ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

Oltáskampányt támogató rendszer

Vaccination campaign support system

OE-NIK Hallgató neve: **Szabados Gergely**
2022 Hallgató törzskönyvi száma **T/006374/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Szabados Gergely**
Törzskönyvi száma: T/006374/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Oltáskampányt támogató rendszer
Vaccination campaign support system

Intézményi konzulens: Fésüs Péter
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia specializáció

A feladat

Tervezzen és építsen egy olyan rendszert, melynek célja, hogy járvány idején megkönnyítse az oltásra való jelentkezést, a vakcinabeadás adminisztrációját, segítse annak operatív működését és az átláthatósággal növelje az oltáskampány iránti bizalmat.

Valósítson meg a .NET keretrendszerben egy olyan webalkalmazást, ami megadott API-n keresztül egy adatbázishoz kapcsolódik és képes módosítani a benne eltárolt adatokat. A webalkalmazás legyen többfelhasználós és tartalmazzon legalább három szerepkört: adminisztrátor, orvos és oltásra jelentkező. Az adminisztrátor szerepkörű userek bármilyen eltárolt adatot módosíthatnak, illetve képesek új felhasználó létrehozására.

A feladat része egy olyan weboldal fejlesztése, ahol az adatbázisban eltárolt adatok alapján kimutatások tekinthetők meg (oltások korosztály szerinti eloszlása, naponta beadott oltások száma). A webalkalmazásban és a hozzá tartozó weboldalon legyen lehetőség a nyelvek közötti váltásra. Biztosítson CAPTCHA védelmet a kártékony botok ellen.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- a lehetséges megvalósítási módokat és megoldásokat,
- a részletes rendszertervet és annak indoklását,
- a megvalósítás során használt technológiák bemutatását,
- a megvalósítás menetét, a munkafázisok és a tapasztalatok leírását,
- a tesztelési módszereket, tesztelési terveket és a tesztelés eredményeit,
- az elkészült rendszer alkalmazási, és továbbfejlesztési lehetőségeit,
- mellékleten a kifejlesztett rendszer forráskódját, illetve a készített dokumentációkat.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. április 27.

Szabady Gergely

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Szabados Gergely .....
Neptun kód: [redacted] .....
Tagozat: Nappali .....

Telefon: [redacted]
Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Oltáskampányt támogató rendszer.....

Szakdolgozat / Diplomamunka[2] címe angolul:

Vaccination campaign support system .....

Intézményi konzulens: Fésüs Péter.....

Külső konzulens: .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.10.07.	Feladatkiírás átbeszélése	Fésüs Péter
2.	2021.10.14.	Szakdolgozat szerkezeti vázlatának megbeszélése	Fésüs Péter
3.	2021.11.25.	Szakirodalmi összefoglaló megbeszélése	Fésüs Péter
4.	2021.12.02.	Szükséges módosítások megbeszélése	Fésüs Péter

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021. 12. 02. .....

Fésüs Péter
.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Szabados Gergely .....

Neptun kód: [redacted] .....

Tagozat: Nappali .....

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Oltáskampányt támogató rendszer

Szakdolgozat / Diplomamunka[2] címe angolul:

Vaccination campaign support system

Intézményi konzulens: Fésüs Péter

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.24.	Szakdolgozat II. feladatok átbeszélése	Fésüs Péter
2.	2022.03.24.	Szükséges módosítások megbeszélése	Fésüs Péter
3.	2022.04.14.	Szükséges módosítások megbeszélése	Fésüs Péter
4.	2022.04.21.	Alkalmazás bemutatása	Fésüs Péter

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022. 04. 25.

Fésüs Péter
.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

1. Bevezetés

Szakdolgozatom témája egy olyan rendszer tervezése és fejlesztése volt, amely megkönnyíti járvány idején az oltásokra való jelentkezést, annak adminisztrációját, az oltások igazolását, illetve könnyen értelmezhető diagramok segítségével bővebb információval ellátni a felhasználókat az oltáskampánnyal kapcsolatos adatokról.

A védőoltások hatékonyságát több kutatás is bizonyította, illetve a statisztikai adatok is azt mutatják, hogy az oltások felvételének hatására csökkenek az esetszámok. A HUN-VE (Hungarian Vaccine Effectiveness, Magyar oltóanyag-hatékonyság) kutatás eredményeinek alapján a védőoltás 88-98%-ra emelte a védelmet a betegség halálos kimenetele, és 69-89%-ra a fertőzéssel szemben. [1]

Az alábbi ábrán megfigyelhető, hogy a 2020 végén kezdődött oltáskampány hatására 2021 nyarára sikerült drasztikusan lecsökkenteni az esetszámokat. [2][3]

1. ábra: Napi új esetszámok az összes beadott oltáshoz viszonyítva

Szakdolgozatom ötletét a 2020-ban kitört COVID-19 járvány elleni oltások igénylésére létrehozott rendszer adta, azonban abból véleményem szerint több dolog hiányzott, amelyek elősegíthették volna a gördülékenyebb működést. Munkámban a rendszertervezés és fejlesztés mellett ilyen bővítményeket is szeretnék bemutatni.

Az első legszembetűnő probléma dolog a külön weboldalra kiszervezett regisztráció, de az időpont kiválasztása során sem a legkényelmesebb felhasználói élményben van része a rendszert használó személyeknek, hiszen a helyszínválasztás esetében csak irányítószámokra lehet keresni. Komoly hiányosság emellett, hogy a sikeres időpontfoglalás esetén nincs lehetőség a foglalás törlésére.

Szakdolgozatomban az általam tervezett rendszer tartalmaz egy olyan webalkalmazást, amelynek segítségével az emberek oltási időpontokat foglalhatnak, oltási statisztikákat tekinthetnek meg, illetve letölthetik az oltási igazolásukat, amennyiben megkapták a harmadik oltást. Az oltási statisztikák regisztráció nélkül is elérhetőek annak érdekében, hogy növeljem az oltás iránti bizalmat. Az adatok vizualizációja során a Google Charts-ot fogom használni. Ha valaki szeretne oltást igényelni, akkor ahhoz először felhasználói fiókot kell létrehoznia. Csak a sikeres regisztrációt követően lehet oltásra jelentkezni, ahol egy felhasználóbarát felületen választható ki a megfelelő időpontot. Jelentkezést követően egy megerősítő email kerül kiküldésre a páciens email címére, ami tartalmaz egy naptárfájlt annak érdekében, hogy könnyedén importálhassa a saját naptárjába az időpontot. A webalkalmazás admin felületén lehetséges az eltárolt adatok szerkesztésére. Az alkalmazást többfelhasználósra terveztem annak, érdekében, hogy csak a megfelelő jogosultsággal rendelkező emberek férjenek hozzá a különböző funkciókhoz (pl. adatok szerkesztése, felhasználók kezelése). Munkámban 4 szerepkör van: adminisztrátor, orvosok, ellenőrök és páciensek (akik az oltásokra jelentkeznek). Fontosnak tartom, hogy az alkalmazás többnyelvű legyen, annak érdekében, hogy a magyarul nem tudó felhasználók is használhassák.

Célom volt emellett még egy Backend (API és adatbázis) kialakítása, ami kiszolgálja a korábban említett webalkalmazást a szükséges adatokkal. A rendszert úgy tervezem, hogy bármikor lehessen bővíteni bármilyen egyéb komponenssel ezáltal növelve a felhasználói élményt.

2. Alkalmazott technológiák

A piacon jelenleg több technológia áll rendelkezésre a kliens oldali webfejlesztéshez, amelyek közül a legjelentősebbek a JQuery, az Angular és a React. Mindhárom technológia esetén rengeteg dokumentáció és előre megírt bővítmény áll rendelkezésre az interneten. A JQuery egy Javascript keretrendszer, amely egyszerűsíti a Javascript fejlesztést, az AJAX kérések és a DOM elemek elérését. Az Angular egy komponens alapú keretrendszer, amelynek segítségével skálázható webalkalmazások készíthetőek, emellett különválasztja a szerver oldali kódot a kliens oldalitól. A kliens és szerver oldali fejlesztéshez végül az ASP.NET-et választottam, mivel fontos volt számomra, hogy a frontend és backend ne különüljön el teljesen egymástól. Az ASP.NET használata során a Razor szintaxis segítségével könnyen irányíthatók a felhasználói oldalon megjelenő adatok.

2.1 ASP.NET

Az ASP.NET egy Microsoft által fejlesztett keretrendszer, amellyel dinamikus weboldalakat, webalkalmazásokat és webserviceket lehet fejleszteni. Első verziója 2002-ben jelent meg. Nevét a szintén Microsoft által fejlesztett ASP-ről (Active Server Page) kapta, azonban a kettő között igen sok különbséget lehet felfedezni.

„Az ASP az IIS (Internet Information Server) kiegészítéseként lehetővé teszi dinamikus weboldalak, web alkalmazások és XML Web Service -ok írását. Az ASP oldalak írását számos beépített objektum teszi egyszerűvé. Minden objektum gyakran használt funkciók egy csoportját valósítja meg, melyek hasznosak dinamikus oldalak fejlesztésénél. ASP 2.0-tól 6 ilyen beépített objektum van: Application, ASPError, Request, Response, Server és Session.” [4]

Az ASP esetében amikor a kliens letölti az oldalt, akkor kap egy HTML-t kap és azt értelmezi a böngészője, az ASP.NET esetében viszont az első betöltéskor le kell fordítani a kódot (emiatt először lassabb, mint a sima ASP), de az újbóli betöltések során már gyorsabb lesz, mivel akkor már a lefordított változat fog futni. Másik nagy előnye, hogy a szintaktikai hibák a fordítás során kiderülnek és így nem akad meg futás közben a program.

Az ASP.NET a .NET futtató környezetére épült, azaz a CLR-ra (Common Language Runtime). Ennek köszönhetően a fejlesztők bármely a .NET által támogatott nyelven írhatnak kódot (például C# és Visual Basic). Több fejlesztési modellt támogat, mint például: ASP.NET Web Forms, ASP.NET MVC, ASP.NET Web Pages, ASP.NET Web API. [5]

A keretrendszer bemutatása során megkerülhetetlen az esemény alapú programozási modell. Ennek működési elve az alábbi: A kliens oldalán váltódnak ki az események különböző felhasználói interakciók következtében, mint például kattintás, módosítás (pl. rádiógombnál). Ennek következtében a kliens oldalon egy esemény váltódik ki egy HTTP kérés formájában, aminek a kezelése a szerver oldalon történik meg a megfelelő eseménykezelővel. Ezt követően a szerver visszaküld egy HTML-t a kliensnek.

