

**Szegedi Tudományegyetem**  
**Informatikai Intézet**

# **SZAKDOLGOZAT**

**Kazár György**

**2022**

**Szegedi Tudományegyetem  
Informatikai Intézet**

**Alvási minőség elemző rendszer**

**Szakdolgozat**

Készítette:

**Kazár György**

gazdaságinformatika BSc  
szakos hallgató

Témavezető:

**Horváth Ferenc**

Szeged  
2022

## ***Feladatkiírás***

A feladat egy olyan webes alkalmazás elkészítése volt, ahol a felhasználók által közölt alvási adatok alapján (pulzusszám és véroxigén szint, felhasználó által értékelt napi alvásminőség) az alkalmazás elemzéseket végez és kimutatásokat készít, és ezt a felhasználó és a kezelőorvosa megtekinthetik. Abnormális pulzusszám vagy véroxigén szint esetén a rendszer értesítést küld a páciensnek és kezelőorvosának.

## **Tartalmi összefoglaló**

- **A téma megnevezése:**

Alvási minőség elemző rendszer

- **A megadott feladat megfogalmazása:**

A feladat egy webes alkalmazás készítése, amely kimutatásokat készít a felhasználó alvási adataiból, a felhasználó pulzusszáma és véroxigén százalék adataiból. Ezen elemzéseket a felhasználó és orvosa is megtekinthetik, a felhasználó értékelheti alvását egy egytől ötig terjedő skálán, és abnormális értékek esetén a rendszer értesítést küld a felhasználónak és kezelő orvosának is.

- **A megoldási mód:**

A szerver oldali adatfeldolgozásra Java Spring, a kliens oldali megvalósításhoz Angular keretrendszert alkalmaztam. Az adatok tárolásához PostgreSQL relációs adatbázist választottam.

- **Alkalmazott eszközök, módszerek:**

A fejlesztéshez szükséges keret rendszereken kívül (Java Spring, Angular), a fejlesztési folyamatok áttekinthetősége érdekében Trello issue trackert, és a forráskód verziókövetésére gitet használtam.

- **Elért eredmények:**

A fejlesztés végeredményeként egy olyan webalkalmazás készült ahol a felhasználók az alvási adataik elemzését megoszthatják kezelőorvosával

- **Kulcsszavak:**

Alvási minőség, Webalkalmazás, Angular, Java Spring, pulzus, véroxigén

## ***Tartalomjegyzék***

<b>Feladatkiírás</b>	<b>3</b>
<b>Tartalmi összefoglaló</b>	<b>4</b>
<b>Bevezetés</b>	<b>6</b>
<b>1. Az alkalmazás elkészítésének célja</b>	<b>6</b>
<b>2. Az alkalmazás bemutatása</b>	<b>7</b>
2.1. Regisztráció	7
2.1.1. Orvos regisztrálása	7
2.1.2. Páciens regisztrálása	8
2.3. Jelszó módosítása	10
2.4. Alvási adatok vizualizálása	11
2.4.1. Adatok megjelenítése az orvos szemszögéből	12
2.5. Adatmodell	15
2.5.1. Relációs adatbázi séma	15
<b>3. Fejlesztői eszközök</b>	<b>16</b>
3.1. Java Spring	16
3.2. Spring Data JPA	19
3.3. Lombok	20
3.5. Angular	22
3.5.1. Komponens	22
3.5.2. Lifecycle Hooks	23
3.5.3. Modul	23
3.6. Angular CLI	24
3.7. PrimeNg	25
<b>4. Fejlesztési folyamatot támogató alkalmazások</b>	<b>25</b>
4.1. Trello	25
4.2. GitHub:	26
<b>Irodalomjegyzék</b>	<b>29</b>
<b>Nyilatkozat</b>	<b>30</b>

## Bevezetés

### 1. Az alkalmazás elkészítésének célja

Napjainkban egyre népszerűbbek a különböző aktivitásmérő, okos kiegészítők, melyek másodpercenként akár több tucat adatot is képes gyűjteni a tulajdonosairól. Szinte minden modern okosórában, aktivitás mérő karkötőben megtalálható a pulzus, illetve véroxigénszint mérő funkcionalitás, de ezen adatok felhasználási területei korlátozott, illetve egy átlagos felhasználó számára az adatokat nem tekinthetők értékesnek.

A leggyakoribb, ezekből az adatokból kinyerhető összefüggések a sportolási tevékenységgel, teljesítménnyel kapcsolatos, esetleg szélsőséges esetekben egészségügyi figyelmeztetésre szolgálhatnak. Úgy gondolom, hogy az így nyert adatok felhasználása számtalan lehetőséget tartogat magában. Szakdolgozatom témájául egy lehetséges adat-felhasználási módot választottam.

Egy általános modernkori népbetegség az alvászavar. A legtöbb embernek hét-nyolc óra alvásra van szüksége átlagosan naponta, ám nem csak az alvással töltött idő befolyásoló tényező, hanem az alvás minősége is nagyban meghatározza az egyén kipihentségét, koncentrációs képességét.

A rossz minőségű, vagy kevés alvás számtalan kockázatot rejt magában, például hozzájárulhat a szív -és érrendszeri betegségek, cukorbetegség kialakulásához így az alvászavar kivizsgálása egy nagyon fontos folyamat.

Egy általános vizsgálat része lehet a kardiorespiratorikus poligráfia is, ahol különböző eszközökkel a mellkasi és hasi légzőmozgáson és szívritmus adatokon kívül a hangjelenségeket, a vér oxigénszintjét, szívfrekvenciát (pulzust) mérnek. Az utóbbi adatok mindegyikére képesek az említett okoseszközök.

Ilyen vizsgálatokat különböző alváslaboratóriumokban végeznek, távol otthonunktól így ezek az adatok bizonyos mértékben eltérhetnek a valóságtól a környezetváltozás miatt.

Szakdolgozatomban egy lehetséges megoldást próbálok bemutatni az utóbb említett problémára.

Egy olyan webalkalmazás fejlesztését választottam témául, ami az okosórából nyert alvás közbeni pulzus, illetve vér-oxigénszint adatok szinkronizálására szolgál, és összefüggéseket állíthat ezekből, amit az orvos és páciense egyaránt láthat, így segítve az alvászavar diagnosztikáját.

Szakdolgozatom során fiktív adatokkal dolgoztam, és orvosilag nem alátámasztott összefüggéseket mutat be.

