitAll()
            .regexMatchers(HttpMethod.PATCH,
                "/users/status/\\w+(/?)?").permitAll()
            .anyRequest().authenticated()
            .and()
            .cors();
    }
}
```

**3.10. kódrészlet.** Példa a Spring Security configure implementációjára

1. **security.formLogin().disable():** A bejelentkezási űrlapot letiltjuk, így a Spring Security nem fogja kezelni a bejelentkezási oldalt.
2. **security.addFilterAfter(new JWTAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class):** Hozzáadunk egy JWTAuthorizationFilter nevű saját filtert a Spring Security szűrőláncához. Ez a filter felelős a JWT alapú hitelesítés és az autorizáció kezeléséért.
3. **security.authorizeRequests() .antMatchers(HttpMethod.POST,"/login") .permitAll():** Engedélyezzük a "/login" POST kérést mindenki számára.
4. **security.anyRequest().authenticated():** minden más kérést csak bejelentkezett felhasználók számára engedélyezünk.

## 3.2. Kliens oldalon használt technológiák

### 3.2.1. Angular alkalmazás felépítése

A weboldal kliens részéhez az Angular frameworkot használtam, ami MVVM<sup>13</sup> architektúrán alapszik.

- **Modell osztályok** - tartalmazzák az alkalmazás adatait és azok metódusait. Ezek a *TypeScript* osztályok, amelyek hasonlóak a szerver oldalon használt DTO típusú osztályokhoz.
- **Nézet** - a nézetek meghatározzák, hogy hogyan jelennek meg az adatok és a felhasználói felület elemek a böngészőben. A nézetet HTML és CSS fájlok képezik
- **ViewModel** - összekapcsolja a modell osztályokat és a nézeteket. A komponensek felelősek az adatok kezeléséért, a business logika végrehajtásáért és az események kezeléséért.

### 3.2.2. TypeScript

A TypeScript[12] egy objektum-orientált script nyelv, amit a Microsoft készített 2012-ben. Legfőbb filozófiája nyelvnek az, hogy "legyen a TypeScript bővebb halmaza a Javascript-nek", ami azt jelenti hogy bármilyen JavaScriptben írt kód a TypeScriptba is érhető. A cél egy olyan eszköz elkészítése volt, mely segíti a fejlesztőket az igazán nagy projektek elkészítésében is és hogy javítsa a komplex alkalmazások fejlesztésének folyamatát. Statikusan típusos, nyílt forráskódú nyelv, illetve operációsrendszer független. Viszont pluszba olyan funkciókkal egészítették ki, mint az objektumorientált programozás.

### 3.2.3. Angular Routing

Az *Angular Routing* [13] az Angular keretrendszer egyik fontos funkciója, amely lehetővé teszi a kliensoldali alkalmazások számára, hogy a böngészőben történő navigáció során különböző oldalakat jelenítsenek meg, anélkül, hogy az egész alkalmazást újra kellene betölteni. Ezáltal segít az alkalmazás sima és gyors navigálásában, és jobb felhasználói élményt nyújt. Ehhez csupán a *RouterModule*-t kell importáljuk és a *RouterModule.forRoot()* metódus hívásával definiálhassuk az útvonalainkat. A gyakori Angular Routing műveletek közé tartozik *routing* paramétereinek átadása, védése a hozzáférési jogosultságok alapján, a paraméterek kiolvasása és a programozott irányítása az alkalmazásból.

```
const routes: Routes = [
  {
    path: '',
    component: ReportListComponent
  },
  {
    path: 'report/:id',
    component: ReportDetailComponent
  }
]
```

<sup>13</sup>Model-view-viewmodel

```

    path: 'add',
    component: AddReportComponent
},
{
  path: 'update/:id',
  component: EditReportComponent
}
];

```

### 3.11. kódrészlet. Példa a routing-ra

*Child Routes* Ezek az útvonalak hierarchikus struktúrát alkotnak, ahol a gyermek útvonalak a szülő útvonal alá tartoznak. Lehetővé teszi az alkalmazás számára, hogy többszintű navigációt valósítson meg, ahol az egyes komponensek vagy oldalak egymásba ágyazódnak. A gyermek útvonalak létrehozása az Angular Routingban a routing beállításokban történik. A szülő útvonalhoz rendelhetünk egy vagy több gyermek útvonalat, és ezeket a *children* tulajdonság segítségével definiálhatjuk.

```

const routes: Routes = [
  {
    path: '',
    component: AppLayoutComponent,
    children: [
      {
        path: '',
        component: DashboardComponent
      },
      { path: 'reports', loadChildren: () =>
          import('../report/report.module').then(m => m.ReportModule),
          canActivate: [AuthGuard], data:{permissions: ["REPORT_MANAGEMENT"]} },
      { path: 'user', loadChildren: () => import('../user/user.module').then(m => m.UserModule), canActivate:[AuthGuard] , data: {permissions: ["USER_MANAGEMENT"]} },
      {
        path: 'permissions',
        loadChildren: () => import('../permission/permission.module').then(m => m.PermissionModule),
        canActivate: [AuthGuard],
        data: {permissions: ["PERMISSION_MANAGEMENT"]}
      },
      { path: 'notifications', loadChildren: () =>
          import('../notification/notification.module').then(m => m.NotificationModule), canActivate:[AuthGuard] }
    ],
  }];

```

### 3.12. kódrészlet. Példa a routing-ra és a Guardok használatára

### 3.2.4. Guardok

Az Angular keretrendszerben a *Guardok* [14] olyan komponensek, amelyek segítségével megvalósítható hogy egy adott komponenshez milyen feltételek alapján lehet hozzáérni. A guardok használatával ellenőrizhetjük a felhasználó jogosultságait, az autentikációt és hitelesítést. Amikor egy routra vagy komponensre navigálunk, a guardok értékelődnek ki, és meghatározzák, hogy a navigáció engedélyezett-e vagy sem. Az Angular keretrendszerben négyféle alapvető guard típus áll rendelkezésre:

- **CanActivate** - meghatározza, hogy az adott útvonalhoz vagy komponenshez való hozzáférés engedélyezett-e. Pld a felhasználó be van-e jelentkezve vagy rendelkezik bizonyos joggal.
- **CanActivateChild** - hasonlóan működik, mint a CanActivate, azonban csak a gyermek útvonalakhoz való hozzáférést ellenőrzi.
- **CanDeactivate** - meghatározza, hogy a felhasználó elhagyhatja-e az adott útvonalat, megakadályozva ezzel az adatvesztést
- **CanLoad** - meghatározza, hogy a modult, ahol az adott útvonal található, betölthető-e

Ezekre példát a [3.12](#) kódrészleten lehet találni.

### 3.2.5. PrimeNG

*PrimeNG* [15] egy nyílt forráskódú UI<sup>14</sup> komponenskönyvtár, amely a TypeScript alapú Angular keretrendszerhez készült. A PrimeNG gazdag és készre formázott UI komponenseket kínál, amelyek segítségével könnyedén és hatékonyan építhetünk felhasználóbarát, modern webalkalmazásokat.

A PrimeNG számos előre elkészített komponenst tartalmaz, például gombok, táblázatok, navigációs menük, panelek, dialógusablakok, toastok, diagramok, naptárak, form-hoz különböző inputok stb. Ezeket a komponenseket könnyen beilleszthetjük az Angular alkalmazásunkba, és testreszabhatjuk az igényeinknek megfelelően.

A PrimeNG komponensek kiváló minőségűek, jól dokumentáltak és könnyen használhatók. A komponensek széles skálája lehetővé teszi az alkalmazások gyors és hatékony fejlesztését, miközben egy egységes és esztétikus felhasználói élményt nyújtanak.

Egyes PrimeNG komponensek extra funkciókkal és testreszabhatósággal rendelkeznek, mint például adattáblázatok adatforrásokkal, szűrési lehetőségekkel és rendezési lehetőségekkel. Emellett a PrimeNG támogatja a reszponzív tervezést és a mobilbarát felhasználói felületek kialakítását is.

Ahhoz hogy igénybe tudjuk venni a PrimeNg előnyeit:

```
npm install primeng  
npm install primeicons
```

### 3.13. kódrészlet. PrimeNG installáció

<sup>14</sup>felhasználói felület

Fontosabb modulok:

- **ButtonModule** - gombok,
- **HttpClientModule** - Http metódusok hívásáért felelős,
- **FormsModule** - űrlapok készítésére,
- **InputTextModule** - szöveges beviteli mezők,
- **CalendarModule** - naptár,
- **DialogModule** - felugró ablak,
- **FileUploadModule** - fájl feltöltés,
- **PaginatorModule** - lapozás,
- **ToastModule** - hiba és sikeres üzenetek megjelenítésére,
- **MenuModule** - menük létrehozásához,
- **ChartModule** - diagramok elkészítéséhez,
- **TranslateModule** - fordítás megvalósításához.

### 3.2.6. Nemzetköziesítés és az i18n

Az alkalmazás különböző nyelveken való működése nagyon egyszerűen megvalósítható, csak a *i18n*<sup>15</sup> modulra van szükségünk, ami minden *translate* szóval jelölt címet, menüsor, üzenetet le fordít. A fordításokat JSON állományokban tároljuk. (pl. en.json, ro.json, hu.json). Kulcs-érték párból adjuk meg azt amit a kódban használunk és annak a fordítása az adott nyelven. A nyelvek közötti váltást egy kis legördülő menüvel tudunk megvalósítani, ahol a menüelemekre a TranslateService *use* függvényét hívjuk meg.