2.2 ASP.NET Core

Az ASP.NET Core a Microsoft által fejlesztett nyílt forráskódú webes keretrendszer, ami egyesíti a korábban fejlesztett keretrendszerek funkcióit (az ASP.NET MVC-t és az ASP.NET Web API-t). 2016-ban adták ki az első változatát. Az ASP.NET Web API nevéből adódóan leginkább az API-k (Application Programming Interface - Alkalmazásprogramozási felület) fejlesztésére lett tervezve. Működése során nem HTML válaszokat küld vissza a szerver, hanem egy objektumot JSON vagy XML formátumban. A másik fontos alkotóelem az ASP.NET MVC keretrendszer, ami a Modell-View-Controller tervezési mintát valósítja meg. Az MVC három egymástól különböző rétegből áll, amelyek jól elkülönülnek egymástól, viszont ennek ellenére mégis laza kapcsolatban állnak egymással. A modell réteg az adatok kezeléséért és tárolásáért felelős, a view a felhasználói felület kinézetéért, a Controller pedig a korábban említett két réteg közti kommunikációért, mondhatni egy irányító szerepet tölt be. [6]

2.3 Az MVC működése

Először a felhasználó egy web requesten keresztül kapcsolatba lép a Controllerrel oly módon, hogy a címsorba beírja az URL-t, majd az ezt követően az alkalmazás routing (útvonalválasztó) komponense megmondja, hogy a Controller melyik metódusát kell

meghívni. A Controller osztályokon belüli metódusok („Action”-ök) közül csak az egyik kerül meghívásra és az ő feladata lesz, hogy a kimenetet megfelelően létrehozza.

Fontos kiemelni, hogy a Controller nem végez mást műveletet, mint az adatok konvertálását/konverzióját. A műveletvégzés (pl adatbázis műveletek, számítási műveletek végzése) az üzleti logika feladata, ami modell rétegben található. Az adott Actiontől függ, hogy szükséges-e a Model réteg osztályaihoz fordulni (amelyek leginkább adatmodellek és a Business Logic), mivel előfordulhat csak egy sima HTML fájl küld vissza a kliensnek. Szükség esetén a modell rétegben megtörténnek a műveletvégzések és példány(ok) keletkeznek, amiket megkap a Controller. Ezt követően a Controller nézetet választ, majd a View-ba áttölti az adatokat. Legvégül a Controller ezt a HTML-t küldi vissza a felhasználó felé. [7]

2.4 CSHTML

A View rétegben kapnak helyet, azok a sablonok, amiket a felhasználó fog megkapni a Controller metódusai alapján és a modell rétegből kapott adatokkal töltődnek fel. A HTML és a CSHTML közötti legjelentősebb különbség, hogy a CSHTML-ben C# kódot is tudunk írni a Razor jelölőnyelv segítségével, ezáltal dinamikussá tudjuk tenni az alapból statikus HTML fájlunkat. A CSHTML tehát a HTML kód mellett tartalmazhat C# kódot, ehhez az fájl első sorában meg kell adnunk a @model kulcsszó segítségével, hogy milyen típusú lesz az általunk használt modell. Ezt követően a HTML kódon belül már csak annyi a teendő, hogy a @ jellel megnyissuk a C# kódblokkokat. Ennek köszönhetően például végig tudunk menni a kapott modellnek az egyik listáján vagy tömbjén vagy megjeleníti egyes tulajdonságainak az értékét. Az aktuális időt például az alábbi módon tudjuk kiirtani: `<p>@DateTime.Now</p>` [8]

2.5 CSS

A CSS (teljesen nevén Cascading Style Sheets – lépcsőzetes stíluslapok) egy stílusleíró nyelv, amivel a HTML fájlok testre szabhatóak. Ennek a segítségével lehet meghatározni, hogy a kliens oldalán megjelenő HTML-ben például milyen színű legyen a háttér, mekkorák legyenek a margók, milyen szélesek legyenek a táblázatok oszlopai. Ezeket a

HTML-ben belül is be lehet állítani, azonban ahogy nőtt az igény az egyre kifinomultabb elemekre és formázásokra célszerűbbnek tűnt mindezt kiszervezni egy külön fájlba.

Az elemek stílusát különböző CSS szelektorok segítségével lehet beállítani: Ha azt szeretnénk, hogy minden HTML elemre érvényes legyen az általunk írt stílusbeállítás, akkor a * jelet kell használnunk. Lehetőség van arra, hogy csak az elem neve alapján (pl: h1, a) vagy annak a leszármazottja alapján állítsunk be különböző stílusokat. Emellett létrehozhatunk egyéni beállításokkal rendelkező attribútumokat is (class és id). Az ezekben megírt stílus beállítások csak ott lesznek érvényben, ahol hivatkozunk rájuk az alábbi módon `class="<class_név>", id="<id_név>"`).

A CSS tehát az oldal stílusáért felelős. Egy CSS fájlt több HTML esetén is használhatunk, azaz különböző oldalak esetén ugyan azok a stílusokbeállítások lesznek érvényben. Ha például módosítjuk egy CSS-ben a margókat, akkor az minden olyan oldal esetében módosulást fog eredményezni, ami azt a CSS-t használja.

Ha a kliens először nyit meg egy oldalt, akkor a HTML-hez kapcsolódó CSS a cache (gyorsítótár) mappába kerül, azaz a következő oldalbetöltés során már nem kell újra letölteni a fájlt, mert már alapból ott lesz a kliens böngészőjében.

2.6 Google Charts

A Google Charts a Google által fejlesztett interaktív webszolgáltatás, aminek segítségével diagramokat lehet készíteni. Ennek segítségével futási időben vizualizálni lehet az adatbázisban eltárolt adatokat. Működési elve, hogy a HTML kódon belül el van helyezve egy Javascript kód, ahol át kell adni a modellben eltárolt adatokat, ami a Google erőforrásait felhasználva diagramokat készít. Előnye, hogy ingyenes, illetve személyre szabhatóak a diagramok. A microsoftos Power Bi-hoz hasonló diagramokat lehet vele készíteni, azonban annak a webes verziója nem ingyenes.[9][10]

3. Alkalmazott szoftverek

3.1 Visual Studio

A piacon jelenleg több integrált fejlesztői környezet áll rendelkezésre, a Visual Studio 2022-t választottam, mivel a .NET fejlesztés során ez nyújtja a legtöbb funkciót számomra. Az alábbi pontok miatt tettem le a voksomat a Visual Studio mellett:

- A moduláris telepítő (Visual Studio Installer) segítségével egyszerűen ki tudom választani, hogy milyen típusú fejlesztést szeretnék végezni (szakdolgozatom esetén ASP.NET Core 6.0) és egy gombnyomásra minden szükséges tool-t telepít. Természetesen, ha nincs rá szükségem, akkor itt tudom törölni őket.
- A kód írása közben lehetőség van Unit tesztelésre, azaz nem szükséges minden egyes tesztírás közben újra lefordítani a teljes solution-t.
- Az Intellisense kódkiegészítő segítségével jóval kényelmesebb a programozás, hiszen elírások esetén rögtön javaslatot tesz a javításra, illetve például metódusok implementálása esetén adatokat szolgáltat a paraméterekről (típus, kötelező-e a megadása, milyen célt szolgál).

3.2 SSMS

A piacon jelenleg több relációsadatbázis-kezelő rendszer található. Ezek közül a munkámhoz leginkább a PostgreSQL, a MySQL, az Oracle és a Microsoft SQL Server felel meg. A voksomat az utóbbi mellett tettem le, mivel széles körben használt és a „The Integrated stack selection" elvet követem, azaz egy gyártó szoftvereit használom, jelen esetben a Microsoftét, mivel fontosnak tartottam, hogy az általam használt eszközök kompatibilisek legyenek egymással, ezáltal könnyítve a saját munkámat.

Szükségem volt még a SQL Server Management Studiora (SSMS), ami szintén egy microsoftos termék. Ezzel a programmal könnyen szerkeszthetők és olvashatók az adatbázisban eltárolt adatok, speciális lekérdezésekkel bővebb információhoz juthatunk, írhatunk eljárásokat, illetve lehetőségünk van arra, hogy az adatokat ne lokálisan, hanem a Microsoft Azure felhőben tároljuk el. Munkám során lokálisan, a webalkalmazást futtató gépen tároltam el az adatokat, így nem volt szükségem az Azure nyújtotta lehetőségekre.

A SSMS rendelkezik egy olyan funkcióval, aminek segítségével ütemezett feladatokat tudunk végrehajtani az eltárolt adatokon. Ezen funkció neve SQL Server Agent. Ezzel lehet teljes adatbázis mentést készíteni, differenciális (előző back-up óta történt változás) biztonsági mentést, tranzakciós log backup-ot, illetve file vagy filecsoport mentést. [11]

3.3 Swagger

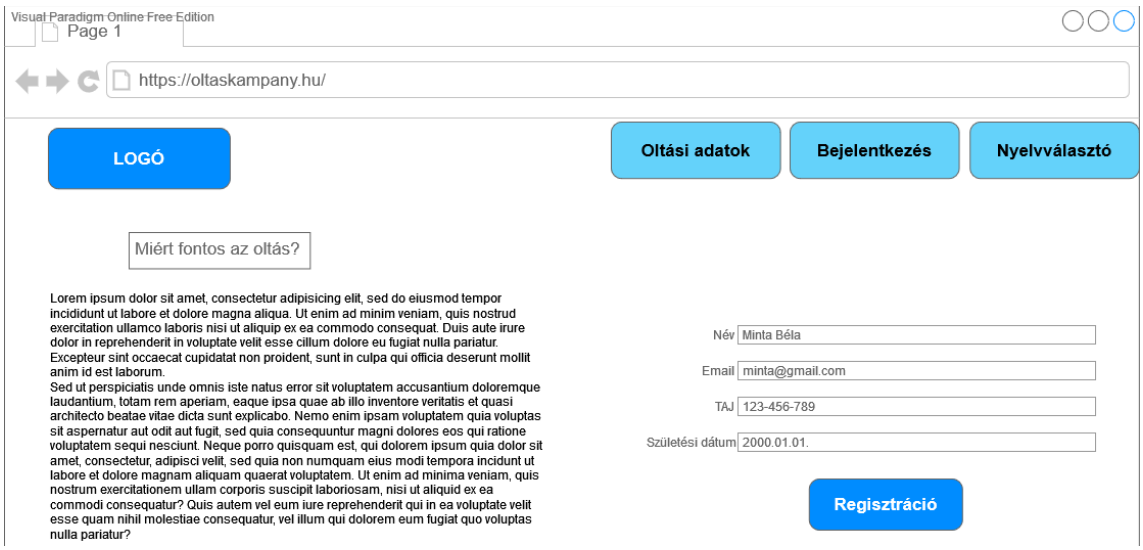
Az API teszteléséhez a Swaggert használtam, amit a Visual Studio-ban egy NuGet Package-ként lehet telepíteni. Ez a program a böngészőn belül olyan HTML oldalt generált le, amivel egyszerűen lehet tesztelni a HTTP hívásokat (GET, POST, HEAD stb.).