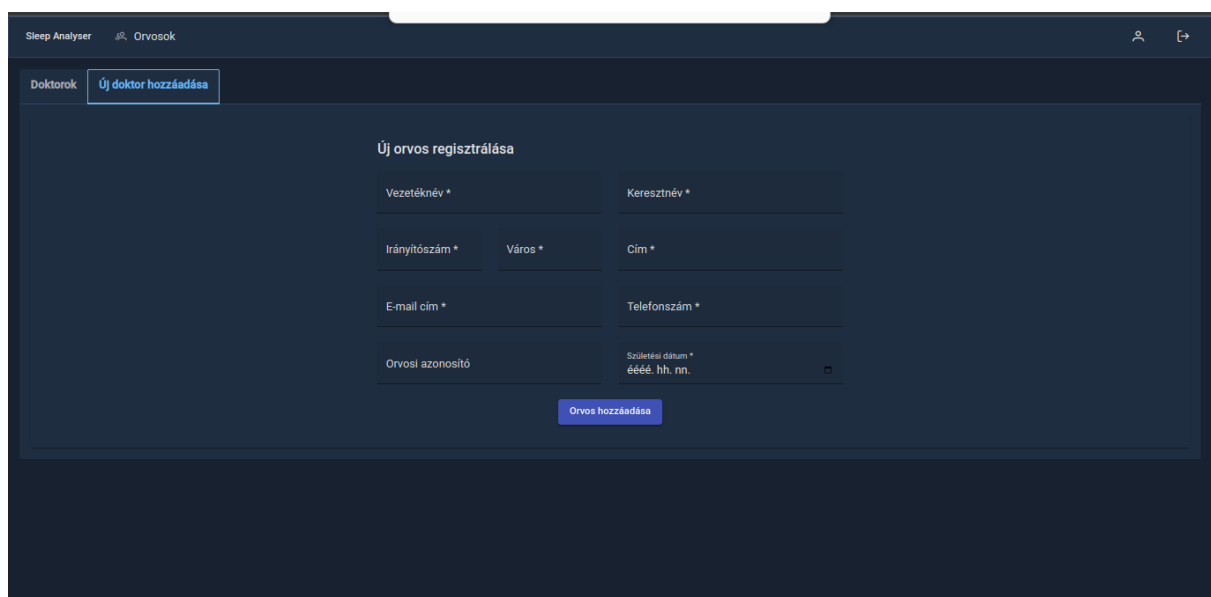
## 2. Az alkalmazás bemutatása

### 2.1. Regisztráció

A rendszer nem rendelkezik külső regisztrációs felülettel.

Új felhasználót csak adminisztrátor vagy orvos tud regisztrálni, a rendszer első indításakor egy adminisztrátori jogkörrel rendelkező felhasználó kerül regisztrálásra, viszont az adminisztrátor csak orvosokat tud hozzáadni. A páciensek regisztrálását az orvosok végzik.

#### 2.1.1. Orvos regisztrálása



2.1.1 ábra: Új orvos regisztrációs felülete.

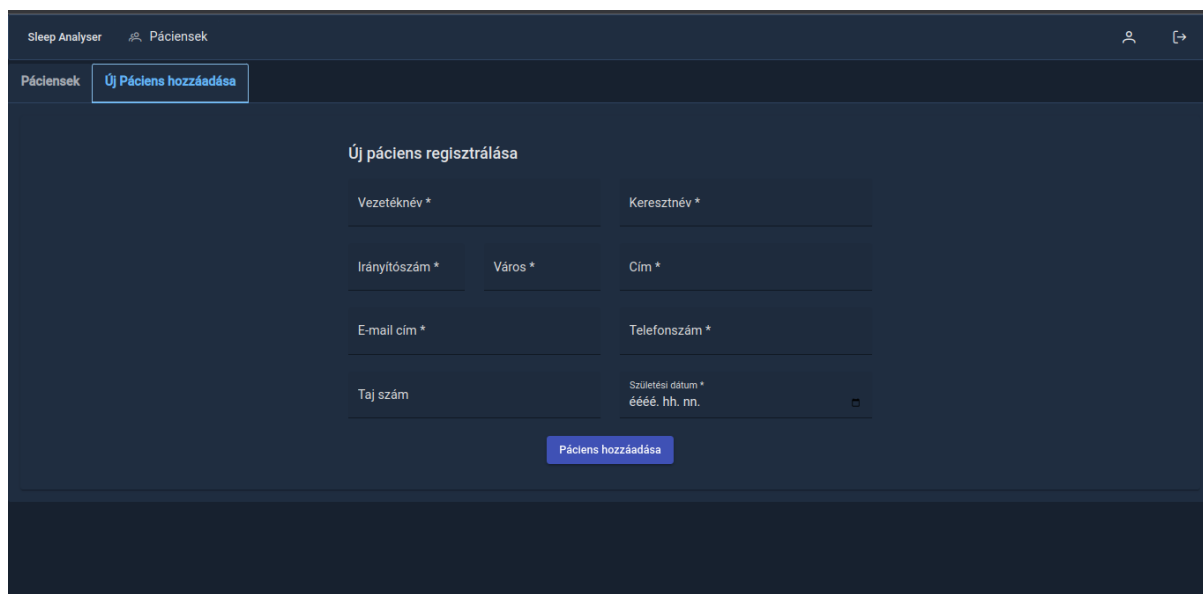
Az orvosokat csak az adminisztrátori jogkörrel rendelkező felhasználó tudja felvenni. A regisztráció során meg kell adni a felvenni kívánt orvos nevét, lakcímét, e-mail címét, telefonszámát, orvosi azonosítóját és születési dátumát.

Az összes mező kitöltése kötelező, ezen felül az email nezőnek tartalmaznia kell “@” és “.” karaktereket és a telefonszámnak is egyedi feltételnek kell megfelelnie. Ezek még a kliens oldalon ellenőrzésre kerülnek és az űrlap mentése addig nem engedélyezett amíg az összes mező értéke nem felel meg a validációnak, ha mégis menteni szeretné a felhasználó egy felugró értesítés is figyelmezteti az űrlap hiányosságáról.

Az orvosi azonosító számának egyedinek kell lennie, ha esetleg a rendszerben szerepel ilyen pecsét számmal már orvos a rendszer elutasítja a kérést és jelzi az adminisztrátornak, hogy egyediségre vonatkozó feltételt sért a kérés.

A regisztrációs folyamat elküldésével a felhasználó mentésre kerül és egy értesítő e-mailt is küld a rendszer a megadott email címre, amely tartalmazza a belépéshez szükséges felhasználónevet és a rendszer által generált jelszót.

### 2.1.2 Páciens regisztrálása

The screenshot shows a web application interface for patient registration. At the top, there's a header with 'Sleep Analyser' and 'Páciensek'. Below this, a navigation bar includes 'Páciensek' and a button 'Új Páciens hozzáadása'. The main content area is titled 'Új páciens regisztrálása' and contains a form with the following fields: 'Vezetéknév \*', 'Keresztnév \*', 'Írányítószám \*', 'Város \*', 'Cím \*', 'E-mail cím \*', 'Telefonszám \*', 'Taj szám', and 'Születési dátum \*' (with a date picker). A blue button labeled 'Páciens hozzáadása' is positioned at the bottom of the form.