```
constructor(public layoutService: LayoutService,
            public translate: TranslateService) {
  translateService.addLangs(['en', 'ro', 'hu']);
  translateService.setDefaultLang('en');
}

useLanguage(language: string) {
  this.translate.use(language);
}

onLangChange(langEvent: any): void {
  this.translateService.use(langEvent.value);
  this.updateTranslations();
}
```

#### 3.14. kódrészlet. Translate menü

<sup>15</sup>internationalization

## 4. fejezet

# A rendszer specifikációja

Egy szoftver tervezése során sokszor nagyon nehéz pontosan leírni, hogy hogyan kell működnie a rendszernek. Ezért több különböző leírást fogalmazunk meg amit követelményeknek nevezzük. Ez az elemzés és tervezés alapegységeként értelmezhető, amelyből a kész rendszert építjük fel. A követelmények leírása pedig az a folyamat, amely során meghatározzuk és elemezzük a szolgáltatásokat és megszorításokat, majd dokumentáljuk és ellenőrzük azokat. A követelmény elnevezés nagyon tág fogalom, így a továbbiakban két különböző megközelítésben bontsuk le.

### 4.1. Felhasználói követelmények

A felhasználói követelményekbe azokat az eseteket mutatom be, amelyekkel a felhasználó találkozhat az alkalmazás használata közben. A rendszert öt különböző felhasználó típus használná.

- **Adminisztrátor** - Az Ő feladata, hogy minden az elvárásoknak megfelelően működjön, ami az alkalmazást illeti. mindenhez joga van, létrehozhat új felhasználókat-bejelentéseket, módosíthatja a részleteket és törölheti is őket. Emellett pedig feljegosíthat különböző típusú felhasználókat.
- **Felügyelő** - A Felügyelő pedig egy moderátor szerepet játszik, aki felügyeli az emberek bejelentéseit, kijavítva azokat amennyiben helyetlenül címkezik fel őket, vagy nem a megfelelő szerveknek adják oda. Miután beérkezik egy jelentés mindenkorábban a moderátor átnézi és korrigálja ott ahol kell. Biztosításak a közösségi tér biztonságát és kényelmét azáltal, hogy beavatkozhat azokban az esetekben, amikor a tagok megsértik a közösségi szabályokat vagy más tagokat zaklatnak, bántalmaznak, vagy sértő módon nyilvánulnak meg. Amennyiben ez előfordul inaktívvá tehetik ezeket a felhasználókat.
- **Városi hivatalnok felhasználók** - A városi hivatalnok felhasználók a városi kor-mányzat jogosult munkatársai, akik a weboldalt a jelentett problémák kezelésére és megválasztására használják. Láthatják és kezelhetik az összes jelentett problémát, ami hozzájuk tartozik. Frissíthetik a jelentés állapotát. Ők a probléma megoldásáért és nyomon követéséért felelősek, a felhasználókhöz nincs közük.

SUPERVISOR	X	▼	EMPLOYEE	X	▼
<b>Permissions</b>					
<input type="checkbox"/> PERMISSION MANAGEMENT			<input type="checkbox"/> PERMISSION MANAGEMENT		
<input checked="" type="checkbox"/> USER MANAGEMENT			<input type="checkbox"/> USER MANAGEMENT		
<input checked="" type="checkbox"/> REPORT MANAGEMENT			<input checked="" type="checkbox"/> REPORT MANAGEMENT		
<input type="checkbox"/> REPORT CLOSE			<input type="checkbox"/> REPORT CLOSE		
<input type="checkbox"/> REPORT EXPORT PDF			<input type="checkbox"/> REPORT EXPORT PDF		

**4.1. ábra.** Felügyelői szerepkör jogai

**4.2. ábra.** Városi hivatalnok szerepkör jog

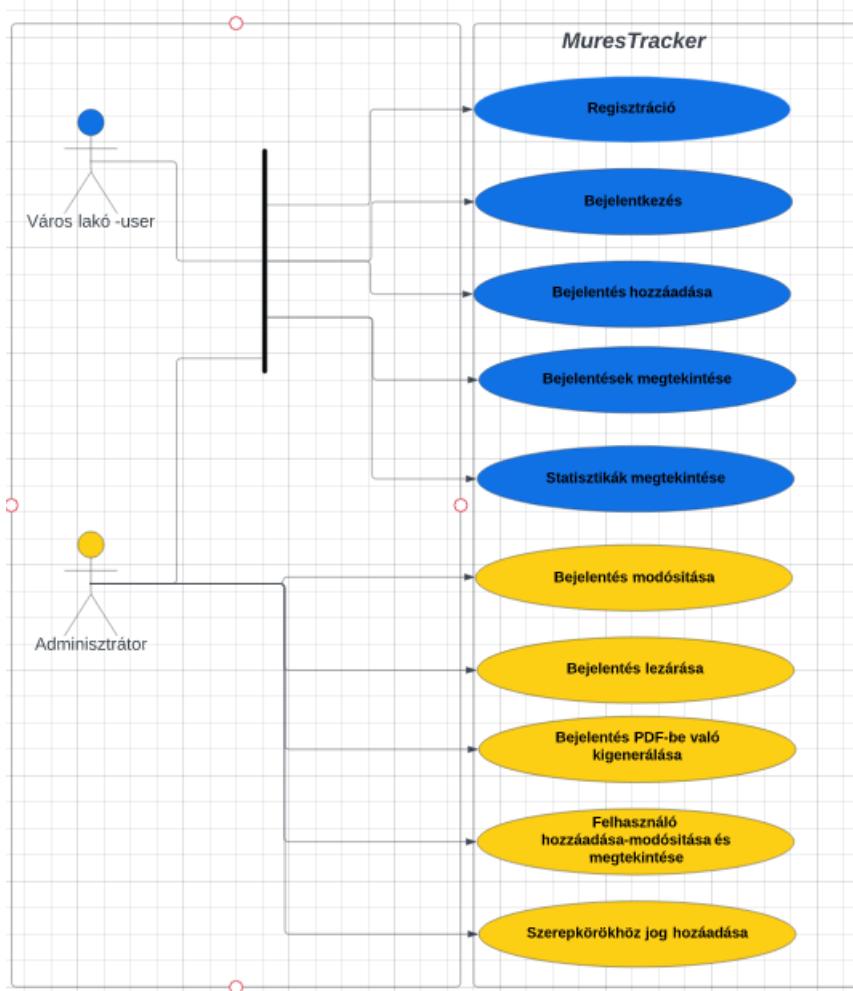
ANALYST	X	▼	CITIZEN	X	▼
<b>Permissions</b>					
<input type="checkbox"/> PERMISSION MANAGEMENT			<input type="checkbox"/> PERMISSION MANAGEMENT		
<input type="checkbox"/> USER MANAGEMENT			<input type="checkbox"/> USER MANAGEMENT		
<input checked="" type="checkbox"/> REPORT MANAGEMENT			<input checked="" type="checkbox"/> REPORT MANAGEMENT		
<input checked="" type="checkbox"/> REPORT CLOSE			<input type="checkbox"/> REPORT CLOSE		
<input checked="" type="checkbox"/> REPORT EXPORT PDF			<input type="checkbox"/> REPORT EXPORT PDF		

**4.3. ábra.** Elemzői szerepkör jogai

**4.4. ábra.** A város lakosai szerepkör jog

- **Elemző** - Az elemző kigenerálhatja a bejelentéseket PDF vagy XLSX formátumba. Hozzáférhet a jelentésekhez, hogy minél részletesebb stratégiai elemzéseket és terveket Készítsen.
- **A város lakosai** - Legutolsó sorban a célközönség a városi lakosság, akik bejelentéseket tehetnek, kommunikálhatnak a város vezetőségével vagy az erre kijelölt személyekkel és nyomon követhetik a bejelentések állapotát. Viszont nekik nincs joguk az admin panelhez.

A használati eset diagramja a 4.5 ábrán látható.



4.5. ábra. Használati eset diagram

#### 4.1.1. Fontosabb használati esetek

- **jelentés hozzáadása**
  - előfeltételek:
    - \* PR1: a felhasználó be van jelentkezve

– *folyamat*:

- \* F1: a felhasználó bejelentkezik
- \* F2: a felhasználó rákattint az "Új jelentés hozzáadása" menüpontra
- \* F3: a felhasználó megadja a bejelenteni kívánt jelentés részleteit: címet, kategóriát kiválasztja, helyszínt megadja, leírást, a dátum automatikusan az aznapra menti el, státusz is NEW lesz, a helyzet súlyosságát is kiválassza és odaadja valakinek és opcionálisan képet csatol.
- \* F4: a felhasználó rákattint a "Mentés" gombra

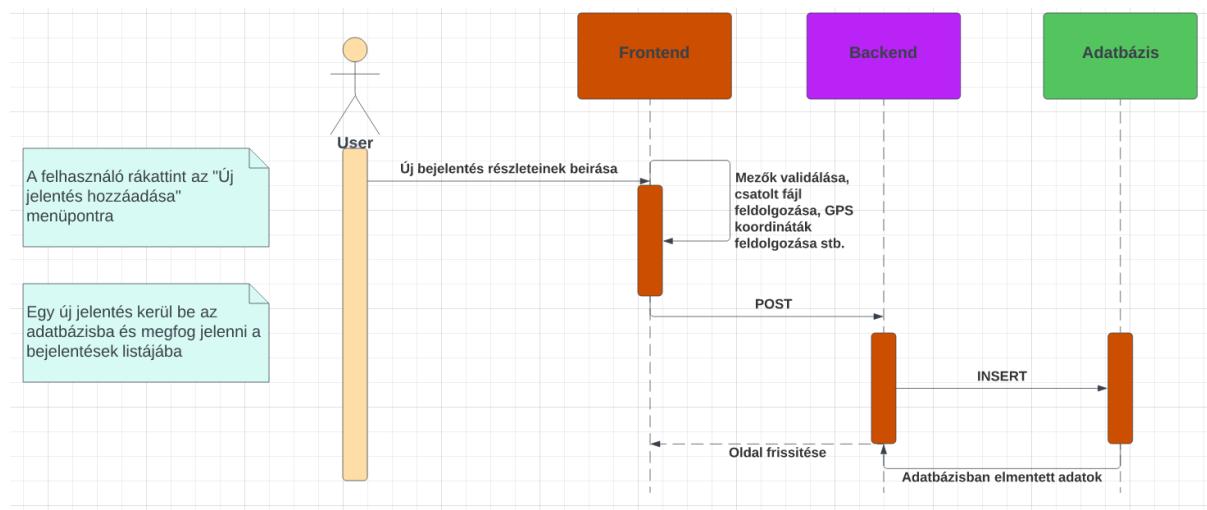
– *alternatív folyamat*:

- \* AF1: ha felhasználó már be van jelentkezve, akkor a folyamat az F2-től kezdődik

– *utófeltételek*:

- \* PO1: a jelentés bekerül az adatbázisba
- \* PO2: az jelentés megjelenik a jelentések listájában
- \* PO3: "A Jelentés sikeresen hozzáadva" Toast megjelenik
- \* PO4: akire "bízták" a jelentést az kap egy értesítést

A folyamat diagramja: [4.6](#)



**4.6. ábra.** Jelentés hozzáadásának szekvenciadiagramja

• **Egy jelentés lezárása**

– *előfeltételek*:

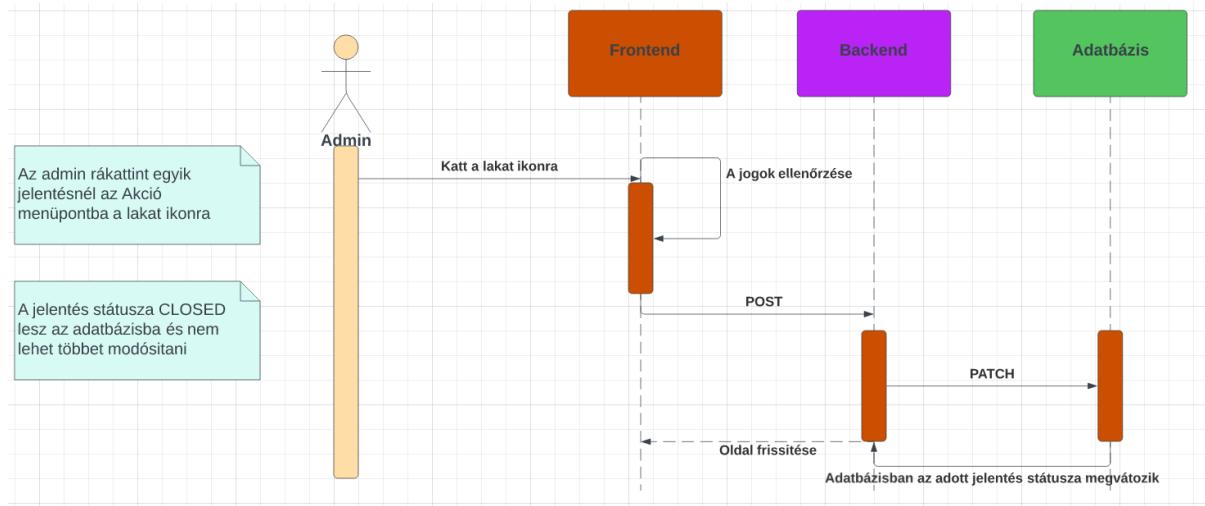
- \* PR1: a felhasználó be van jelentkezve
- \* PR2: a felhasználó admin vagy moderátor szerepkörrel rendelkezik
- \* PR3: a jelentésnek FIXED státusza van

– *folyamat*:

- \* F1: a felhasználó bejelentkezik

- \* F2: a felhasználó rákattint az "Jelentések" menüpontra
- \* F3: a felhasználó rákattint egyik jelentésnél az Akció menüpontba a lakkára
- \* F4: megjelenik a Biztos beakarod zárni a jelentést Toast
- \* F5: a felhasználó rákattint az Igenre
- \* F6: sikeresen lezártad az adott jelentést
- *alternatív folyamat:*
- \* AF1: ha felhasználó már be van jelentkezve, akkor a folyamat az F2-től kezdődik
- *utófeltételek:*
- \* PO1: az adatbázisban a jelentés státusza megváltozik
- \* PO2: a jelentés listába is megváltozik a státusz
- \* PO3: a felhasználó aki hozzáadta a jelentést kap egy értesítést

A folyamat diagramja 4.7