4. Általános követelmény specifikáció

Ebben a fejezetben azt mutatom be, hogy az általam fejlesztett rendszernek milyen követelményeknek kell megfelelnie. Tartalmaznia kell egy webalkalmazást, amelyben dinamikus és statikus oldalak találhatók, illetve egy backendet, ami az adatbázis módosításokért, illetve az adatok tárolásáért felelős.

Számomra kiemelt szempont volt, hogy a legfontosabb dolgok egy helyen összpontosuljanak. Jelen esetben például a magyar vakcinaregisztráció egyik problémáját hoznám elő: külön weboldalon volt a regisztráció és az utána következő oltási időpont kiválasztása. Az általam fejlesztett rendszerben egy helyen kell összpontosulnia a regisztrációtól kezdve az oltásigazolásig mindennek.

Minden egyes oldal tetején szerepelnie kell egy navigációs sávnak annak érdekében, hogy megkönnyítse a főoldal és az aloldalak közötti navigációt. A sávnak tartalmaznia kell egy logót, amelyre kattintva mindig vissza lehet jutni a főoldalra. A logótól jobbra kell elhelyezni a fontosabb aloldalakra, illetve a bejelentkezés oldalra mutató linkeket.



2. ábra: Főoldal látványterve

A főoldalon navigációs sáv alatti területnek két részből kell állnia: bal oldalon egy leírás az oltás fontosságáról, míg jobb oldalon egy regisztrációs blokk, ahol lehetőség van új felhasználói fiók létrehozására. Itt a teljes név megadása mellett meg kell adni a születési

időt, email címet, illetve a TAJ számot. A kártékony botok ellen reCAPTCHA-t alkalmazok. Tekintettel arra, hogy az EESZT belső rendszerhez nincs hozzáférésem, így kihagyom a TAJ validálást.

Ha a felhasználó sikeres kitöltötte a regisztrációs blokkban a szükséges mezőket, akkor egy visszaigazoló emailt kell kapnia. Ebben az emailben egy egyszeri jelszó szerepel (ezzel tud majd belejelentkezni, de utána rögtön meg kell változtatnia). Ezt követően egy olyan felület nyílik meg előtte, ahol több funkció közül választhat, amelyekről bővebben a következő alfejezetben fogok írni.

A főoldal mellett egy egyszerűbb aloldal létrehozását is fontosnak tartom annak érdekében, hogy egyéb érdekes információhoz jussanak az emberek a kampányt illetően. Ennek érdekében létre kell hoznom egy olyan aloldalt, ahol az oltási statisztikák tekinthetők meg dinamikus diagramokon.

A Frontend felület mellett szükséges egy backend (REST API) kialakítására is. Ennek feladata a user felületről érkező kérések feldolgozása, illetve információk szolgáltatása. Például, ha egy felhasználó új fiókot hoz létre, akkor ennek a backendnek a feladata a fiók létrehozása és a hozzá szükséges adatok eltárolása az adatbázisban. Fontos szempont volt számomra, hogy az adatbázisban eltárolt információkról lehessen biztonsági mentést készíteni (akár az adatbáziskezelő szoftverrel) annak érdekében, hogyha technikai probléma esetén leállnának a háttérszolgáltatások, akkor ne vesszenek el az adatok. [12]

Fontosnak tartottam, hogy széles körben használható legyen az alkalmazás, ezért többnyelvűnek kell lennie. A magyar nyelv mellett az angol és német nyelvet kell biztosítani a felhasználók számára. A nyelvek közötti váltás egy egyszerű legördülő menüvel kell biztosítani, amely a navigációs sáv jobb szélén található. A kiválasztott nyelvre kattintva minden szöveg át kell változnia a kiválasztott nyelvnek megfelelő változatára.

4.1 Felhasználók

A webalkalmazásnak többfelhasználósnak kell lennie annak érdekében, hogy a különböző szerepkörű felhasználók más-más funkciókhoz férjenek hozzá. Szakdolgozatomban 4 szerepkört fogok létrehozni:

- Páciensek: Csak az alapvető funkciókhoz férnek hozzá. Alapvetően nekik készül a rendszer, hiszen ők jelentkezhetnek oltásra.
- Orvosok: Csak a vakcinabeadást tudják bejegyezni.
- Admin/rendszergazda: Oltási időpontok, helyszínek és a felhasználók kezelése.
- Ellenőr: Megtekintheti a QR kód által mutatott oldal tartalmát.

4.2 Átláthatóság – Oltási adatok

Nagy hangsúlyt fektetek az átláthatóságra ezért egy olyan tájékoztató oldal is része a rendszernek, ahol diagramok segítségével egyszerűen és könnyen értelmezhetően bővebb információhoz juthatnak az emberek az oltáskampánnyal kapcsolatban. Az oldal eléréséhez a navigációs sávon belül az „Oltási adatok”-ra kell kattintani. Ennek az aloldalnak a megtekintéséhez nem kell regisztrálni, pontosan azért, hogy növelje az oltás iránti bizalmat. A diagramok létrehozásához a Google Charts-ot használntam, ami egy Google által fejlesztett ingyenes webszolgáltatás.

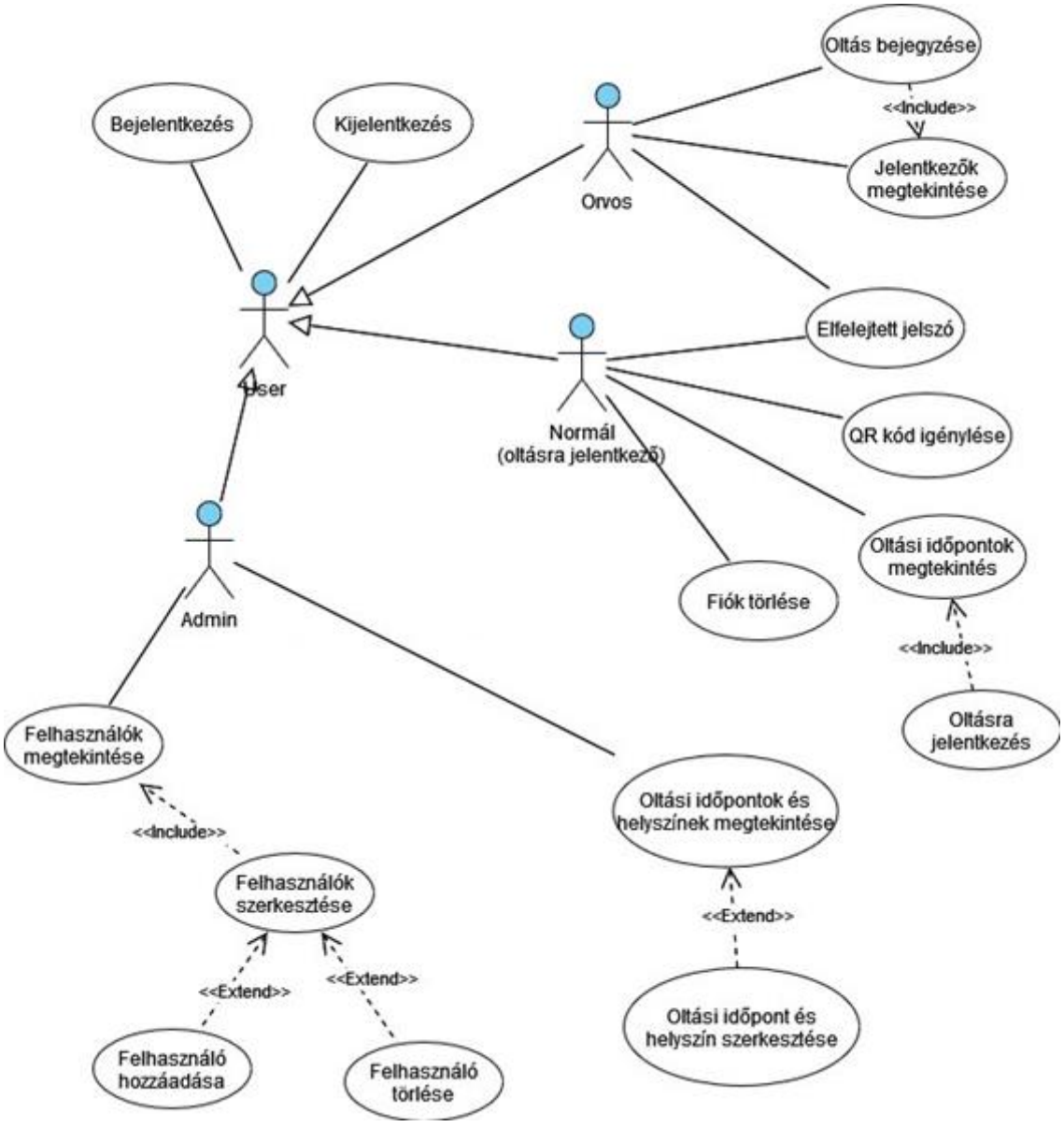
Az alábbi adatok ábrázoltam a diagramok segítségével:

- Naponta beadott oltások száma
- A regisztráltak hány százaléka kapott már oltást
- Korosztályok szerinti eloszlásban a beadott oltások száma
- Az újonnan regisztrálók száma naponta

5. Funkcionális követelmény

Ebben a részben azt szeretném bemutatni, hogy a rendszer használói számára milyen funkciók és szolgáltatások lesznek elérhetők, hogyan kell működnie az egyes fázisokban, illetve funkciók milyen módon érhetők el.