2.1.2. ábra: Páciens regisztrációs felület

A Páciens regisztrációs űrlapnak is hasonló megköötéseknek kell megfelelnie, mint az orvos regisztrálásakor, kiegészítve a TAJ kártya mezővel, amely hossza kilenc karakter és hármásával tagolt. Ezt maszk használatával jelzi a mező a felhasználónak.

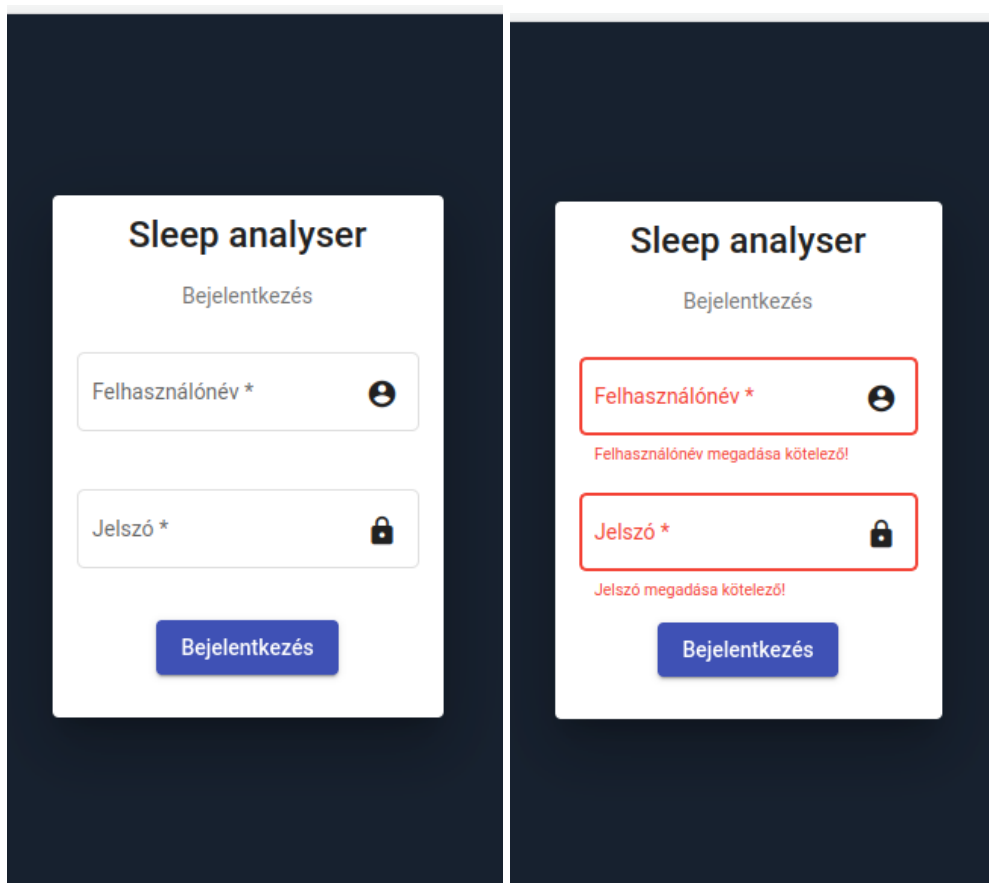
Páciens a rendszerbe csak orvos regisztrálhat, hiszen a regisztráció során történik meg az Orvos - Páciens kapcsolat mentése is. Ebben az esetben a regisztrációs



folyamatot végrehajtó orvos azonosítója lesz a pácienshez kapcsolva, így biztosítva azt, hogy csak a beteget ellenőrző orvos férjen a beteg alvási adataihoz. A Páciens regisztrálásakor, ugyanazokat az adatokat kell megadni mint az orvosoknál, kivétel az orvosi pecsét száma, ezt felváltja a TAJ kártya szám, amire szintén egyediséget érintő megszorítás vonatkozik. A Páciens szintén értesítést kap a megadott email címére, amely tartalmazza a belépéshez szükséges adatokat.

A regisztrációt követően az első belépésnél az orvosoknál és a pácienseknél is egy felugró ablak jelenik meg amely figyelmezteti őket arra, hogy a belépés után változtassák meg jelszavukat, és a jelszó változtatását követően az alkalmazás aktiválni fogja a felhasználói fiókot.

## 2.2. Bejelentkezés



The image displays two versions of the 'Sleep analyser' login interface. The left version is the standard login screen with a dark blue background and a white login box. It contains the title 'Sleep analyser', the subtitle 'Bejelentkezés', and two input fields: 'Felhasználónév \*' (with a user icon) and 'Jelszó \*' (with a lock icon). A blue 'Bejelentkezés' button is at the bottom. The right version shows the same interface but with red borders around the input fields and red error messages below them: 'Felhasználónév megadása kötelező!' and 'Jelszó megadása kötelező!'.

2.2. ábra: Bejelentkező felület és a mezők kötelező kitöltését jelző üzenetek

A bejelentkező felület egy általános felhasználónév és jelszó megadásával történik. A mezőket validátorokkal láttam el amelyek jelzik a felhasználónak, hogy kötelező

azokat kitölteni.

A Felhasználók hitelesítésére és a jogosultságaik kezelésére JSON Web token-t (JWT) használtam. A JWT lényege, hogy sikeres bejelentkezés után a szerver kiállít egy JSON objektumot a felhasználó szükséges adatairól és ezt egy titkosított aláírással hitelesíti, így amikor a bejelentkezés után a felhasználó további kéréseket küld a szervernek, akkor a kérésben elég ezt a tokent átadni és ebből a szerver tudni fogja, hogy a felhasználó a rendszer felhasználója és a token érvényes-e még.

A bejelentkezés után a szerver kettő tokent állít ki egy `access_token` és egy `refresh_token`. Erre azért volt szükség, hogy a felhasználói élményt javítani tudjam, hiszen, ha csak egy token lenne, akkor a lejáratot követően, ha a felhasználó egy újabb kérést küld a szervernek, a szerver el fogja utasítani a kérést és kilépteti a felhasználót az alkalmazásból.

Erre az esetre hoztam létre a refresh token-t amely lejárat ideje hosszabb mint az `access_token` lejárat, és ha a szerver egy lejárt token választ ad vissza a kérésre, akkor az alkalmazás elküld egy token megújítási kérést amely újra létrehozza a felhasználóhoz tartozó tokeneket.