4.7. ábra. Egy jelentés lezárása szekvenciadiagramja

- **felhasználó státuszának megváltoztatása**
  - *előfeltételek:*
    - \* PR1: a felhasználó be van jelentkezve
    - \* PR1: a felhasználó rendelkezik admin joggal
  - *folyamat:*
    - \* F1: a felhasználó bejelentkezik
    - \* F2: a felhasználó rákattint a "Felhasználók" menüpontra
    - \* F3: a felhasználó rákattint az egyik felhasználó Akció menüpontjába lévő Update User ikonra

- \* F4: a felhasználó rákattint a Felhasználó státusza label alatt lévő "toggle" gombra
- \* F5: a felhasználó rákattint a Mentés gombra
- *alternatív folyamat:*
  - \* AF1: ha felhasználó már be van jelentkezve, akkor a folyamat az F2-től kezdődik
- *utófeltételek:*
  - \* PO1: az adatbázisba modósul a felhasználó státusza
  - \* PO2: a listába INACTIV-ként fog szerepelni
  - \* PO3: a felhasználó nem fog tudni bejelentkezni

## 4.2. Rendszerkövetelmények

### 4.2.1. Funkcionális követelmények

A főbb funkcionálitások, és köztük lévő kapcsolat, a [4.8](#) ábrán van szemléltetve.

Az alkalmazás legfőbb funkcionálitásai:

- **Bejelentkezés:** A weboldal használata érdekében be kell jelentkezni. Helyes felhasználónév és jelszó megadása esetén a felhasználó sikeresen be tud lépni. A weboldal minden hibát jelez a felhasználónak (pl. nincs megadva a szükséges inputok valamelyike vagy helytelenül van megadva)
- **Regisztráció:** Amennyiben nem rendelkezünk aktív felhasználóval, a bejelentkezés gomb melletti regisztráció gombbal eljutunk a Regisztráció oldalhoz, ahol meg kell adni az adatait: név, email cím és telefonszám. Ha minden rendbe van, a felhasználó elmentődik az adatbázisba, és kap egy üdvözlő emailt a rendszertől a felhasználó nevével és az idegenes jelszójával.
- **Kijelentkezés:** a header jobboldali részén található Kijelentkezés ikonra kattintva a felhasználó kijelentkezik.
- **Probléma bejelentése:** Lehetővé teszi a felhasználók számára, hogy problémákat küldjenek be részletes információkkal, beleértve a leírást, a helyszínt és lehetőség szerint fényképet.
- **A bejelentett problémák listájának megtekintése:** A bejelentett problémák listázva vannak egy táblázatba és fel van tüntetve az állapotuk így a felhasználók tisztán tudják követni, hogy mi történik az általuk vagy más által bejelentett helyzetekkel. Emellett a bejelentések kategorizálva vannak a bejelentés típusa és sürgőssége alapján + térkép
- **A bejelentett problémák szerkesztése:** Főleg a moderátorok korrigálják a helytelenül beküldött jelentéseket vagy a státuszát frissítik.
- **Statisztikák:** a "Statisztikák" menüpontra kattintva a felhasználó láthatja az elemzők által elkészített érdekes diagramokat a bejelentett adatokról.

#### 4.2.2. Nem funkcionális követelmények

A nem funkcionális követelmények olyan követelmények, amelyek nem közvetlenül a rendszer funkcióira vonatkoznak, hanem az általános minőségi jellemzőkre, teljesítményre, megbízhatóságra, biztonságra, skálázhatóságra vagy egyéb nem-funkcionális területekre vonatkoznak. Ezek a követelmények segítenek meghatározni az alkalmazás tervezésének és fejlesztésének irányát, és meghatározzák az elvárt minőségi szintet. A nemfunkcionális követelmények a rendszer egészére vonatkozó eredő rendszertulajdonságokra koncentrálnak.

- **Termék követelmények**

- *Felhasználói felület (UI)*: Áttekinthető és felhasználóbarát felhasználói felület ami lehetővé teszi, hogy bárki könnyedén tudja használni A Konzisztens megjelenés és elrendezés az oldalon, ami fontos a felhasználói élmény szempontjából, mivel egy egységes és átlátható dizájn segít a felhasználóknak gyorsan



4.8. ábra. Aktivitás diagram

és hatékonyan navigálni az oldalon, megérteni az információkat és könnyedén használni a funkcionálitásokat. Emellett az alkalmazás reszponzív, az oldal jól működik különböző eszközökön és képernyőmréteken. A kinézetet illetően, különös figyelmet szántam a színösszeállításnak, amely a kék színt helyezi előtérbe, ez stabilitást, és megbízhatóságot jelképez, amivel azt sugalljuk, hogy a problémák és hibák megoldására fókusztálunk. A kék szín az égbolt, a tenger színe, az intellektus, az elmélkedés, a megnyugvás kifejezője

- *Megbízhatóság:* Manapság az egyik legfontosabb tulajdonság egy weboldal esetén a sebessége, ami nagyba függ attól mennyire optimalizált. SEO [16] szempontból érdemes értékes és releváns kulcsszavakkal ellátni. A hosszú betöltési idővel rendelkező oldalak kellemetlen és rossz felhasználói élményt nyújtanak. Az oldalam gyorsnak tekinthető, az oldal betöltés nem vesz sok időt igénybe. Az összes erőforrás megfelelően optimizált. "Az oldalsebesség a különböző paraméterek elemzésének összefoglalója, beleértve a képméretet, a CSS méretet, a megjelenítést blokkoló erőforrásokat, a szövegtömörítést, az előzetes betöltséhez szükséges kulcs kéréseket és más elemzéseket." [LINK](#)
- *Biztonság:* az alkalmazásban megadott jelszavak az adatbázisban titkosítva vannak eltárolva. Az 5 különböző szerepkör erre szolgál, hogy az érzékeny felhasználói információk ne legyenek elérhetőek mindenki számára. Emellett az Angular által kinált Guardnak köszönhetően ha nem vagyunk bejelentkezve a megfelelő szerepkörrel rendelkező felhasználóval hiába probáljuk a linket bepotyogni nem tudjuk elérni, a Login oldalra fog dobni. Ezenfelül a backend része is az SQL injection elleni védelemnek megfelelően van lekezelve, hisz egy Postman által se tudják elérni vagy manipulálni az adatokat azoknak akiknek nincs joguk, ezáltal feltörhetetlen és a felhasználók érzékeny adatai.

- **Folyamat követelmények**

- *Kódolási standardok* A kódolási standardok célja, hogy egységesítsék a kód megjelenését és szerkezetét a fejlesztői csapatban. Ez megkönnyíti a kód megértését, karbantarthatóságát és olvashatóságát más fejlesztők és a jövőbeli önmagunk számára.
  - \* camelCase standard alkalmazása változók és függvények esetébe
  - \* dash-case használata a .html, .ts, .scss a kliens oldalon lévő komponensek, modulok, és csomagok elnevezésénél
  - \* hibakezelés és kivételkezelés egységesen
- *Verziókövetés*
  - \* Git verziókövető rendszer használata, azon belül github asztali klienssel
- *Fejlesztési folyamat követése*
  - \* Githubon belül projekt létrehozása
  - \* Trello [17]  
Lebontottam taskokra és Trellon követtem végig a PLANNED, WORK és DONE kártyák segítségével

- **Külső követelmények**