Alább olvasható, hogy milyen használati esetek (use case) fordulhatnak elő a különböző szerepkörű felhasználók esetében, milyen funkciókat érhetnek el és milyen módon használható a rendszer számukra.



3. ábra: Use Case Diagram

5.1 Regisztráció

A rendszer bizonyos funkcióhoz feltétlenül szükséges a regisztráció annak érdekében, hogy ki lehessen küszöbölni a nem megfelelő használatot, például nem létező személyekkel jelentkezni minden oltási időpontra. Regisztráció során a felhasználónak meg kell adnia a nevét, születési idejét, email címét és a TAJ számát. Az email cím lesz a felhasználónév. Az adatok megadását követően egy Google reCAPTHA tesztet is ki kell tölteni annak érdekében, hogy a rendszer megbizonyosodjon arról, hogy emberi próbálkozással akarnak új felhasználót létrehozni. A regisztráció gomb megnyomását követően a felhasználó kap egy emailt, amely egy ideiglenes jelszót fog tartalmazni.

5.2 Bejelentkezés

Ha a felhasználó nem változtatta meg az ideiglenes jelszavát, és a megfelelő ideiglenes jelszót írta be, akkor a bejelentkezés gomb megnyomását követően meg kell változtatnia a jelszavát, annak érdekében, hogy a rendszer további részéhez hozzáférhessen. A jelszavaknak legalább 8 karakterből kell állniuk, és az alábbiak közül 3-at kell tartalmazniuk: nagybetűk (A-Z), kisbetűk (a-z), számok (0-9) és speciális karakterek (!@#$%^&*). A Google reCAPTHA tesztet itt is ki kell tölteni, ahhoz, hogy bejelentkezés sikeres legyen.

5.3 Elfelejtett jelszó

Ha egy már regisztrált felhasználó elfelejtette a jelszavát vagy véletlenül kitörölte a regisztráció során kapott ideglenes jelszavát tartozó e-mailjét, akkor itt lehetősége van egy új ideiglenes jelszó igénylésére.

5.4 Bejelentkezési utáni funkciók szerepkörök szerint

5.4.1 Páciens oltottsági állapotának ellenőrzése

Ez egy olyan oldal, amit csak a bejelentkezett admin, doktor és ellenőr szerepkörű felhasználók, illetve az adott páciens (akinek az védettségi állapotát mutatja) láthatnak. A megnyitásához tudni kell a pontos URL-t (amit a QR kód tartalmaz). Ezen az oldalon megtekinthető az adott páciens oltásai, illetve, hogy védettnek minősül-e, azaz rendelkezik már 3 oltással és a 3. óta eltelt legalább 14 nap.

5.4.2 Admin szerepkörű felhasználók által elérhető funkciók

Elsőként az admin szerepkörű felhasználók számára elérhető funkciókat szeretném bemutatni. A nevéből adódóan ők lesznek felelősek a rendszerben eltárolt adatokért és azok napra készen tartásáért.

Felhasználók kezelése

Bejelentkezés követően automatikus a felhasználók szerkesztése oldal fog megjelenni, ahol listázásra kerül az összes admin és orvos szerepkörű felhasználó.

Új felhasználó hozzáadása

Az oldal tetején elhelyezésre kerül egy új felhasználó létrehozása gomb, amire rákattintva egy új oldal nyílik meg, ahol lehetőség lesz új orvos és admin szerepkörű felhasználók hozzáadására. A szerepkör kiválasztása mellett meg kell adni az új felhasználó nevét és email címét. Fontos, hogy a rendszerben eltárolt email címmel ne lehessen új felhasználót létrehozni.

Felhasználó adatok kezelése és törlése

Az újfelhasználó létrehozása gomb alatti területen lesznek kilistázva a felhasználók téglalap alakzatokban. Az alakzat tetején a felhasználó egyedi azonosítója szerepel, majd alatta a név és a szerepkör. Emellett 2 gomb is elhelyezésre kerül, melyek közül az egyikkel azonnal törölni lehet a felhasználót a rendszerből.

A másik gombra kattintva szerkeszteni lehet a kiválasztott felhasználó adatait. Meg lehet változtatni a felhasználó nevét, email címét, illetve a szerepkörét. Ha orvos szerepkörű a felhasználó vagy az lett kiválasztva, akkor meg kell jelennie egy új blokknak, ahol oltási alkalmakat(időpontokat) lehet rendelni a felhasználóhoz, illetve törölni a meglévőket.

Oltási időpontok/alkalmak szerkesztése

Ebben a menüpontban lehetőség van új oltási időpontok felvitelére, a meglévők szerkesztésére, illetve új oltási időpontok is hozzáadhatóak. A menüpontra kattintva az oldal tetején szerepelnie egy gombnak amire rákattintva új oltási alkalmat lehet létrehozni. Alatta ki kell listázni a már meglévő alkalmakat, ahol megjelenítésre kerül az adott alkalom ID-ja, dátuma, a jelentkezők és a férőhelyek száma, illetve a vakcina típusa.

Az alkalmakat lehessen törölni és módosítani. Módosítás során az alábbiakat lehet megváltoztatni:

- Az oltási alkalomhoz kapcsolódó oltási helyszín ID-ja
- Dátum
- Napszak (ennek egy szabadon szerkeszthető szövegnek kell lenni, annak érdekében, hogy komplexebb időpontok is megadhasson a felhasználó)
- Férőhely száma
- Vakcina típusa

Fontos, hogy csak létező oltási helyszínt lehessen az alkalomhoz rendelni. Ha nem létező, ID-t ír be a felhasználó, akkor a módosításnak sikertelen és hibaüzenettel jelzi a felhasználó felé a problémát.

Oltási helyszínek szerkesztése

Erre a menüpontra kattintva egy hasonló funkciók érhetőek el, mint az oltási alkalmak szerkesztésénél, azaz hozzá lehet adni új oltási helyszíneket, a meglévőket meg lehet tekinteni, szerkeszteni, illetve törölni őket. Listázás során megjelenítésre kerül a helyszínhez tartozó összes adat:

- Megye
- Város
- Irányítószám
- Utca
- Házszám
- Helyszín neve

A szerkesztésre gombra kattintva, az összes tulajdonságot lehessen szerkeszteni, bármilyen megkötés nélkül.

5.4.3 Ellenőr szerepkörű felhasználók által elérhető funkciók

Csak a páciensek oltottsági állapotát tudják megtekinteni.

5.4.4 Orvos szerepkörű felhasználók által elérhető funkciók

Vakcina bejegyzése

Tekintettel arra, hogy az orvosok legfőbb feladata a vakcina beadása, így csak ezt az egy funkciót érhetik el. Bejelentkezést követően listázásra kerül az összes olyan páciens, aki az orvos beosztásával megegyező alkalomra jelentkezett. Az orvosnak vakcina törzsszáma beírása mellett csak egy jóváhagyás gombot kell megnyomnia és az oltás azonnal bejegyzésre kerül.

5.4.5 Páciens szerepkörű felhasználók által elérhető funkciók

Fiók törlése

A pácienseknek lehetősége van a fiókjuk törlésére. Ebben az esetben a regisztrált oltásjelentkezések is törlődnek a rendszerből.

Oltásra való jelentkezés

Ebben menüben lehetőség van az oltási helyszínek, vakcina és időpont szerint szűrni. Ki lehet választani egy adott vakcinát és akkor csak olyan időpontok kerülnek listázásra, ahol a kiválasztott vakcinát adják. A helyszínek közötti keresés esetén először ki kell választani a megyét majd ezt követően a várost is. Ezt követően láthatóvá válnak a szabad időpontok, amelyek melletti található jelentkezés gombra kell kattintani az időpont véglegesítése érdekében.

Egyszerre csak egy időpontra lehet jelentkezni. A következő oltásra csak akkor lehet jelentkezni, ha az előző vakcina beadását követően eltelt 30 nap.

Oltásra való jelentkezést követően egy automatikus email kerül kiküldése annak érdekében, hogy a felhasználó megbizonyosodjon a sikeres jelentkezésről. Az email tartalmazza az oltás pontos adatait VCAL formátumban, amely importálható a felhasználó saját naptárába.

Oltásról való lejelentkezés

Ha van aktív jelentkezése a felhasználónak, akkor bejelentkezést követően legyen lehetősége törölni az aktív jelentkezést. Ebben az esetben a felhasználó nem kap emailben értesítést a törlésről.

QR kód generálása

Ha a páciens megkapta a harmadik oltását és a harmadik oltás óta eltelt 14 nap, azaz védettnek minősül, akkor egy QR kód jelenik meg az alkalmazáson belül a páciens oldalán, amit le is tud tölteni. Ennek a QR kódnak a segítségével tud megbizonyosodni egy harmadik fél arról, hogy az adott személy rendelkezik-e oltással. A QR kód beolvasását követően a link egy bejelentkezést igénylő oldalra vezet, ahol az oltások listázásra kerülnek, illetve, hogy az adott személy védettnek minősül e. Az oldal az adott páciensen (akinek a védettségét mutatja) kívül csak az ellenőr, admin, doktor szerepkörű felhasználók számára látható.

6. Adatbázis felépítés

Ebben a részben bemutatom a korábbi fejezetekben definiált követelmény specifikációknak megfelelően az adatmodellt. Az adatbázis tervezése során az alábbi lépések szerint haladtam:

1. Egyedek definiálása

Először meg kell határozni, hogy az adatmodellen belül milyen egyedek legyenek. Az egyedek olyan egymástól jól megkülönböztethető objektumok, amelyeket tulajdonságokkal jellemzünk. Törekedni kell a redundancia csökkentésére, mivel ez egyrészt felesleges tárhelyigényt jelent, illetve későbbiekben bonyolultabbá teheti a rendszer a karbantartását. Több egyed együttesen egyedhalmazt alkot, ami az én esetemben egy táblát jelent.