Ezt a működést az angular Http Interceptorral oldottam meg amely lényege, hogy minden kimenő kérésnél megállítja a folyamatot, a kérés fejlécéhez hozzáadja a "Authorization" kulcsot és a token értékét, majd elküldi a kérést a szervernek, és várja a választ. A válasz érkezésekor megnézi, hogy sikeresen lefutott-e a kérés vagy esetleg hibával tért vissza, ha hibával tért vissza és a hiba a token érvényességével hozható összefüggésbe, akkor egy újabb kérést indít ami megújítja a tokeneket.

## 2.3. Jelszó módosítása

2.3. ábra: Jelszó módosító felület

A felhasználót az első bejelentkezés után egy felugró ablak fogadja, amely figyelmezteti a felhasználót, hogy az automatikusan generált jelszavát célszerű cserélni és az új jelszó megadása után aktiválásra kerül a fiókja, ezért elhelyezésre került egy “Jelszó módosítása” gomb is a modalon amely átirányít a jelszó módosítása oldalra.

A jelszó módosítása oldalon meg kell adnia a felhasználónak a jelenlegi jelszavát és egy általa választott új jelszót, amelyre a következő megkötések vonatkoznak:

- A jelszónak tartalmaznia kell legalább egy kisbetűt.
- A jelszónak tartalmaznia kell legalább egy nagybetűt.
- A jelszónak tartalmaznia kell legalább egy számot
- A jelszó hossza legyen legalább 8 karakter.

Ezek a feltételek az “Új jelszó” mezőre kattintva láthatóvá válnak a felhasználó számára is. Ha az űrlap megfelel a megszorításoknak és az új jelszó és az új jelszó megerősítése mezők értéke megegyezik, akkor elérhetővé válik a jelszó módosítása gomb.

A módosítás gombra kattintva az új jelszó elküldésre kerül a szerverre és a szerver ellenőrzi, hogy a jelenlegi jelszó tényleg a felhasználóhoz tartozik, ha igen elmenti az új jelszót és választ küld a kliensnek a sikeres műveletről, ha nem akkor egy hiba választ küld vissza, a válaszokat pedig egy toast üzenetben megjeleníti az alkalmazás.

## 2.4. Alvási adatok vizualizálása

Az alvási fázisok meghatározásához, a páciens életkorát és az életkorhoz tartozó átlagos nyugalmi pulzusszámot vettem alapul. A cikkben [1] említett életkorokat és pulzusszámokat alkalmaztam, amelyek a következők:

- 50 év alatti felnőttek esetében 64 ütés / perc
- 50 - 60 év között 68 ütés / perc
- 60 - 80 év között 73 ütés / perc.

Az átlagos nyugalmi pulzusszám meghatározása után az alvási fázisok meghatározásához mozgó átlagokat alkalmaztam, amelyek egymáshoz viszonyított elmozdulása meghatározza az alvási fázisok hosszát.

Mély alvás esetén a pulzusszám csökken, tehát a rendszerem a mély alvást akkor számolja, ha a páciens pulzusszáma eléri a nyugalmi pulzusszámot és a 3 egységnyi mozgóátlag értéke az 5 egységnyi mozgóátlag alatt van.

REM alvási fázis esetén, a pulzusszám emelkedik. Ebben a fázisban álmodunk és a jó vagy rossz álmok hatására a pulzusszám emelkedésnek indul. A rendszerem azt az időszakot tekinti REM fázisnak, ahol a pulzusszám a nyugalmi pulzusszámnál kevesebb és a 3 egységnyi mozgóátlag az 5 egységnyi mozgó átlagnál nagyobb (a pulzusszám emelkedik).

Könnyed alvási fázis pedig akkor következik be, ha az aktuális pulzusszám az átlagos nyugalmi pulzusszámot meghaladja.

### 2.4.1. Adatok megjelenítése az orvos szemszögéből

Az orvos jogosultsággal rendelkező felhasználó a bejelentkezés után a hozzá tartozó páciensek listáját láthatja táblázatos megjelenítésben. A táblázatos megjelenítésben láthatja a páciens személyes adatait és a pácienshez tartozó műveleteket. Az orvos megtekintheti a páciens alvási adatait, módosíthatja az adatlapját és törölheti a páciens a listából.

Sleep Analyser							
Páciensek							
Páciensek Új Páciens hozzáadása Alvási adatok							
Vezetéknév ↑↓	Keresztnév ↑↓	Irányítószám ↑↓	Város ↑↓	Cím ↑↓	Telefonszám ↑↓	email ↑↓	Taj szám ↑↓
Kis	Géza	6725	Szeged	y utca 123	35-12/2271-788	tedah76610@kuvasin.com	111-722-822
Könyves	Kálmán	6722	Szeged	asd utca 234	22-27/8280-791	kese@mailinator.com	122-267-221

2.4.1.1. ábra: Az orvoshoz tartozó páciensek listája

A páciens törlése során az orvos egy felugró ablakot lát, amelyet elutasíthat vagy elfogadhat. A felugró ablak elfogadása esetén egy töltő ikon jelenik meg és a törlés sikerességét követően az orvos értesítést kap a folyamat sikerességéről.

Az orvosnak lehetősége van a páciens adatainak szerkesztésére, ha a regisztráció során elgépelés történt vagy a felhasználó adatai megváltoztak. A szerkesztés gombra kattintva átnavigálja a páciens adatlapjára.

A páciens alvási adatainak megjelenítése gombra kattintva az orvosnál megjelenik az elemző oldal ahol három különböző nézet közül választhat.



3.4.1.2. ábra: A páciens alvási adatainak vizualizálása napi nézetben.

Az első és alapértelmezetten ez jelenik meg, a napi nézet, amelyen külön láthatja a felhasználó az előző napi pulzusszámának és vér oxigén értékének vonal diagramjait

az alvás és a különböző alvási fázisai (könnyed, mély és REM alvás) hosszát , pulzus és véroxigén szint minimális, maximális és átlagos értékét.

Az oldalon elhelyezésre került egy nap választó gomb is, így a felhasználó váltani tud a lapok között egyesével vagy akár egy adott napot is meg tud tekinteni, ha a dátumra kattint. Ekkor egy dátum választó ablak nyílik meg ahol a felhasználó választhat a napok között.

A heti nézetre kattintva egy oszlop diagram nyílik meg, ahol már csak az alvás hossza kerül vizualizálásra. Az alvás hosszától függően eltérő színű oszlopok jelennek meg. 6 óra alvás alatt piros, 6 és 7 óra alvás esetén sárga és 7 órát meghaladó alvás esetén zöld színben.



3.4.1.3. ábra: A páciens heti alvási adatainak megjelenítése

A havi nézet hasonló a heti nézethez, itt értelemszerűen a hónap alvási adatai kerültek megjelenítésre.