- *Etikai*: az oldal semmilyen adatot nem küldd tovább külső forrásoknak és az érzékeny adatokkal se él vissza.
- *Hardver és Hálózati követelmények*: A MuresTracker oldal bármilyen böngészőből megnyitható, bármilyen eszközön. Internet kapcsolat szükséges mindegyik funkció eléréséhez. A RAM szükséges, hogy a böngésző akadás mentesen működjön az azt megnyitó készüléken és az alkalmazás gond nélküli használatához szükség van Firefox (> 30), Chrome (> 35) vagy Opera(>35) böngészőre.

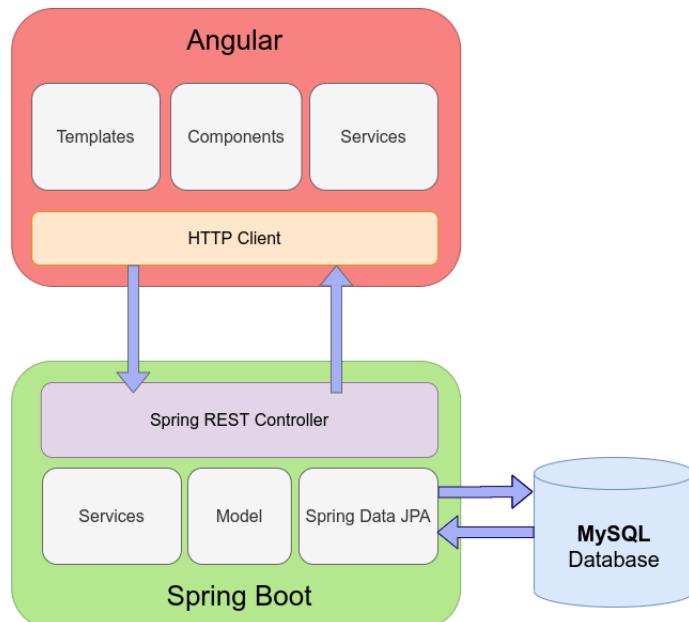
# 5. fejezet

## Tervezés

### 5.1. A rendszer architektúrája

A MuresTracker rendszere három fő komponensből áll : **adatbázis, szerver oldal** (backend) és **kliens oldal** (frontend). Ennek szemléltetése az 5.1 ábrán látható.

- *adatbázis*: legalsó réteg, amely az adatok tárolásáért felelős. A Spring Boot kapcsolódik az adatbázishoz és elvégzi a szükséges műveleteket.
- *szerver oldal*: Spring Boot framework által, tartalmazza a business logikát és az adatkezelésért felelős. A klienstől kapott REST API kéréseket szolgálja ki.
- *kliens oldal*: Angular framework, a felhasználói felületért felelős és az interakciót biztosítja a felhasználó számára. Ez lehetővé teszi a felhasználó számára az adatok megtekintését, szerkesztését és interakciót a rendszerrel.

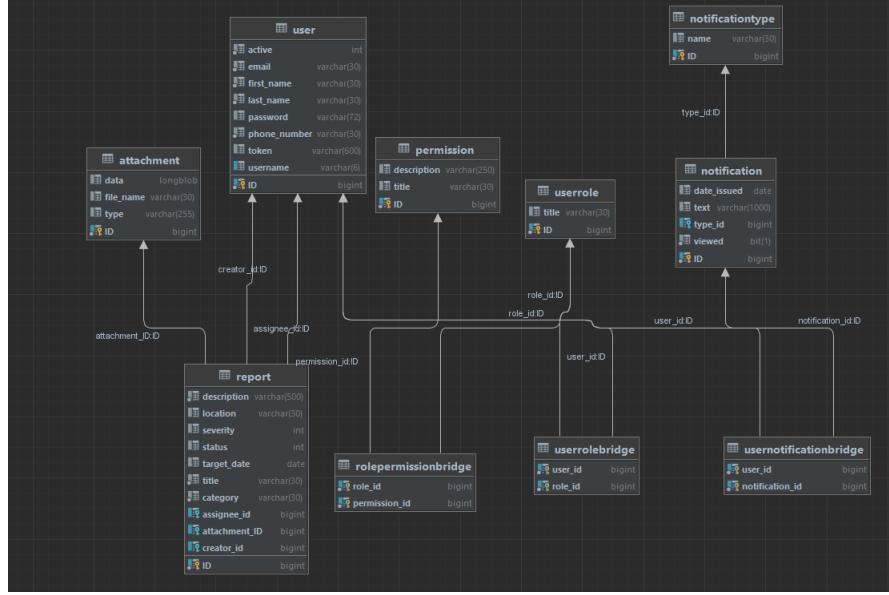


5.1. Ábra. Rendszer architektúrája

## 5.2. Adatbázis

MySQLt használtam az alkalmazásomba, ami egy nyílt forráskódú relációs adatbázis-kezelő rendszer MySQL egyszerűen telepíthető és használható, így nem kellett sok időt ráfordítanom hosszas konfigurációkra. Remekül működik a Spring Bootal, automatikusan konfigurálja. Az adatbázisba a táblák és oszlopok automatikusan jönnek létre az @Entity annotációval ellátott Java osztályokra. A @ManyToMany, @ManyToOne, @OneToMany annotációk segítségével könnyedén feltudjuk állítani kapcsolattípusokat a tábláink között. Emelett képes kezelní nagy adatmennyiségeket és nagy terhelést és jó teljesítményt nyújt adatbázis műveletek során, mint például adatlekérdezések vagy adatok beszúrása, frissítése, törlése.

Az adatbázis táblák szerkezete az 5.2 ábrán látható.



5.2. ábra. Adatbázis táblák szerkezete

## 5.3. Felhasználói interfész tervezése

Egy weboldal prototípusa megmutatja a weboldalon futó funkciók vizuális megjelenítését. A jó munkához idő kell! – tartja a mondás, nem véletlen. A legtöbb esetben azt szeretnénk, hogy amit megálmodtunk, az a lehető legrövidebb időn belül elkészüljön és működjön. Megvannak a nagyvonalak, csak össze kell illeszteni őket. Ez azonban a fejlesztésnél nem működik. Mert a nagyvonalakon mögött rengeteg részlet bújik meg, és pont ezek a részletek fogják meghatározni a felhasználói élményt. A prototípus elkészítése abban segít, hogy ezek a részletek tisztázódjanak, és a felhasználói folyamatok folyamatosan épüljenek egymásra.

# **6. fejezet**

## **Kivitelezés**

Az alkalmazás tervezésének első lépéseiben azt próbáltam meghatározni, hogy ki a célcsoport és hogy igényeket szeretnék ezzel kiszolgálni. E fejezetem célja az alkalmazás elkészítési eljárásom és a működésének bemutatása, annak függvényébe, hogy milyen szerepkörrel rendelkező felhasználó használja. Megkülönböztethető öt típusú felhasználó. De lényeges különbség csak az egyszerű városlakó és Adminisztrátor között van. Regisztráció esetén alapértelmezetten városlakó jön létre ami megfelel egy szimpla usernek. A back-end része az applikációnak magába foglalja a szerver, adatbázisok létrehozását. A front-end része, ami a háttérben létrehozott funkcionálisokat életre kelti a felhasználó szeme előtt.

### **6.1. Kliens oldali megvalósítás**

A kliens oldali megvalósítás az alkalmazás fejlesztési folyamatának azon része, amikor a felhasználók számára látható felhasználói felületet hozzuk létre és kezeljük a felhasználói interakciókat. Jelen esetben egy Angularban megírt MVVM architektúrájú alkalmazás. A html fájlok jelképezik a nézetet, a komponensek a nézetmodellt, valamint a Typescript interfések a modelleket. A szerverrel való kommunikáció a service osztályokon keresztül történik, aszinkron kérések segítségével.

Az alkalmazás minden egyes oldala egy külön kis Angular komponens, amelyek beszédes névvel rendelkeznek és így elkerülöm a nagy terjedelmű fájlokat. Ezeket a három különböző mappára osztottam le Components, ami áll egy component.ts, html és egy scss részből. Models, ami tartalmazza a modelleket és a service mappa. Lebontottságának köszönhetően, a struktúra könnyen átlátható és hatékonyan tudtam fejleszteni. A front-end illetve a back-end összeköttetése HTTP requestek által lett megvalósítva. A felhasználó szeretne valamilyen információt, a UI-on kattint, onnan egy requestet küld a szervernek, az eléri az adatbázist, megkapja a kért információt, majd visszaküldi a UI-nak, és megjelenik a felhasználó előtt a várt információ.

#### **6.1.1. Szerver oldallal való kommunikáció**

A kliens oldalon a szerverrel való kommunikáció aszinkron HTTP kérések segítségével történik. A kliens oldalon HTTP kéréseket küldtem a szerver felé. Ehhez a *Service* osztályokba a *HttpClient* függvényt használtam a paraméterek beállításával, mivel a

kérésekhez általában szükség van bizonyos paraméterek beállítására, például ha egy felhasználó adatait név alapján szeretné lekérni. Miután a szerver válaszolt a kliens oldali kérésre, az adatot megjelenítettem vagy feldolgoztam. Ha a szerver hibát küldött vissza, kezelnem kellett ezeket a hibákat hibaüzenetek formájába. Tekintve, hogy szükséges a bejelentkezés az oldal használatához, az alkalmazásnak szüksége van autentikációra vagy hitelesítésre a szerverrel való kommunikáció során, a kliensnek gondoskodnia kell a szükséges hitelesítési adatok küldéséről a szervernek. Ez nálam tokennel és bejelentkezási adatokkal történik.

Példa a service osztályban történő http hívásra a 6.1

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpHeaders} from "@angular/common/http";
import {Event} from "../_models/event"

const baseUrl = 'http://localhost:8080/api/events';
const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type': 'application/json',
    'Access-Control-Allow-Origin': '*'
  })
};

@Injectable({
  providedIn: 'root'
})
export class EventService {

  constructor(private http: HttpClient) { }

  getById(id: number) {
    return this.http.get(`${baseUrl}/${id}`)
  }

  saveEvent(event: Event) {
    return this.http.post(baseUrl, event, httpOptions)
  }
  ...
}
```