2.Egyedhalmazok mezőinek definiálása

Az egyedek definiálását követően el kell dönteni, hogy azok milyen tulajdonságokkal rendelkezzenek, azaz milyen adatokat szeretnénk eltárolni. Ezekhez elsőként beszédes mezőneveket kell kitalálni, majd ezt követően meghatározni az adott mező típusát, ami lehet szám, karakterlánc, bináris vagy dátum. Minden táblának kötelezően tartalmaznia kell egy elsődleges kulcsot (Primary key), aminek egyedinek kell lennie annak érdekében, hogy megkülönböztethessük a tábla többi rekordjától. Az elsődleges kulcs lehet egy egyetlen mező vagy akár több mező alapján összetett kulcs. A legegyszerűbb módja az elsődleges kulcsnak egy mező, amelyet automatikusan növekvő sorszámozásra állítunk, ezáltal biztosítva az egyediséget.

3. Kapcsolatok definiálása

Annak érdekében, hogy több táblából egyszerre tudjunk adatokat lekérdezni kapcsolatot kell teremteni közöttük. A táblák közötti kapcsolatokat új mezők beszúrásával tudjuk elérni. Ebben a mezőben lesznek eltárolva a hivatkozott tábla elsődleges kulcsai, amit itt idegen kulcsnak nevezünk (Foreign key). Egy tábla több idegen kulcsot is tartalmazhat. Létrehozhatunk megszorításokat annak érdekében, hogy csak létező kulcsok szerepelhessenek az idegen kulcsos mezőkben.

Az elméleti tervezést követően a Visual Studio 2022 fejlesztői környezetben létrehoztam az API projektet, amiben megírtam a modelleket, illetve definiáltam az adatbázis elérési útvonalát, amiről később teszek említést.

Code First megközelítést követtem, tehát az adatbázis tábláit C# nyelven osztályokban definiáltam, aminek előnye, hogy folyamatosan követi a kódon belül végzett módosításokat. Ha a modellek szerkesztése megtörtént, akkor a következő feladat a migráció készítése, ami definiálja az adatbázisba beszúrandó modelleken végzett módosításokat. Ahhoz, hogy ilyen migrációt készíthessünk az alábbi parancsot kell lefuttatni a Package Manager Consolon: „`Add-Migration [Migráció neve]`”. Ezt követően le kell futtatni az adatbázis frissítés parancsot (`Update-Database`), mivel csak ekkor változtatja meg az adatbázis tábláit.

Az API projekten belül Connection String segítségével definiáltam a fejlesztői környezet számára az adatbázis elérési útját:

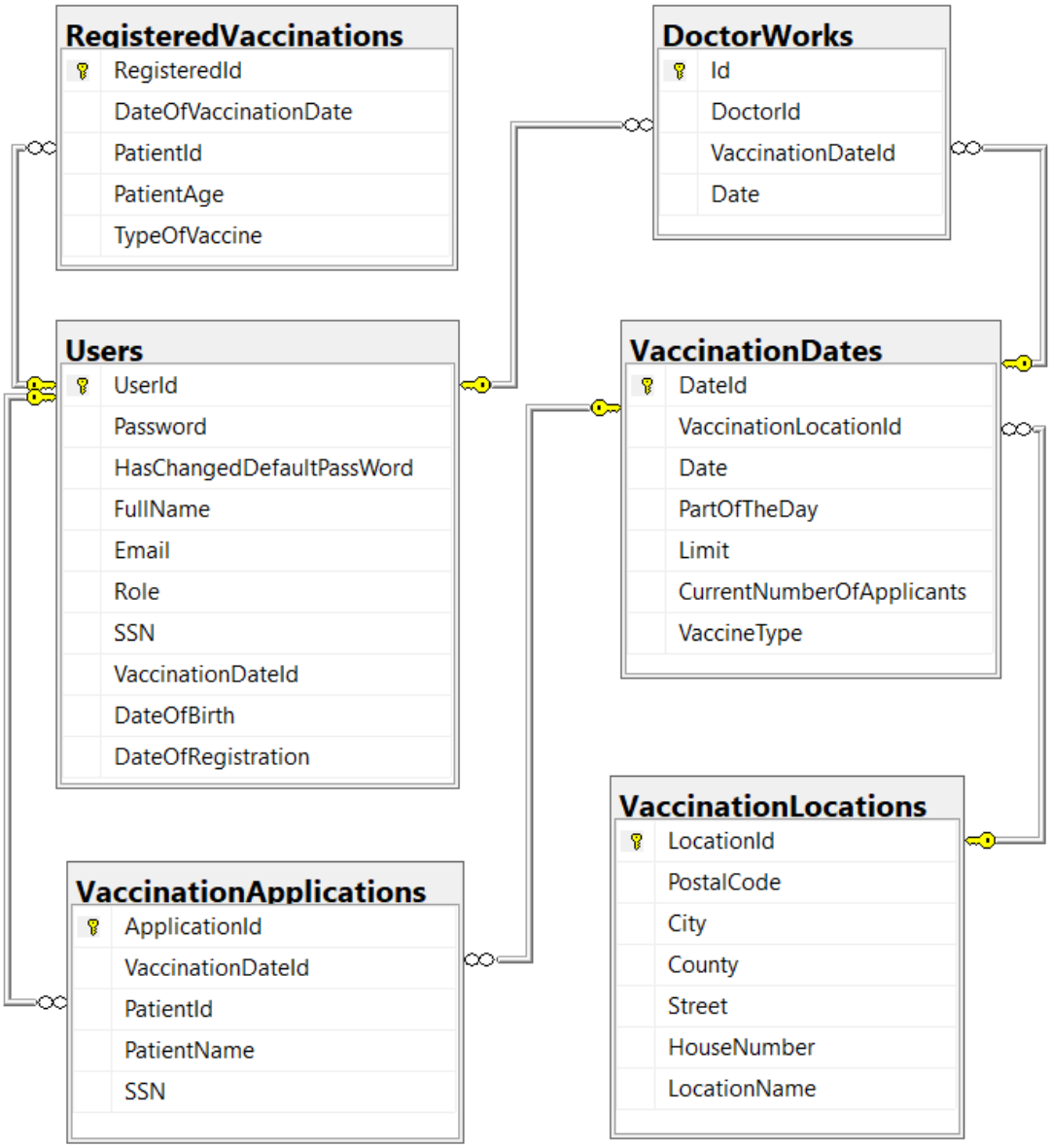
Az appsettings.json-ben elhelyeztem a Connection String-et:

```
 "ConnectionStrings":  {
    "ConStr":
"Server=.;Database=szakdolgozatDB;MultipleActiveResultSets=True;Trusted_Conn
ection=True;"
  },
```

majd hivatkoztam rá a Program.cs osztályban:

```
builder.Services.AddDbContext<SzakdogaContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("ConStr")
));
```

Az Update-Database parancs lefuttatását követően az SzakdolgozatDb adatbázisban az alábbi táblák kerültek beszúrásra:



4. ábra: Relációs adatbázis modellje

Ezt követően a SQL Server Management Studion belül manuálisan létrehoztam a kapcsolatokat a táblák között (Melléklet 16.1 fejezet), tehát idegen kulcs megszorításokat tettem a már meglévő mezőkre (amik másik tábla elsődleges kulcsait tartalmazzák). A megszorítás általános formája az SQL Server esetében az alábbi:

```
ALTER TABLE [Tábla_neve]
ADD CONSTRAINT [Megszorítás_neve]
FOREIGN KEY ([mező_neve_ami_idegen_kulcsokat_tartalmaz]) REFERENCES
[Hivatkozott_tábla_neve]([hivatkozott_mező_neve]);
```

6.1 RegisteredVaccinations tábla

Ebben a táblában vannak eltárolva azok az oltásokhoz tartozó adatok, amiket csak az orvosok tudnak bejegyezni. Az elsődleges kulcs a RegisteredId mező, amely egyesével növekvő értékeket tartalmaz. A PatientId egy idegen kulcs a Users tábla UserId mezőjére.

Neve	Tipus	Nullable	Tulajdonság
RegisteredId	Integer	NOT NULL	A bejegyzett oltás ID-ja
DateOfVaccinationDate	Dátum		Az oltás dátuma
PatientId	Uniqueidentifier		A páciens ID-ja
PatientAge	Integer		A páciens kora
TypeOfVaccine	Nvarchar(max)		Az oltás típusa
RegistrationNumber	Nvarchar(max)		A vakcina törzsszáma

1. táblázat: RegisteredVaccinations tábla

6.2 Doctorworks tábla

Itt vannak eltárolva, hogy a doktorok milyen beosztásban dolgoznak. Azért fontos ennek a táblának a megléte, hogy egy adott helyszínen, adott időpontban csak azokat a pácienseket mutassa az oda beosztott doktor(ok)nak, akik arra az alkalomra regisztráltak.

Neve	Tipus	Nullable	Tulajdonság
Id	Integer	NOT NULL	A munka ID-ja
DoctorId	Uniqueidentifier		A doktor ID-ja
VaccinationDateId	Integer		A vakcinaidőpont ID-ja
Date	Dátum		A munka dátuma

2. táblázat: Doctorworks tábla

6.3 Users tábla

Az adatbázis legfontosabb táblája, ami a felhasználókat és a hozzájuk tartozó adatokat tárolja el. Annak érdekében, hogy az adatbázisból ne lehessen látni a felhasználókhoz tartozó jelszavakat titkosításra van szükség, ami az alkalmazáson belül történik. A `HasChangedDefaultPassWord` mező azt a célt szolgálja, hogy a felhasználókat arra kényszerítsük, hogy az első bejelentkezés során (illetve új jelszó igénylését követően) kötelezően változtassák meg a jelszavakat.

Neve	Típus	Nullable	Tulajdonság
UserId	Uniqueidentifier	NOT NULL	A felhasználó egyedi azonosítója
Password	Nvarchar(max)		A felhasználó jelszava
HasChangedDefault PassWord	Bit		Megváltoztatta e már a kezdeti a jelszavát
FullName	Nvarchar(max)		A felhasználó teljes neve
Email	Nvarchar(max)		Email cím
Role	Nvarchar(max)		A felhasználó szerepköre
DateOfRegistration	Dátum		A regisztráció időpontja
SSN	Big Integer	Nullable	TAJ szám
VaccinationDateId	Integer		Oltási alkalom ID-ja
DateOfBirth	Dátum		Születési idő

3. táblázat: Users tábla

6.4 VaccinationsDates tábla

Az adatbázis másik legfontosabb táblája az oltási alkalmakat(időpontokat) tárolja. Az oltási alkalmak 3 fontosabb összetevője a helyszín, a dátum és a napszak. A helyszínhez tartozó adatokat a VaccinationLocatotions táblából tudjuk kinyerni a VaccinationLocationId (idegen kulcs) segítségével. A napszak karakterlánc típusú, tehát lehetőség van az időpontok speciálisabb leírására (Pl: „9-18 oltás, délben ebédszünet”). Ha limit egyenlő a jelentkezők számával, akkor már nem lehet jelentkezi az oltásra.