3.4.1.4. ábra: Az alvási adatokból generált diagram havi nézete.

## 2.5. Adatmodell

Az adatok tárolásához PostgreSQL relációs adatbázis-kezelő alkalmazást választottam.

Az adatokat öt táblára bontottam, ezek a következők:

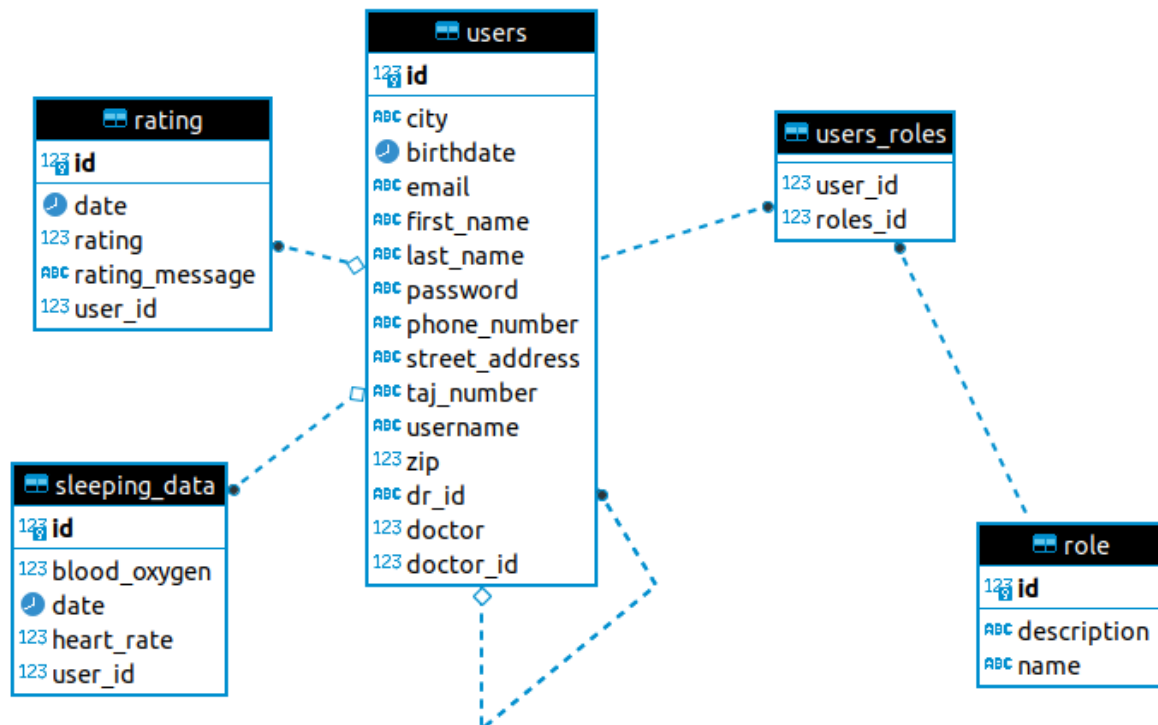
- Users - a felhasználók adatait tárolja
- role - jogosultságok tárolása
- users-roles - A felhasználókhöz kapcsolódó jogosultságokat tartalmazza.
- sleeping\_data - Az alvási adatokat tartalmazza.
- rating - A felhasználó napi alvási minőség értékeléseket tartalmazza.

### 2.5.1 Relációs adatbázis séma

Az adattáblák pontosabb specifikációja, ahol az elsődleges kulcsokat aláhúzással, az idegen kulcsokat pedig dőlt kiemeléssel jelöltem.

- Users(id, first\_name, last\_name, username, zip, city, street\_address, birthdate, email, password, phone\_number, *taj\_number*, *doctor\_id*)

- Users\_roles(*user\_id*, *role\_id*)
- Role(id, name, description)
- Rating(id, date, rating, rating\_message, *user\_id*)
- Sleeping\_data(id, blood\_oxygen, heart\_rate, date, *user\_id*)



2.5.1. ábra: Adatbázis egyed - kapcsolat diagram.

### 3. Fejlesztői eszközök

#### 3.1. Java Spring

A backend megvalósításához Java Spring keretrendszert választottam Lombok java könyvtárral kiegészítve. A Java egy platformfüggetlen, objektum orientált, erősen típusos programozási nyelv, amelyhez elérhető a spring keretrendszer.



A Java spring [1] egy nyílt forráskódú, java alapú keretrendszer, amely az IoC (Inversion of Control) tervezési mintát alkalmazza.

A spring keretrendszert több önnálló modul alkotja, amelyeket a projekt létrehozásakor vagy későbbiekben a függőségeket leíró pom.xml fájl bővítésével megadhatjuk.

A Java Spring projekt létrehozásához a start.spring.io [2] weboldalt használtam, ahol a projekt létrehozásához szükséges adatok megadását követően (projekt neve , használandó programozási nyelv, a használni kívánt spring boot verzió, java verzió) lehetőség van különböző modulok hozzáadására.

Az általam használt modulok:

- Spring web - Web alkalmazások készítésére használható, külső hozzáféréseket biztosít a kliens számára a megadott útvonalakhoz (endpoint).
- Spring security - Autentikáció és Autorizáció kezelésére szolgáló komponens.
- Spring Data JPA - Az adatkezelés megvalósítására szolgáló komponens.
- PostgreSQL Driver - adatbázis kapcsolatot megvalósító driver.

Általában a Springet backend fejlesztésre REST API-ként szokták használni, de alapvetően egy MVC (Model-View-Controller) fejlesztési mintára épül, amely képes önmagában egy futtatható alkalmazás készítésére.

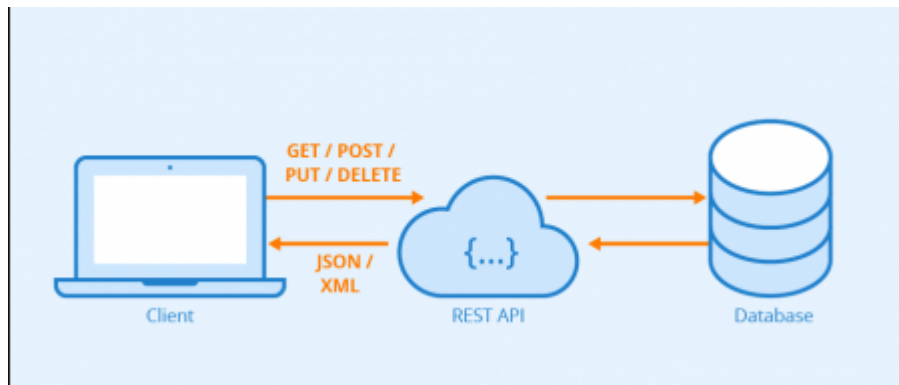
Az MVC esetén a projekt átláthatósága a cél. A felhasználói felület és az üzleti logika szétválasztására alkalmazott módszer. Három jól elkülöníthető részre bontja az alkalmazást.