**6.1. kódrészlet.** Http hívás kliensből

### 6.1.2. Service hívások eredményének feldolgozása

Ahogy a fenti pontba említem, mikor a kliens oldalon szolgáltatásokat(service)( "service" szóval hivatkoznak azokra a komponensekre vagy osztályokra, amelyek a kliens oldalon felelősek a szerverrel vagy más adatforrásokkal történő kommunikációért és adatmanipulációért.) hívunk, aszinkron módon történnek, tehát a válaszokat későbbi időpontban kapjuk meg. A szolgáltatások observables vagy promise-ok formájában

adják vissza az eredményeket. Ezeknek a feldolgozásához "fel kell iratkozzunk" a `subscribe` metódussal az `Observablere` vagy meg kell várni a `Promiset`. Amikor egy `Observable` objektumra meghívom a `subscribe()` metódust, akkor feliratkozok az objektum adatáramlására. A metódus callback függvényeket vár és ezeket a `next`, `error` nevű paraméterekkel adom meg. Az itt visszakapott értéket kimentjük, a `get` paramétere, és megjelenítünk egy sikeres üzenetet amennyiben a várt eredményt kaptuk, ellenkező esetben hibaüzenetet. Vagy feldolgozzuk az adatot.

### 6.1.3. Angular routing

Az Angular Routing segítségével lehetőség van a különböző oldalak közötti navigációra.

Az Angular Routing segítségével definiálhatunk útvonalakat (routes), amelyek meghatározzák, hogy melyik komponens jelenjen meg az adott URL-en vagy útvonalon.

Az útvonalakhoz további paramétereket is adhatunk, amelyek alapján a komponensek dinamikusan módosíthatják a megjelenített tartalmat.

Az fő komponens az "app", az itt található elemek töltődnek be legelőször. A `router-outlet` elem felelős a komponensek dinamikus megjelenítéséről az alkalmazásban.

Amikor az Angular alkalmazás navigál egy adott útvonalra, a `router-outlet` helyén jelenik meg a megfelelő komponens tartalma.

Az `app-routing-module` pedig egy modul, amelyben definiáljuk a szoftver útvonalait (routes). Létrehozunk útvonalakat az alkalmazás különböző komponenseihez és segítségével meghatározzuk, hogy melyik útvonalhoz melyik komponens tartozik, és itt állíthassuk be a guardokat.

Az alkalmazás routes tömbje a 6.2 kódblokkban figyelhető meg.