Neve	Típus	Nullable	Tulajdonság
DateId	Integer	NOT NULL	Az alkalom ID-ja
VaccinationLocationId	Integer		A helyszín ID-ja
Date	Dátum		Dátum
PartOfTheDay	Nvarchar(max)		Napszak
Limit	Integer		Férőhelyek száma
CurrentNumberOfApplicants	Integer		Az aktuális jelentkezők száma
VaccineType	Nvarchar(max)		Vakcina típusa

4. táblázat: VaccinationsDates tábla

6.5 VaccinationApplications tábla

Ebben a táblában vannak eltárolva az aktuális jelentkezések. Miután egy doktor megkezdi a munkát az alkalmazás egyszerűen megkeresi adott doktornak az aznapi beosztáshoz tartozó pácienseket a VaccinationDateId alapján.

Neve	Típus	Nullable	Tulajdonság
ApplicationId	Integer	NOT NULL	A jelentkezés ID-ja
VaccinatioDateId	Integer		Oltási alkalom ID-ja
PatientId	Uniqueidentifier		A páciens ID-ja
PatientName	Nvarchar(max)		A páciens neve
SSN	Integer		

5. táblázat: VaccinationApplications tábla

6.6 VaccinationLocations tábla

Ebben a táblában vannak eltárolva az oltási helyszínekhez tartotó adatok. A helyszínadatok kiegészülnek egy LocationName mezővel, annak értekében, hogy bővebb információkat is meg lehessen adni az adott lokációról. Például, hogy egy adott kórház melyik osztályán lesz az oltás.

Neve	Típus	Nullable	Tulajdonság
LocationId	Integer	NOT NULL	A helyszín ID-ja
PostalCode	Integer		Irányítószám
City	Nvarchar(max)		Város
County	Nvarchar(max)		Megye
Street	Nvarchar(max)		Utca
HouseNumber	Nvarchar(max)		Házszám
LocationName	Nvarchar(max)		Helyszín neve

6. táblázat: VaccinationLocations tábla

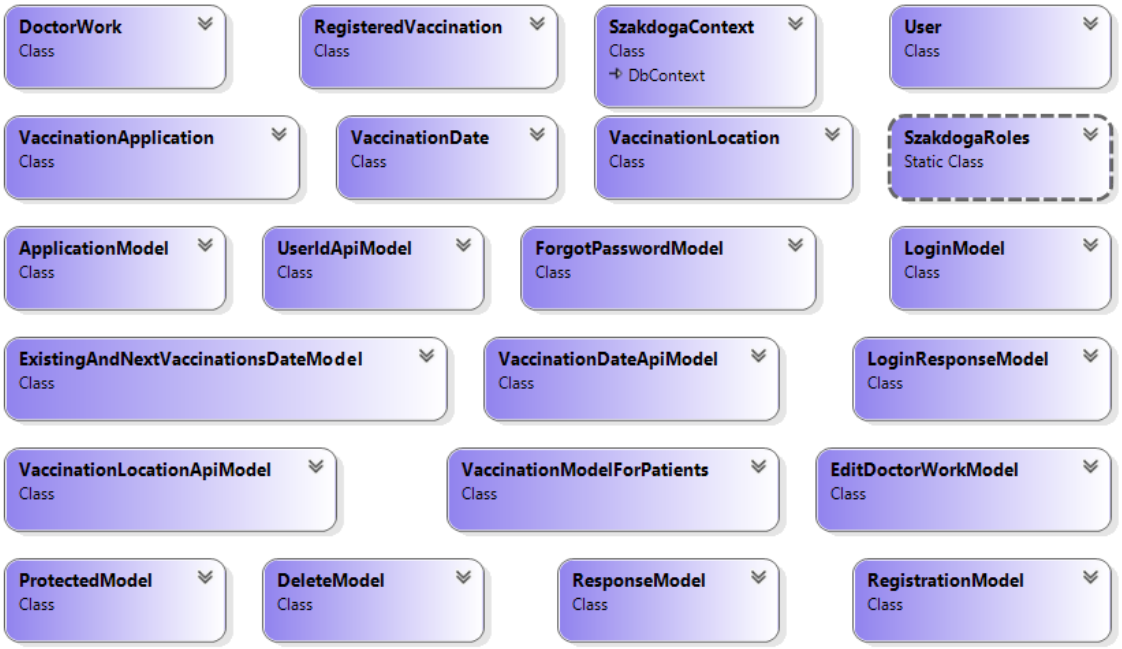
7. Megvalósítás

Az előző fejezetben említésre került, hogy készítettem egy Backend API projektet (SzakdolgoaztApi néven), de emellett létrehoztam még a Frontend felületet SzakdolgozatWeb néven. A két projekt közötti kommunikáció során JSON fájlok kerülnek küldésre. Mindkét projekt .NET 6 verziójú és a Visual Studio 2022-es fejlesztői környezetben hoztam létre őket.

7.1 Osztályok

Modellek

Az Adatbázis felépítés fejezetben bemutatásra került a 6 legfontosabb adatbázis modell, azonban a 7-es ábrán látható, hogy az adatbázismodellek kiegészülnek további osztályokkal, amik az API kommunikáció miatt szükségesek. A nevük beszédes, annak érdekében, hogy felismerhető legyen a felhasználási területük. Speciálisabb modellek is készültek, ilyen például az `ExistingAndNextVaccianionsDateModel`, ami a páciens meglévő és jövőbeni oltásaihoz kapcsolódó adatokat tárolja.

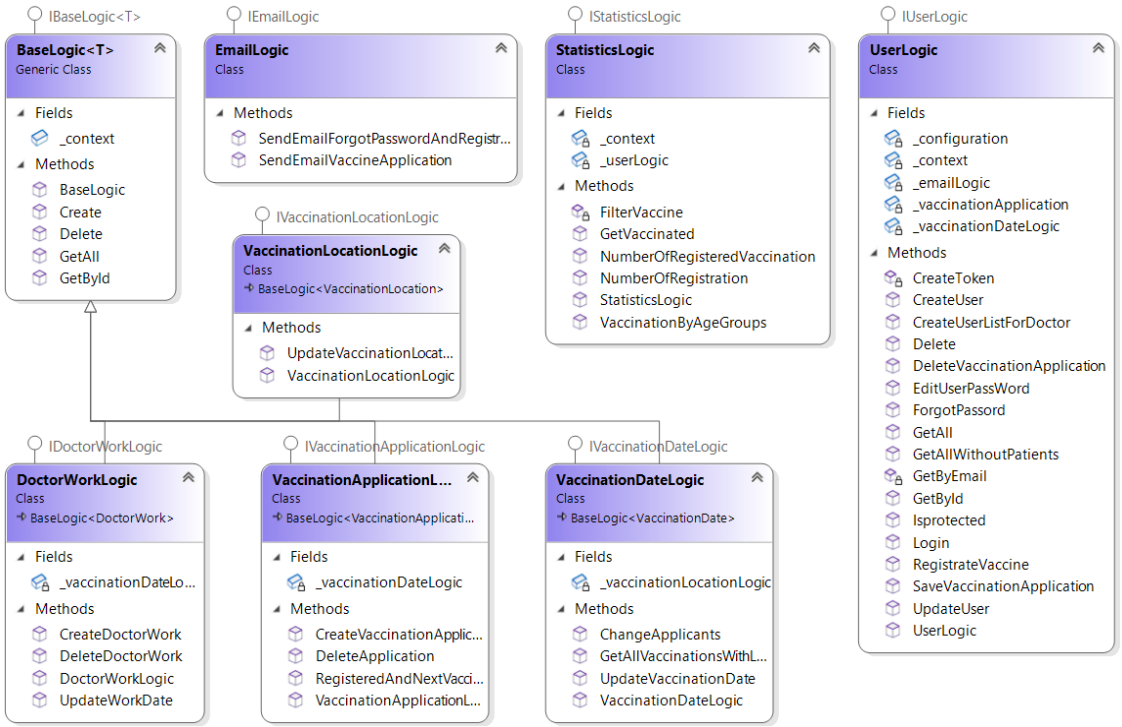


5. ábra: SzakdolgozatWeb Model réteg osztályai

Logikai réteg osztályai:

A logikai réteg osztályai 4 csoportra különíthetőek el.

- Oltási helyszínek és időpontok teljes körű szerkesztése (CRUD: Create, Read, Update, Delete). Oltásra történő jelentkezés és annak törlése, illetve az oltási alkalmakhoz orvos hozzárendelésre és annak törlése. (`VaccinationDateLogic`, `DoctorWorkLogic`, `VaccinationLocationLogic`, `VaccinationApplicaionLogic`)
- Az `UserLogic` osztály felelős az összes felhasználókkal kapcsolatos funkcióért, mint például új felhasználók létrehozása, a jelenlegiek kezelése, bejelentkezés, oltásjelentkezés és védettség ellenőrzése.
- A `StatisticsLogic` osztály felelős a bejelentkezés nélkül elérhető oltási diagramokhoz szükséges adatok generálásáért.
- Az `EmailLogic` osztálynak az alábbiakban van szerepe: Regisztrációt követően az ideiglenes jelszó kiküldése, elfelejtett jelszó esetén új jelszó, illetve oltásra való jelentkezést követően a sikeres jelentkezésről egy email küldése, ami tartalmazza egy Vcalendar fájlban az oltáshoz kapcsolódó fontos információkat (helyszín és időpont).