Model - Az adatok tárolását és az adatkezelést valósítja meg. A model végez adatbázis műveleteket a controller által küldött adatokkal.

View - A felhasználói felület megjelenítése a fő feladata, itt kerül vizualizálásra a modell által átadott adat és a felhasználó számára elérhető műveletek, különböző user interface-ek segítségével (gombok, input mezők).

Controller - A felhasználói felületet vezérli, felelős az adatok betöltéséért és egyéb felhasználói műveletek végrehajtásáért. Például a controller felelős egy gomb megnyomását követően milyen műveletek hajtodjanak végre. Tehát a controller feladata, hogy a felhasználói műveleteket parancsokká alakítsa.

A REST API (Representational State Transfer Application Public Interface) lényege, hogy a kliens különböző kéréseit szolgálja ki, és válaszoljon a kérésekre a megfelelő adatokkal JSON vagy XML formátumban.



3.1.1. ábra: Kliens és szerver kommunikációja

forrás:

<https://www.seobility.net/en/wiki/images/thumb/f/f1/Rest-API.png/450px-Rest-API.png>

Ezt a működést a Rest kontrollerekkel tudja megvalósítani, melyek útvonalakat (endpoint) biztosítanak a klienseknek a szerverrel való kommunikációhoz.

A kéréseknek négy típusa létezik.

- A "Get" kérések adatok lekérésére használhatóak.
- A "Post" kérések új adatok mentésére, bonyolult altab, sok paraméteres lekérések megvalósítására használható.
- A "Put" kéréseket meglévő adatok módosítására használják.
- A "Delete" kérés pedig az adatok törlésére alkalmazható.

Ezekre a kérésekre a szerver válaszolhat egyszerű logikai válasszal, a mentett objektum visszaadásával de célszerű a válaszokat DTO -ként (Data Transfer Object) átadni. A DTO egy olyan objektum, melynek az egyetlen célja, hogy tárolja az adatot egy megadott formátumban, és a kliens felé továbbításra, vagy éppen a kienstől érkezve feldolgozásra kerüljön, és csak a lényeges információt tartalmazza. Például egy felhasználó lekérése esetén nem célszerű a felhasználó minden adatát a kliens oldalra elküldeni, ezért a felhasználó objektumot egy DTO-vá alakíthatjuk.

A controllerek az adat kezelését a service-eknek delegálják, itt kerül megvalósításra a tényleges üzleti logika. A service-ben végzett műveletek után az adatbázis kommunikációt a Repository osztályok végzik.

### 3.2. Spring Data JPA

A JPA (Java persistence API) egy olyan szolgáltatás amely a java osztályokat képezile relációs adattábla elemeire. Egy entitás az adott táblázat egy sorát jelenti.

A JPA biztosítja az alapvető CRUD (Create, Read, Update, Delete ) adatbázis lekéréseket, viszont ezek használatához két alapvető dologra van szükségünk.

Első lépés, hogy minden olyan osztályt, amelyet adattáblákban szeretnénk tárolni el kell látni @Entity annotációval.

```
1      package sleeperanalyser.srv.Entities;
2
3      import ...
4
5      @Entity
6      @Data
7      @AllArgsConstructor
8      @NoArgsConstructor
9      public class Role {
10
11          @Id
12          @GeneratedValue(strategy = GenerationType.AUTO)
13          private Long id;
14          private String name;
15          private String description;
16      }
```

3.2.1 ábra: @Entity annotációval ellátott osztály

Második lépésben készítenünk kell egy repository interface-t az adott osztályhoz, amelyet ki kell bővíteni a JpaRepository interface-el, amelynek meg kell adni az osztályt, amelyhez szeretnénk használni a repository-t.

```

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
    List<User> findAll();
    List<User> findByRolesName(String rolename);
}

```

### 3.2.2 ábra: Repository felépítése

Amikor egy Repository implementációt hoz létre, elemzi a megadott entitás összes metódusát, és a metódosnevekből automatikusan lekéréseket generál.

Ilyen például a legegyszerűbb lekérés a findById, amely az entitás id attribútuma alapján szűri az adattáblát vagy a findAll(), amely visszaadja az összes elemét az adattáblának. Szűréseknél a findBy kulcsszóval és ezt követően az osztály attribútumának nevének megadásával bármilyen attribútum szűrés lekérhető, "And" vagy "Or" kulcsszó hozzáadásával a szűrés feltételei bővíthetőek.

A JPA használatával lehetőségünk van entítások közötti kapcsolatokat megvalósítására is.

- @OneToOne - Egy az egyhez kapcsolat esetén egy elemhez csak egy elem tartozhat. Ilyenkor az annotációval ellátott entitás adattáblája kiegészül egy elemmel, amely a kapcsolt entitás egyedi azonosítóját tárolja.
- @OneToMany és @ManyToOne - Egy a többhöz kapcsolat megvalósítására használható. Ebben az esetben az entitáshoz létrejön egy idegen kulcs amely a megadott másik entitás elsődleges kulcsa lesz.
- @ManyToMany - Több a többhöz kapcsolatot valósít meg. Létrehozásra kerül egy kapcsoló tábla, amely tartalmazza a két osztály kulcsait.

## 3.3. Lombok

A Lombok [3] egy java könyvtár, mely segít a forráskód olvashatóbb megvalósításában, hiszen például aomboknak köszönhetően nem kell megírni az osztályokhoz tartozó getter, setter metódusokat és konstruktorokat. Ezen felül az

adattagok is elláthatók például @NotNull annotációkkal amellyel biztosítható hogy az adattagok inicializálása megtörténjen .