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'event-list', component: EventListComponent },
  { path: 'event-edit/:id', component: EventEditComponent, canDeactivate:
    [DirtyCheckGuard] },
  { path: 'event-new/:id', component: EventNewComponent, canDeactivate:
    [DirtyCheckGuard] },
  { path: 'event-details/:id', component: EventDetailsComponent },
  { path: 'registration/:id', component: RegistrationComponent },
  { path: 'registrationlist/:id', component: RegistrationListComponent },
  { path: 'changePassword', component: ResetPasswordComponent },
  { path: 'statistics', component: StatisticsComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' }
];
```

### 6.2. kódrészlet. Routes tömb

## **6.2. Szerver oldali megvalósítás**

A szerver oldal a Spring Boot framework segítségével, amely lehetővé teszi a hatékony kommunikációt a kliensek számára a szerveroldali alkalmazással.

A 3. fejezetben bemutatott technológiákat tartalmazza.

## **6.3. Az alkalmazás működése**

A fejezet a weboldal oldalait vizuálisan fogja bemutatni, magyarázattal együtt.

### **6.3.1. Bejelentkezés**

(??) A legelső oldal ami az alkalmazás megnyitása után fogad az a bejelentkező oldal.A jobb felső sarokba lehetőségünk van megváltoztatni az oldal nyelvét.A helyes felhasználó név és jelszó kombináció beírásával beléphetünk az oldalra, persze miután igazoltuk, hogy nem vagyunk robotok. Amennyiben nem rendelkezünk érvényes felhasználóval lehetőségünk van regisztrálni.

### **6.3.2. Regisztráció**

(??) Ahhoz hogy felhasználók legyünk a MuresTrackeren kötelezőszerűen meg kell adjuk a nevünket, telefonszámunkat, email címünket és a szerepkör listából csak a szimpla usernek megfelelő városlakót tudjuk kiválasztani. Fontos hogy az email címünket helyesen adjuk meg, mert a nevünkön kigenerált felhasználó nevet és ideiglenes jelszót emailbe fogjuk megkapni.(Amit később megtudunk változtatni)

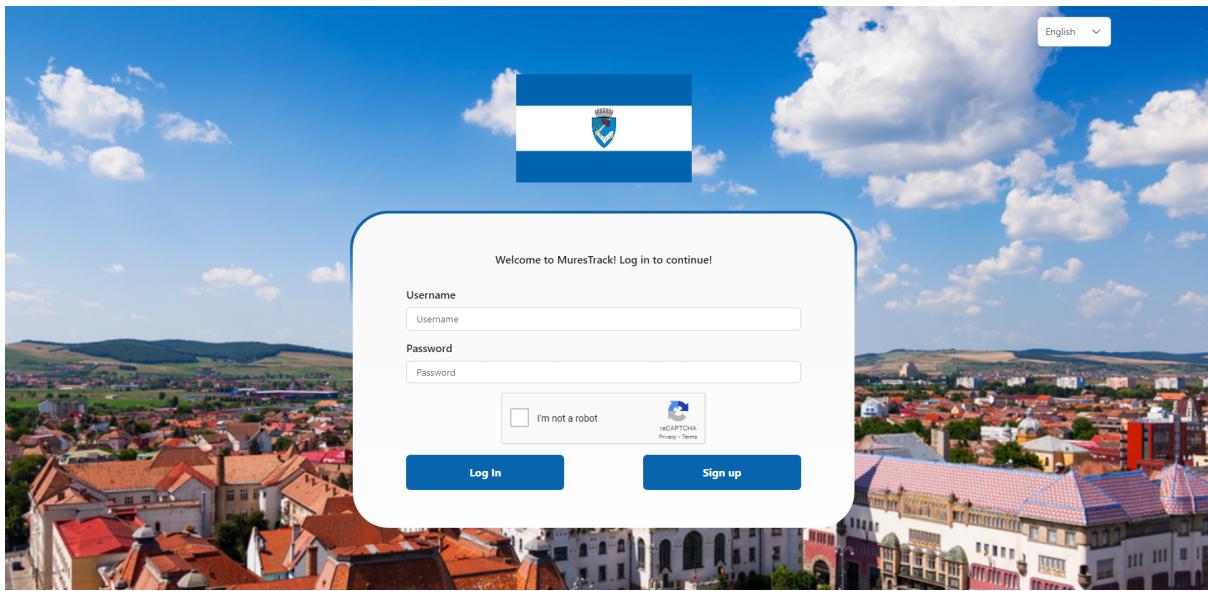
### **6.3.3. Kezdőlap**

(??) Sikeres bejelentkezés után, a kezdőlapon megjelenik a bejelentkezett felhasználó neve és szerepkörének megfelelően a menüsávban az elemek.

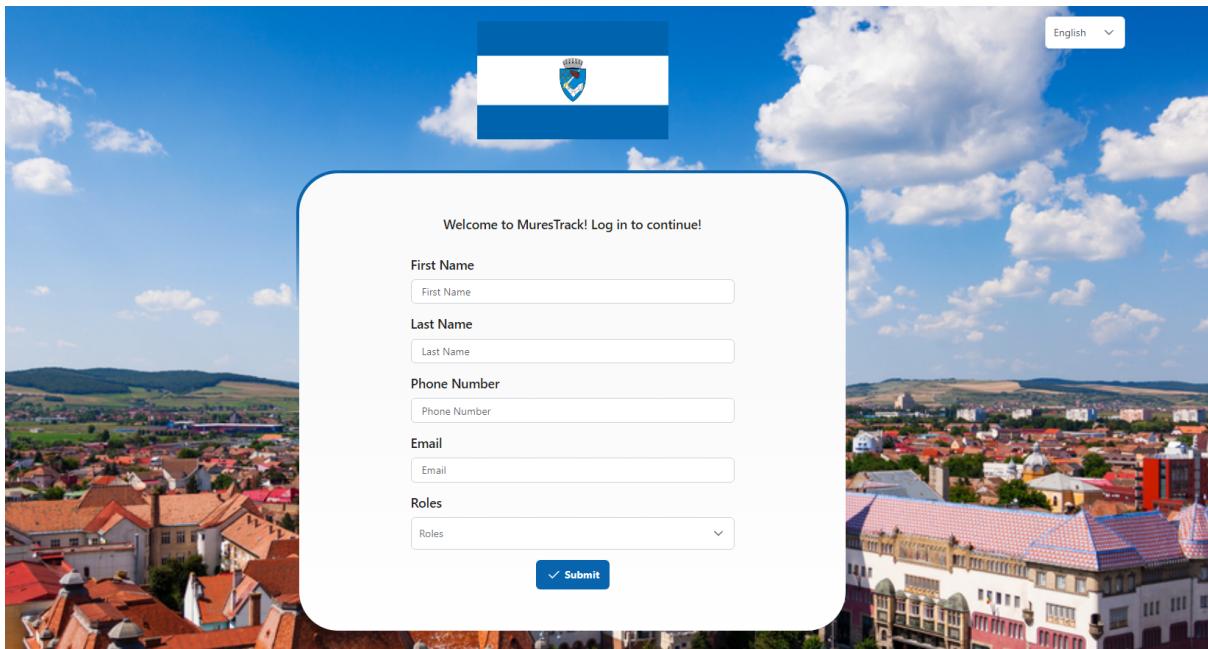
Az oldal bal oldalán helyezkedik el a menüsáv, ami az alkalmazás bármely pontján megtalálható és a három vonalkára kattintva lehetőségünk van eltüntetni a menüsávot, ha valamit teljes képernyőn szeretnénk megtekinteni. Az említett három vonalka után az alkalmazás neve látható és a jobb oldalon megint lehetőségünk van nyelvet változtatni, ha elmulasztottuk a bejelentkezésnél. Mellette található az *Értesítések* ikonja és mellette a *Kijelentkezés* ikon van.

### **6.3.4. Értesítések**

Rákattintva a levél ikonra egy oldalsó rezpontív menü nyilik ki az értesítéseinkkel. Ezeket akkor kapjuk ha módosítottunk egy bejelentést vagy valamilyen személyes adatot. Abba a pillanatba amikor kapjuk egy toast formájába jelenik meg, de ez pár másodperc után eltűnik és rákattintva a levére visszatudjuk olvasni. Amennyiben olvasatlan értesítéseink vannak egy piros pont jelzi ezt nekünk.



6.1. ábra. Bejelentkezés



6.2. ábra. Regisztráció

≡ MuresTracker

HOME

PAGES

STATISTICS

Welcome to MuresTrack, Mark Siko!

You are logged as admin

Reports 12

Users 18

Notifications 47

Permissions 5

MuresTracker © 2023

6.3. ábra. Dashboard

Username ↑↓	First Name ↑↓	Last Name ↑↓	Email ↑↓	Phone Number ↑↓	User Status ↑↓	Action
admin	Siko	Mark	admin@gmail.com	+40741923877	ACTIVE	
adamet	Nagy	Kalman	anna@gmail.com	+40741923876	ACTIVE	
localc	Consiliul	Local	tgmures@gmail.com	0265 268 330	ACTIVE	
muresp	Primaria	Mures	primaria@mures.com	0265 268 330	ACTIVE	
drumua	Administratia	Drumurilor	drum@pod.com	0265 217 241	ACTIVE	
tgmurp	Politia	TgMures	politia@mures.com	0265 202 300	ACTIVE	
karolk	Kovacs	Karoly	karoly@gmail.com	0741923871	ACTIVE	
levens	Siko	Levente	sikolev@gmail.com	074192568	ACTIVE	
petern	Nagy	Peter	nagypeti@yahoo.com	074197621	INACTIVE	
anitaf	Ferenczi	Anita	anita@gmail.com	074287612	ACTIVE	

1 of 2 << < 1 2 > >> 10 ▾

MuresTracker © 2023

6.4. ábra. Felhasználók listája

Update User

First Name  
Consiliul

Last Name  
Local

Phone Number  
0265 268 330

Email  
tgmures@gmail.com

Password

Roles

User status  
 Active

✓ Submit

6.5. ábra. Felhasználók szerkesztése

### **6.3.5. Felhasználók listája, szerkesztése és hozzáadása**

Ezt az oldalt csak azok tudják elérni akiknek van USER MANAGEMENT joguk. Egy részponzív táblázatba láthassuk az összes felhasználót és szervet. Lehetőségünk van egyedi paginátort rakni(hány felhasználót lássunk per oldal), filtrozni Abc sorrendbe és keresni a felhasználó entitás minden oszlopja alapján. Adminként tudunk új felhasználókat hozzáadni ami ugyanúgy működik mint a regisztráció. Az akció oszlopba az ikonra kattintva a felhasználó adatait tudjuk szerkeszteni és akár inaktiv taggá tehessük őket, ezáltal kizárvva őket a weboldalról.