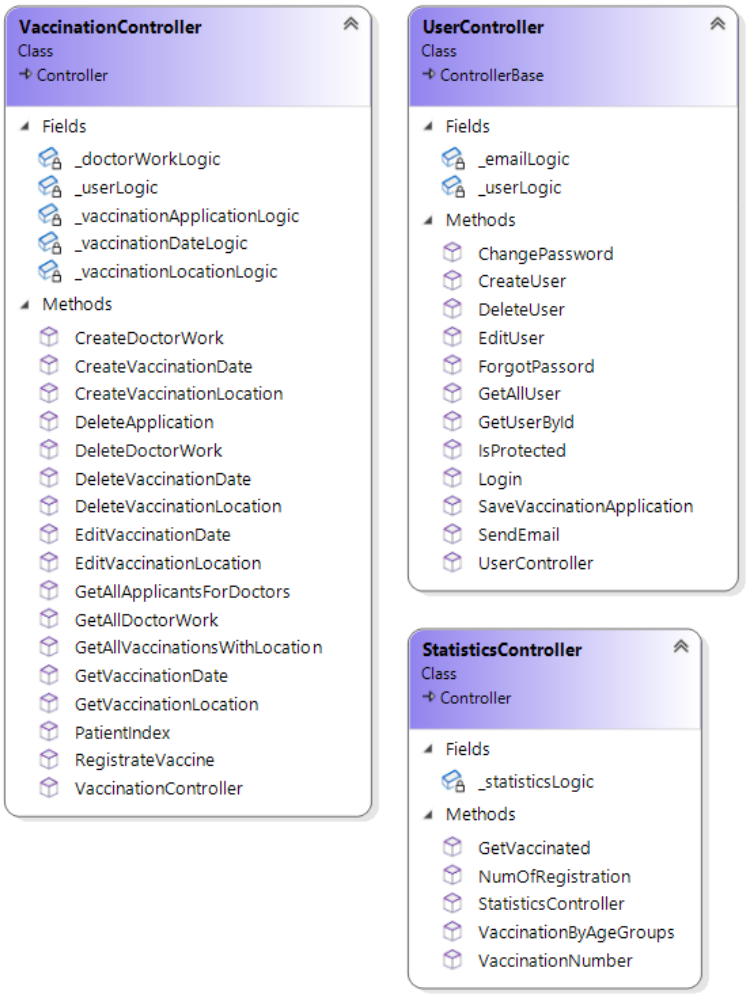


6. ábra: SzakdolgozatWeb Logic réteg osztályai

Controller réteg:

A Backend oldalon ez a réteg felelős a külső rendszerekkel való kommunikációért. Fontos kiemelni, hogy az érkező HTTP hívások esetén először authoriázció történik, annak érdekében, hogy illetéktelenek ne tudják használni az API nyújtotta funkciókat.

- `VaccinationController`: Oltási időpontok, helyszínek lekérdezése, szerkesztése és frissítése. Oltási jelentkezések bejegyzése és törlése. Doktor szerepkörű felhasználókhoz oltási időpont rendelése (`DoctorWork`).
- `UserController`: Bejelentkezés – Autentikáció elvégzése, Felhasználók lekérdése, szerkesztése, oltottsági állapot ellenőrzése, email küldő szolgáltatás.
- `StatictisController`: Az oltási adatok oldalon szereplő diagramok adatainak biztosítása.

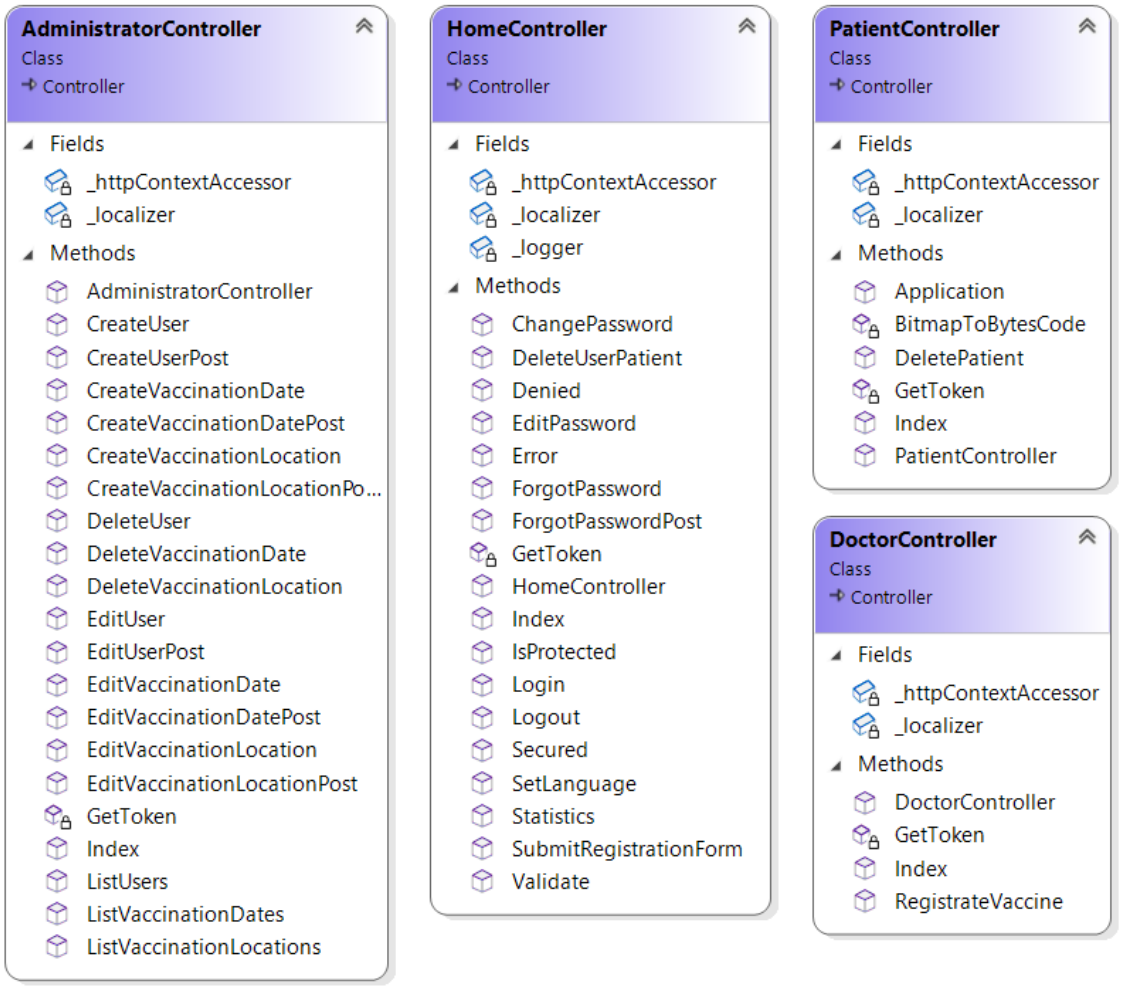


7. ábra: SzakdolgozatAPI Controller réteg osztályai

SzakdolgozatWeb projekt

Ez a projekt Frontend szerepet tölt be, azaz ennek a feladata a Backend (`SzakdolgozatAPI`) felőli érkező adatok megjelenítése, valamint a felhasználó által igényelt kérések továbbítása a Backend felé.

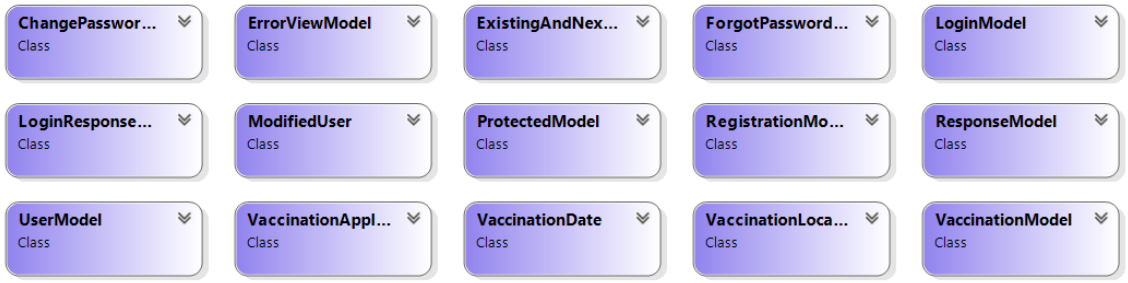
- `HomeController`: Ez a Controller felelős a be-és kijelentkezést, az oltási állapot ellenőrzéséért, a nyelválasztásért, regisztrációért és az elfelejtett jelszó funkcióért.
- `AdministratorController`: Oltási időpontok és helyszínek szerkesztése, illetve az admin és doktor szerepkörű felhasználók szerkesztése.
- `PatientController`: Oltási időpontra fel-és lejelentkezés, fiók törlése
- `DoctorController`: Oltásra jelentkezett páciensek listázása, illetve oltások bejegyzése



8. ábra: SzakdolgozatWeb Controller réteg osztályai

Model réteg

A Frontend oldalon létrehozott modellek egyetlen szerepe a Backend (`SzakdolgozatAPI`) felől érkező adatok feldolgozás és megjelenítése, illetve mezők kitöltése/szerkesztése esetén az adatok visszaküldése.



9. ábra: SzakdolgozatWeb Model réteg

7.2 Jogosultságok ellenőrzése

A webalkalmazás bizonyos funkciói csak, abban az esetben érhetőek el a felhasználók számára, ha már bejelentkeztek, illetve, ha joguk van megtekinteni. A Visual Studio fejlesztői környezeten belül lehetőség van olyan MVC projekt létrehozására, amelyben már implementálva van az authorizáció, illetve az autentikáció, azonban a külön Frontend és Backend miatt számomra ez nem volt választható opció.