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Rating {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private Date date;
    private int rating;
    private String ratingMessage;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private User user;
}
```

3.3.1 ábra: Egy java osztály létrehozása

Fontosabb annotációk például:

- @Getter - Ezzel az annotációval ellátott osztrákoknál a fordításkor legenerálja az osztály minden adattagjához tartozó gettereket.
- @Setter -Az osztályhoz tartozó adattagokhoz generál settereket.
- @NotNull - Az ilyen ammotációval ellátott adattagok értéke nem lehet null.
- @NoArgsConstructor - paraméter nélküli konstruktor, ahol az adattagok típusától függően 0, false vagy null értékekkel inicializálja azokat.
- @RequiredArgsKonstruktor - egy paraméteres konstruktort generál minden olyan adattaghoz, amely nem inicializált és final tulajdonsággal van ellátva vagy az adattag @NotNull annotációval van ellátva. A @NotNull annotációval ellátott adattagoknál a konstruktor egy null ellenőrzés is tartalmaz, ha az érték null értéket vesz fel NullPointerException hibát dob.
- @AllArrgsKonstruktor - az osztály minden eleméhez generál konstruktort.

- `@Data` - Az annotáció használatával az osztályhoz tartozó minden általános metódus (getter, setter, toString, requiredArgsConstructor) generálása megvalósul.

### 3.5. Angular

A kliens oldal fejlesztéséhez Angular [4] keretrendszert választottam. Az Angular a Google által fejlesztett, nyílt forráskódú komponens alapú keretrendszer SPA (single page application) webalkalmazások fejlesztésére.

Az SPA olyan webalkalmazás, ahol nincs szükség a teljes oldal újratöltésére az interakciók során, ezt a DOM (Document Object Model) manipulálásával tudja megvalósítani.

Elsősorban webes tartalmak frontend fejlesztésére használják, de támogatja a PWA (progressive web application) készítését is.

#### 3.5.1. Komponens

Egy angular projekt komponensekből és modulokból épül fel.

A komponensek tartalmazzak egy template fájlt, ami a megjelenést és más komponenseket tartalmazhat, és ts fájlt, ami egy osztály a template-hez szükséges adatok lekérése és a különböző elemek vezérlése történik.

Ahhoz hogy ez az egész komponensként is működjön, `@Component` annotációval kell ellátni az osztályt. Ebben az annotációban lehet megadni a komponens különböző tulajdonságait:

- `selector` - A komponens elnevezése, ezen a néven tudjuk más komponensekben használni azt.
- `templateUrl` - A megjelenítendő komponenst HTML része. Itt történik a komponens vizualizációja.
- `styleUrls` - A template fájlra vonatkozó stíluslapokat itt adhatjuk meg