### **6.3.6. Bejelentések listája, szerkesztése és hozzáadása**

Ez az oldal az alkalmazás legfontosabb oldala. Felépitése ugyanaz mint a Felhasználók listájának. Táblázatba lássuk az összes bejelentést. Preferencia szerint modósíthassuk, hogy hány bejelentést lássunk oldalonként. Filterezni tudunk és keresni. Ez mind az öt felhasználó esetén megjelenik, de az akció oszlopba viszont a három ikont Szerkesztés-részletek megtekintése, bejelentés bezárása és pdf-xls kigenerálását csak azok lássák akiknek megvannak a megfelelő jogaiak.

A bejelentések részleteit egy külön oldalon tekinthetjük meg vagy szerkesztés esetén (?? ábra), ugyanez az oldal van akkor is, amikor új bejelentést szeretnénk bejelenteni, csak akkor üres az űrlap. Ami nincs megjelenítve a listába az a Leírás, a hozzácsatolt kép és a térkép.

### **6.3.7. Szerepkörök**

Ezt az oldalt csak az adminok tudják elérni. A legördülő mezőből kitudjuk választani az öt szerepkört. Egy szerepkört kiválasztva a legördülő mezőből megjelennek azok a jogok a listába amik ahhoz a szerepkörhöz tartoznak. Ezzel jogokat megtudjuk vonni vagy hozzátudjuk adni különböző szerepkörhöz.

### **6.3.8. Statisztikák**

A statisztikák menüpontra kattintva megjelennek az elemzők által elkészített statisztikák. Jelenleg négy különböző diagram van az oldalon. Az első az aktív és inaktív felhasználók arányát mutassa be és a többi három a jelentéseket ábrázolja státusz, kategória és súlyosság alapján.

## **6.4. Segédeszközök**

Fejlesztői környezetet illetően, szerver oldalhoz az **IntelliJ IDEA-t** [18] használtam. Nagy mértékben elősegítette a gyors és hatékony kódolást. minden megtalálható benne ami csak kell: kódkiegészítés, formatálás, a git-tel való könnyed kommunikáció és különböző pluginok.

A megírt REST API teszteléséhez a **Postman** [19]-t használtam.

Title	Category	Location	Target Date	Severity	Created by	Status	Assigned to	Action
Broken street light	PUBLIC FACILITY	Strada Rodnei 1	28.08.2022	HIGH	admin	CLOSED	drumua	
Gas leaks	ROAD_ISSUES	Strada Godeanu 33	06.03.2023	HIGH	admin	CLOSED	drumua	
Illegal Construction	ENVIRONMENTAL_ISSUES	Calea Sighisoarei	05.03.2023	MEDIUM	admin	INFO_NEEDED	tgmurp	
Damaged playground	PUBLIC FACILITY	Strada Muncii 12	05.03.2023	MEDIUM	admin	FIXED	muresp	
Big Pothole	UTILITY_ISSUES	Bulevardul Pandurilor	07.03.2023	HIGH	admin	CLOSED	drumua	
Water main breaks	BUILDING_ISSUES	Strada Banet 21	13.03.2023	CRITICAL	admin	IN_PROGRESS	drumua	
Garbage left on the street	BUILDING_ISSUES	Strada Bolyai 11	05.03.2023	HIGH	admin	NEW	localc	
Power cuts	ENVIRONMENTAL_ISSUES	Strada Transilvaniei 1	05.03.2023	HIGH	admin	FIXED	muresp	
Illegal dumping of waste	PUBLIC FACILITY	Bulevardul Pandurilor nr 41	07.03.2023	MEDIUM	admin	FIXED	muresp	
Water on the road	ENVIRONMENTAL_ISSUES	Strada Bolyai 14	26.06.2023	CRITICAL	admin	NEW	drumua	

6.6. ábra. Bejelentések listája

**Edit Report**

**Title**  
Broken street light

**Category**  
PUBLIC FACILITY

**Location**  
Strada Rodnei 1

**Description**  
On street Rodnei the street light doesn't work! This may cause accident, please fix it ASAP!!!!  
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s.

**Target Date**  
08/28/2022

**Status**  
CLOSED

**Severity**  
HIGH

**Created by**  
admin

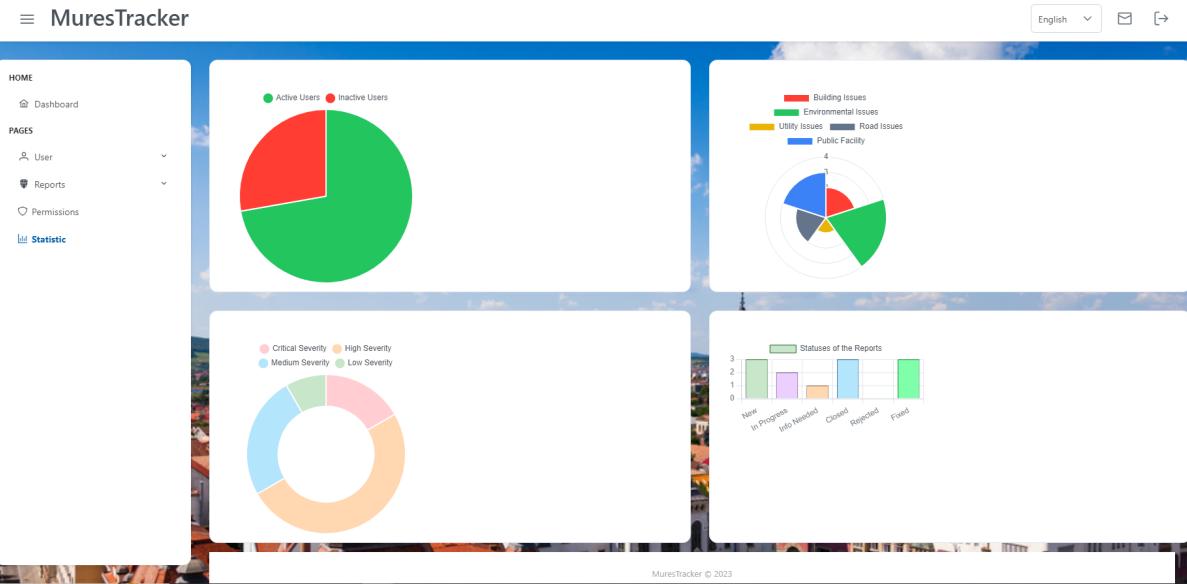
**Assigned to**  
drumua

**+Choose**

**Map** **Satellite**

The map shows a street view of Târgu Mureş with several landmarks labeled: Grand Hotel, Mureş Mall, Pensiunea Europa, Târgu Mureş, LIBERTĂȚII, and BUDAI NAGY ANTAL. Roads are numbered 15, 13, and 10.

6.7. ábra. Bejelentések szerkesztése



6.8. ábra. Statisztika

Title	Illegal Construction
Description	<p>Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. Lorem Ipsum is simply dummy text of the printing and typesetting industry.</p> <p>Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.</p>
Category	ENVIRONMENTAL_ISSUES
Location	Calea Sighisoarei
Target date	2023-03-05 00:00:00.0
Severity	MEDIUM
Status	INFO_NEEDED
Creator	admin
Assignee	tgmurp

6.9. ábra. Jelentésről kigenerált PDF

# Összefoglaló

Következtetésképpen kijelenthető, hogy a MuresTracker egy hasznos eszköz lenne a városlakók és a városi hatóságok számára egyaránt. Lehetővé teszi a gyors és hatékony kommunikációt a problémákról és a városi fejlesztésekkel kapcsolatban, valamint a nyomon követését és az eredmények dokumentálását.

Visszatekintve a célkitűzésre sikeresült elérni a főbb célokat. Bár az elején sokkal komplexebb és több funkciót képzeltem el az alkalmazásomnak, de nem gondoltam volna, hogy a kis részlet feladatak mennyire eltudnak húzódni. Fejlesztésem során szem előtt tartottam a bővíthetőséget és amit nem sikerült most leimplementálnom, az minden továbbfejlesztési lehetőséggé váll. Mind kliens, mind pedig szerver oldalon a lehető legmodernebb és legújabb technológiákat használtam fel, így könnyen továbbfejleszthető és bővíthető.

Elmondható, hogy az webalkalmazás megtervezése és folyamata nem volt egyszerű feladat, rengeteg akadálya ütköztem de végül sikeresült átlépni ezeket és sikeresült egy működőképes és modern weboldalt fejleszteni, ami hozzájárulhat a városi környezet javításához és a lakók életminőségének növeléséhez.

Továbbfejlesztési lehetőség közül megemlíteném:

- szerver üzemeltetés, hogy ne csak lokálisan fusson és valósághűen kezdjék el használni a városokba
- skálázhatóság, hogy több városba is legyen elérhető
- komment szekció a bejelentett problémák alatt
- fórum, chatelés lehetősége
- Android és iOS kliensalkalmazások készítés,
- több statisztika
- feedback lehetőség

GitHub repository linkjei:

<https://github.com/Mark-i7/murestrack-BE>

<https://github.com/Mark-i7/murestrack-FE>

# Ábrák jegyzéke

3.1. Felhasználó vezérlő függőségei . . . . .	15
3.2. Bejelentések vezérlő függőségei . . . . .	15
3.3. A Presentation-Business-Persistence-Database megközelítés . . . . .	22
3.4. A Repository-Service-Controller rétegstruktúra megközelítés . . . . .	23
4.1. Felügyelői szerepkör jogai . . . . .	31
4.2. Városi hivatalnok szerepkör jog . . . . .	31
4.3. Elemzői szerepkör jogai . . . . .	31
4.4. A város lakosai szerepkör jog . . . . .	31
4.5. Használati eset diagram . . . . .	32
4.6. Jelentés hozzáadásának szekvenciadiagramja . . . . .	33
4.7. Egy jelentés lezárása szekvenciadiagramja . . . . .	34
4.8. Aktivitás diagram . . . . .	36
5.1. Rendszer architektúrája . . . . .	39
5.2. Adatbázis táblák szerkezete . . . . .	40
6.1. Bejelentkezés . . . . .	45
6.2. Regisztráció . . . . .	45
6.3. Dashboard . . . . .	45
6.4. Felhasználók listája . . . . .	46
6.5. Felhasználók szerkesztése . . . . .	46
6.6. Bejelentések listája . . . . .	48
6.7. Bejelentések szerkesztése . . . . .	48
6.8. Statisztika . . . . .	49
6.9. Jelentésről kigenerált PDF . . . . .	49

# Irodalomjegyzék

- [1] „Mi is az a keretrendszer?” <https://prog.hu/szotar/keretrendszer>. (elérés dátuma: 2023. jún. 25.).
- [2] „Java keretrendszerek.” <http://faragocsaba.wikidot.com/java-frameworks>. (elérés dátuma: 2023. jún. 25.).
- [3] „Inversion of control a springbe.” <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>. (elérés dátuma: 2023. jún. 25.).
- [4] „Bean lényege.” <https://medium.com/javarevisited/spring-beans-in-depth-a6d8b31db8a1>. (elérés dátuma: 2023. jún. 25.).
- [5] „Hogyan működik a rest api ?” <https://testerlab.io/blog/hogyan-mÁskÁudik-egy-rest-api/>. (elérés dátuma: 2023. jún. 25.).
- [6] „Rest architekturája.” <https://www.geeksforgeeks.org/rest-api-architectural-constraints/?ref=lbp>. (elérés dátuma: 2023. jún. 25.).
- [7] „Spring és spring boot összehasonlítása.” <https://www.baeldung.com/spring-vs-spring-boot>. (elérés dátuma: 2023. jún. 25.).
- [8] „Project lombok alapjai.” <https://www.baeldung.com/intro-to-project-lombok>. (elérés dátuma: 2023. jún. 25.).
- [9] „Spring boot architektúrája.” <https://www.javatpoint.com/spring-boot-architecture>. (elérés dátuma: 2023. jún. 25.).
- [10] „Spring hivatalos dokumentációja.” <https://spring.io/projects/spring-data-jpa>. (elérés dátuma: 2023. jún. 25.).
- [11] „Bevezetés a spring securitybe.” <https://spring.io/projects/spring-security>. (elérés dátuma: 2023. jún. 25.).
- [12] „Typescript bevezetés.” <http://nyelvek.inf.elte.hu/leirasok/TypeScript/index.php?chapter=1>. (elérés dátuma: 2023. jún. 25.).
- [13] „Angular routing.” <https://www.smashingmagazine.com/2018/11/a-complete-guide-to-routing-in-angular/>. (elérés dátuma: 2023. jún. 27.).

- [14] „Guardok, canactivate.” <https://www.telerik.com/blogs/angular-basics-canactivate-introduction-routing-guards>. (elérés dátuma: 2023. jún. 27.).
- [15] „Primeng hivatalos dokumentációja.” <https://www.primefaces.org/primeng/>. (elérés dátuma: 2023. jún. 27.).
- [16] „Weboldal elemzése CEO szemmel.” <https://www.logicalminddesign.hu/seo-keresooptimalizalas/weboldal-elemzes>. (elérés dátuma: 2023. jún. 25.).
- [17] „Trello hivatalos dokumentációja.” <https://trello.com/hu/guide/trello-101>. (elérés dátuma: 2023. jún. 27.).
- [18] „Jetbrains hivatalos dokumentációja.” <https://www.jetbrains.com/help/idea/getting-started.html>. (elérés dátuma: 2023. jún. 27.).
- [19] „Postman hivatalos dokumentációja.” <https://learning.postman.com/docs/introduction/overview/>. (elérés dátuma: 2023. jún. 27.).