```

@Component({
  selector: 'app-change-password',
  templateUrl: './change-password.component.html',
  styleUrls: ['./change-password.component.css'],
})
export class ChangePasswordComponent {

```

3.5.1. ábra: Komponens létrehozása

### 3.5.2. Lifecycle Hooks

Minden komponens életciklussal rendelkezik. Az angular rendelkezik lifecycle hook-okkal, melyek a komponens különböző életeseményekor hívhatók meg. A lifecycle hookok a konstruktor lefutását követően aktiválódnak.

Gyakoribb életciklusok:

- ngOnChanges - lefut, ha a komponens @Input paraméterei változnak
- ngOnInit - lefut az ngOnChanges után, akkor is lefut, ha az ngOnChanges nem érzékelt változást
- ngDoCheck - az ngOnChanges után hívható meg. Olyan változások esetén alkalmazza, amelyeket az angular nem vesz észre.
- ngAfterViewInit - A template betöltése után fut le.
- ngOnDestroy - Közvetlenül a komponens megszűnése előtt fut le

### 3.5.3. Modul

A komponenseket modulok fogják össze. Ahhoz, hogy egy osztály modulként működjön, @NgModule annotációval kell ellátni, és a következő tulajdonságait kell beállítani:

- declarations - Ebben a listában a modulhoz tartozó komponenseket, pipe-okat, és direktívákat adhatjuk meg.
- imports - Más használni modulok listáját adhatjuk itt meg, a primeNg-ből használt UI (user interface) elemeket itt importálom
- exports - A modulban szereplő komponensek listája, amelyeket más modul is használhat
- providers - Azon service-ek listája, amelyeket a modul komponensei használhatnak.

- bootstrap - Ebben a pontban az alkalmazás kezdő komponensét kell megadni, ezt általában csak az AppModule-ban szokták beállítani, és a fő komponenst tartalmazza.

Minden Angular alkalmazásnak van legalább egy gyökér modulja (AppModule), amely az alkalmazás belépési pontja is egyben.

### 3.6. Angular CLI

Az angular CLI [5] (command-line interface) egy parancssori kezelőfelület, amely segítségével egy angular projektet egyszerűen el lehet kezdeni, futtatni és új elemeket generálni.

Pár hasznos parancs, amelyet a szakdolgozatom során alkalmaztam:

- `ng new <projekt_neve>` - Az `ng new` paranccsal egy új angular projektet lehet létrehozni, amely felépíti a projekt vázát egy `appModule`-t, `AppComponent`-t és a komponensek navigálásáért felelős `app-routing` module-t.
- `ng generate <típus> <név>` - A `generate` parancs létrehozza a megadott angular specifikus fájlokat és a típusa alapján fel annotálja azokat. Komponensek esetén beilleszti azokat a modulba (alapértelmezetten az AppModule-ba). Több típusú állomány generálása lehetséges, például `service`, `component`, `module`, `interface`, `enum`.
- `ng serve` - Az alkalmazás futtatására használható. Több hasznos kapcsolója is létezik:
  - `--open` : ami az alkalmazás indulását követően megnyitja a böngészőben az alkalmazást egy új ablakban.
  - `--host <ip cím>` : lokális hálózaton belül más eszközről is elérhetővé válik az alkalmazás a megadott ip címen, így könnyedén tesztelhető az alkalmazás különböző platformokon.
  - `--port` : Az alkalmazás eléréséhez szükséges port



- “-- watch” : Az alkalmazás fejlesztése során a forráskód változásakor automatikusan frissíti az alkalmazást.
- ng lint - Konfiguráció alapján a forráskód elemzése, ha nem felel meg a konfigurációban megadott tulajdonságoknak a forráskód, akkor jelzi a hibát

### **3.7. PrimeNg**

A primeNg [6] egy nyílt forráskódú user interface könyvtár, amely lehetővé teszi a gyors és reszponzív fejlesztést. A PrimeNG megkönnyíti a különböző komponensek használatát, ami segít az egyoldalas alkalmazások (SPA) létrehozásában .

Több animációt is tartalmaz, például a gomb komponensekhez elérhető a ripple effektus, amely egy kattintási animációt jelent, ezzel a felhasználó jobban látható visszajelzést kap a gomb kattintásáról, így javítható a felhasználói élmény.

Másik nagyszerű komponense a táblázat amely alapértelmezetten támogatja a lapozás, oszlopok szerinti szűrést és rendezést. Ezeket a funkciókat könnyedén alkalmazhatjuk a komponens meghívásakor különböző tulajdonságok beállításával.

## **4.Fejlesztési folyamatot támogató alkalmazások**

A fejlesztési folyamatok nyomon követésére két alkalmazást használtam. A feladatok állapotának követésére a trello.com alkalmazást használtam, amely egy munkafolyamat irányítási rendszer és a forráskód verziókövetéséhez a github-ot.

### **4.1. Trello**

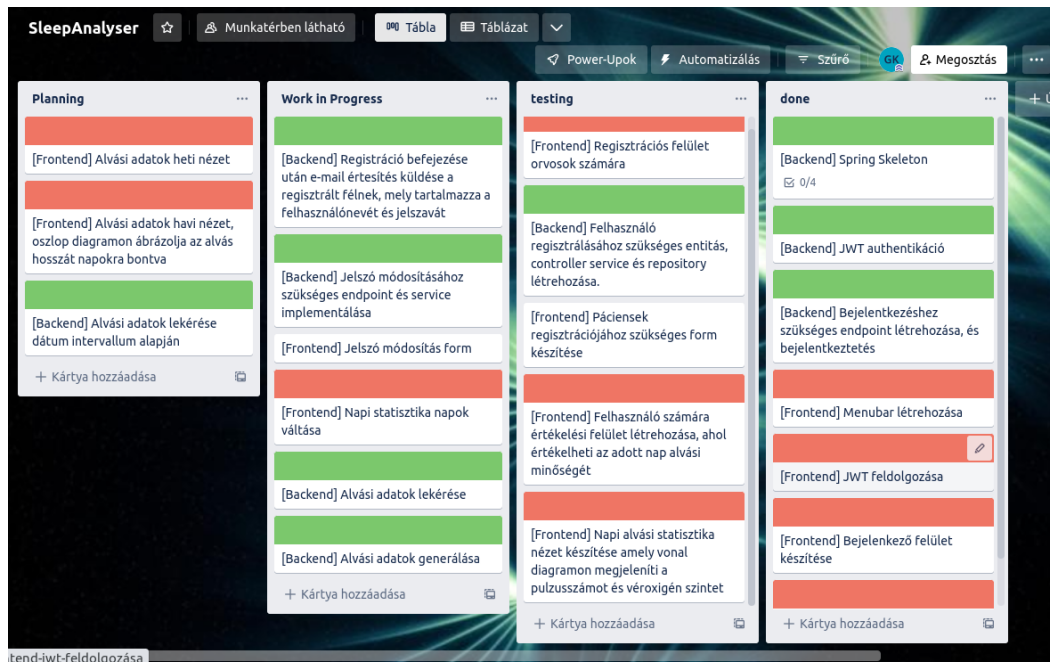
A csoportos fejlesztés során fontos, hogy a fejlesztők lássák mely feladatok elvégzése tartozik hozzájuk és az elvégzett feladatok milyen készültségi szinten állnak, ezért a projektmunka fejlesztése során a feladatok nyomon követésére a kanban fejlesztési módszertant alkalmaztam.

A kanban[8] egy lean módszer emberek csoportos munkájának menedzselésére és fejlesztésére. A kanban alapja egy táblázat, ahol meghatározzuk a munkafolyamatokat amelyeken egy feladat végig megy, ez lesz a feladat életciklusa. A táblázat egyes oszlopai reprezentálják a munkafolyamatokat.

Minden feladat a “planning” státuszba kerül, itt készül a feladat pontos leírása amit a fejlesztés során meg kell valósítani.

Ha a fejlesztő egy feladaton elkezd dolgozni, akkor a feladat kártyáját át kell mozgatnia a “Work in progress” oszlopba, így mindenki számára egyértelmű, hogy valaki elkezdett dolgozni az adott feladaton.

A feladat implementálása után átkerül a “Testing” fázisba, ahol a tesztelő ellenőrzi a működést és sikeres teszt után a feladat elvégzettnek “Done” tekinthető.



4.1. ábra: A projektmunkán feladatainak kanban táblázata.

## 4.2. GitHub

A verziókövetés nagyon fontos eleme a szoftverfejlesztésnek, mert a forráskód változásait valahogy követnünk kell, hiszen egy projekten belül több fejlesztő és párhuzamosan dolgozik, és a forráskódban végzett változásaik beillesztését vezérelni kell.

Tehát a verziókövetés a forráskód nyomon követését és forráskódban bekövetkezett változások kezelését jelenti. A fejlesztés során a Git forráskód-kezelő rendszert használtam, a verzió-kezelt adatok tárolására pedig Github -ot használtam.

A verzió-kezelésnél általában két alap elágazás jelenik meg. A “master” vagy “main” branch, amely általában a már kiadott és alkalmazás formájában elérhető forráskódot tartalmazza.

A "Main" branch egyik elágazása a "develop". Ezen az elágazáson a már kész és már egy belső rendszeren tesztelt, működő forráskódokat tárolják. Ebből az ágból lehet új feladatokat indítani és ebbe az ágba kerül beillesztésre egy elvégzett feladat. Egy feladat elvégzése után a forráskód feltöltését követően a githubon lehetőség van úgynevezett "Pull request" létrehozására, amely az elvégzett feladat branch-ét és egy cél brach-et tartalmaz, ahova az új forráskódot szeretnénk hozzáadni. Ilyenkor megy kérelem létrehozása történik, ahol a fejlesztő látja a forráskódban történt változásokat és az ehhez tartozó rövid leírást. Ezt követően a pull request elfogadásával a felhasználó által létrehozott ágon lévő forráskód beillesztésre kerül a célkén megadott branch-be.

## **Összegzés**

Összességében úgy gondolom, a szakdolgozatom során egy olyan alkalmazást sikerült elkészíteni ahol felhasználók tekinthetik meg az alvási adatainak kiértékelését és megoszthatják ezt kezelő orvosukkal. A projekt munka során úgy gondolom tanultam olyan dolgokat, melyeket későbbi

## ***Irodalomjegyzék***

1. Java spring - Hivatalos oldal: <https://spring.io/projects/spring-framework>
2. Java initializr - Hivatalos oldal: <https://start.spring.io/>
3. Lombok - Hivatalos oldal: <https://projectlombok.org/features/>
4. Angular - Hivatalos oldal: <https://angular.io/guide/architecture>
5. Angular CLI - Hivatalos oldal: <https://angular.io/cli>
6. PrimeNG - Hivatalos oldal: <https://www.primefaces.org/primeng/>
7. Kanban - wikipédia:  
[https://hu.wikipedia.org/wiki/Kanban\\_\(szoftverfejleszt%C3%A9s\)](https://hu.wikipedia.org/wiki/Kanban_(szoftverfejleszt%C3%A9s))
8. Alvási pulzusszámmal kapcsolatos cikk:  
<https://healthcarestrategyconsulting.com/puls-vo-sne-norma>

## **Nyilatkozat**

Alulírott Kazár György gazdaságinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, gazdaságinformatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Diplomamunka Repozitóriumban tárolja.

Szeged, 2022.november xx.

.....  
aláírás