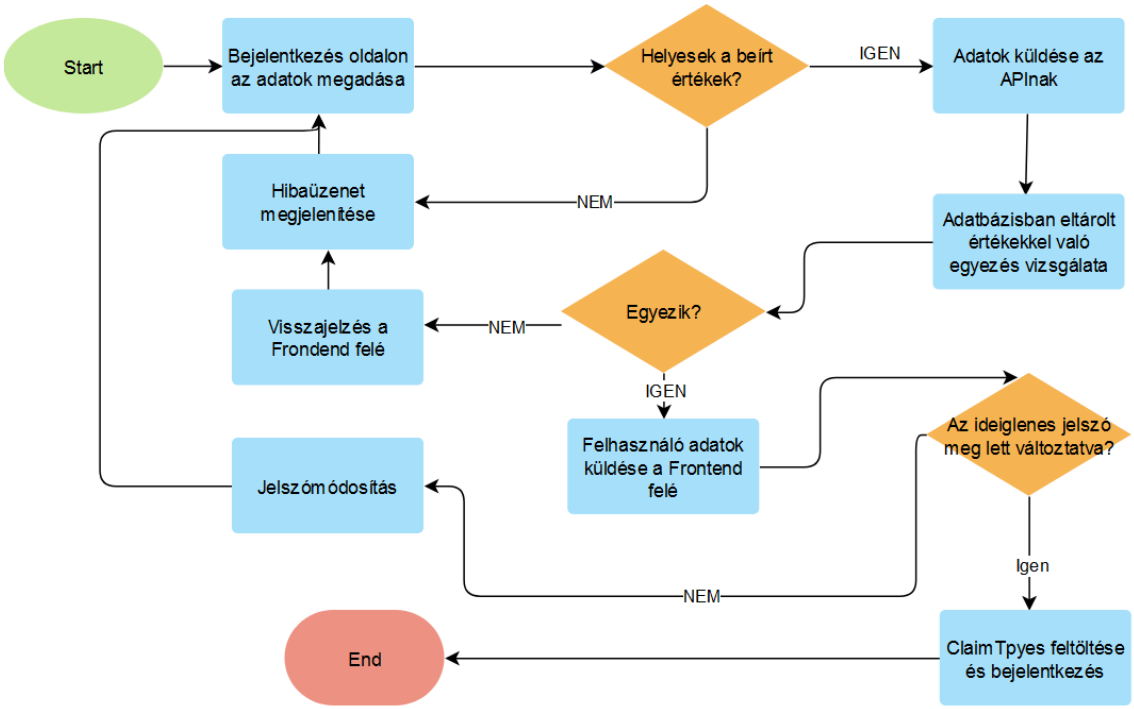
7.2.1 Autentikáció + folyamatábra + folyamatok leírása

Bejelentkezés során a felhasználónak egy űrlapot kell kitöltenie, amelyben meg kell adni az email címét és a jelszavát, majd ki kell tölteni a CAPTCHA tesztet. A mezők után található küldés gomb megnyomásával az űrlapban tárolt értékek egy POST metódus formájában küldésre kerülnek a `HomeControllerValidate` metódusába. (Melléklet: 16.2 Bejelentkezés – Validate metódus)

Ha bármelyik mező üresen marad vagy az email cím mezőbe írt érték nem email formátumú, akkor a felhasználó hibaüzenet fog kapni, benne a probléma leírásával.

Sikeres űrlapkitöltés esetén az email és a jelszó küldésre kerül az API felé, ahol megtörténik az ellenőrzés. Ha a megadott paraméterek megegyeznek az adatbázisban eltárolt értékekkel, akkor a visszatérő JSON objektum tartalmazza a felhasználói fiók fontosabb adatait: név, email, szerepkör és egy token amivel a későbbiekben hitelesíteni

tudja magát az API meghívott metódusaiban. Ha a felhasználó még nem változtatta meg a kezdeti jelszavát, akkor átirányításra kerül egy másik oldalra, ahol azt meg kell változtatnia. Sikeres jelszóváltoztatást követően ismét be kell jelentkezni. A folyamat az alábbi ábrán látható:



10. ábra: Az Autentikáció folyamata

7.2.2 Jelszó titkosítás

Annak érdekében, hogy az adatbázisban eltárolt jelszavak ne legyen olvashatóak titkosításra van szükség. A probléma megoldásához a BCrypt.Net Nuget csomag nyújtotta szolgáltatásokat vettem igénybe. Telepítéséhez a Visual Studio Package Manager Console-on az alábbi parancsot kell lefuttatni: `Install – Package BCrypt.Net – Next`. A használata nagyon egyszerű: a hasítófüggvényen belül paraméterként át kell adni a titkosítandó jelszót. A függvény egy karakterlánccal(string) fog visszatérni. Példa:

```
string passwordHash = BCrypt.Net.BCrypt.HashPassword("Jelszó");
```

Ha ellenőrizni szeretnénk, hogy a felhasználó jó jelszót írt-e be, akkor az alábbi függvényt kell használni (az első paraméter a fejhasználó által beírt jelszó, a másik a pedig az adatbázisban eltárolt kódolt jelszó):

```
bool verified = BCrypt.Net.BCrypt.Verify("beírt jelszó", passwordHash);
```

7.2.3 Authorizáció + folyamatábra + folyamatok leírása

Frontend

Az authorizáció történhet teljes Controller szinten, Action szinten, illetve Razor oldalakon az elágazásokon belül. Ha azt szeretnénk, hogy Controlleren belül csak bejelentkezett felhasználó számára legyen elérhető az adott funkció, akkor `[Authorize]` attribútumot kell elhelyezni a korlátozandó Action neve fölött. Ha Controllerhez tartozó összes Actiont szeretnénk korlátozni, akkor a Controller definiálása fölötti sorban kell elhelyezni az attribútumot. Amennyiben szerepkör szerint szűrésre van szükség, akkor az attribútumot az alábbi módon ki kell kiegészíteni: `[Authorize(Roles = "Szerepkör")]`. Razor nézeteken belül korlátozások az alábbi módon implementálhatók: Elágazásokban belül az alábbi metódust kell meghívni, ami true vagy false értékkel tér vissza: `User.IsInRole("RoleName")`.

Backend – API

Sikeres bejelentkezést követően a felhasználó adatai (`Claimtypes.Name`, `ClaimTypes.Email` és `ClaimTypes.Sid`) a böngészőn belül egy Cookieban kerültek eltárolásra. Ezek közül a `ClaimTypes.Sid` egy olyan tokent tartalmaz, amivel a felhasználó hitelesíteni tudja magát az API hívások során. Ez egy olyan kódolt karakterlánc, ami tartalmazza a felhasználó jogosultságát (szerepkörét), azonban ez csak az API által dekódolható. Használatához a HTTP hívás során a Header részbe az alábbiakat kell beszúrni: `"Authorization", "Bearer " + TokenString`

7.3 Többnyelvűség implementálása

Egy többnyelvű weboldal lehetővé teszi, hogy az általa nyújtott szolgáltatások ne csak egy adott nyelven beszélő személyek számára legyen elérhető, hanem jóval szélesebb körben. A funkció implementálása két fontos részből tevődik össze: globalizációból és lokalizációból.

Globalizáció alatt azt a folyamatot értjük, amikor különböző kultúrák számára (akár földrajzi területek szerint) más nyelven, módon kerülnek megjelenítésre a funkciók, szövegek.

Lokalizáció alatt az a folyamatot értjük, amikor egy globalizált webalkalmazás használata során eldöntésre kerül, hogy milyen nyelvi, kulturális elemek kerüljenek megjelenítésre. Ez történhet manuálisan a felhasználó által (például a navigációs sávban egy legördülő listában való nyelvválasztással) vagy automatikusan az alkalmazás betöltése során: Ekkor a felhasználó lokációja alapján a megfelelő kulturális, nyelvi elemek kerülnek betöltésre. Ha nincsenek megfelelő nyelvi fájlok, akkor az alapértelmezett nyelvi elemek kerülnek betöltésre.

A funkció implementálásához létrehoztam a lokalizált nyelvi forrásfájlokat, amelyeket meghatározott mappastruktúra szerint kellett eltárolnom a Frontend projekten belül. Az elnevezési szabályok az alábbiak:

- View: Resources/Views/[Controller neve]/[View neve].[nyelvi kód].resx
  Példa: Resources/Views/Home/Index.en.resx, bizonyos esetekben lehetőség van az ország szerinti nyelvválasztásra: Resources/Views/Home/Index.en-US.resx
- Modellek: Resources/ViewModels/[Modell neve].[nyelvi kód].resx vagy
  Resources/ViewModels.[Modell neve].[nyelvi kód].resx
  Példa: Resources/ViewModels/LoginViewModel.de.resx
  Resources/ViewModels.LoginViewModel.de.resx
- Controller: Resources/Controllers/[Controller neve].[nyelvi kód].resx vagy
  Resources/Controllers.[Controller neve].[nyelvi kód].resx
  Példa: Resources/Controllers/HomeController.hu.resx vagy
  Resources/Controllers.HomeController.hu.resx

A nyelvi forrásfájlok 2 részből tevődnek össze: Name és Value. A Name kulcs szerepet játszik, tehát ezzel kell hivatkozni a fordított szövegre, ami a Value oszlopban szerepel. Fontos, hogy ugyan ahhoz a Nézethez/ViewModelhez/Controllerhez tartozó forrásfájlok esetén ugyan azok a Name értékek szerepeljenek.

Munkámban az összes nézeten belül implementálva lett a többnyelvűség, ami az alábbi módon történt: A nézetek (cshtml fájl) elején elhelyeztem az alábbi kódot: `@inject Microsoft.AspNetCore.Mvc.Localization.IViewLocalizer Localizer`

Az `IViewLocalizer` szolgáltatás biztosít lokalizált karakterlácot (string) a nézet számára és a `ViewLocalizer` osztály pedig implementálja ezt az interfészt és megkeresi a megfelelő lokalizált nyelvi forrásfájlt.

Ezt követően már csak hivatkozni kell a forrásfájlon belüli sorra az alábbi módon: `@Localizer["Pagetitle"]` Az idézőjel közötti értéket fogja keresni a forrásfájl Name oszlopában és a Value oszlopban szereplő értéket fogja megjeleníteni a felhasználónak, ami akár tartalmazhat HTML kódot is.

7.4 Email küldés

A korábbi fejezetekben már említésre került az email szolgáltatás, amit most részletesebben bemutatok. A szolgáltatás az `SzakdolgozatApi` projekten belül, az `EmailLogic` osztályban került implementálásra. Két metódust tartalmaz: `SendEmailForgotPasswordAndRegistration` és `SendEmailVaccineApplication`, amelyek a keretrendszer több beépített objektumát, metódusait használják. (Melléklet 16.3 fejezet)

```
SendEmailForgotPasswordAndRegistration(string to, bool IsRegistration, string name, string password)
```

Ez a metódus abban az esetben kerül meghívásra, ha egy létező felhasználó elfelejtette a jelszavát és egy új ideiglenes jelszót igényelt vagy új felhasználót hoztak létre és kiküldésre kerül számára egy ideiglenes jelszó. A metódus paraméterei: to = felhasználó email címe, `IsRegistration` = Ha regisztráció, akkor true, különben false, `name` = a felhasználó neve, `password` = ideiglenes jelszó.

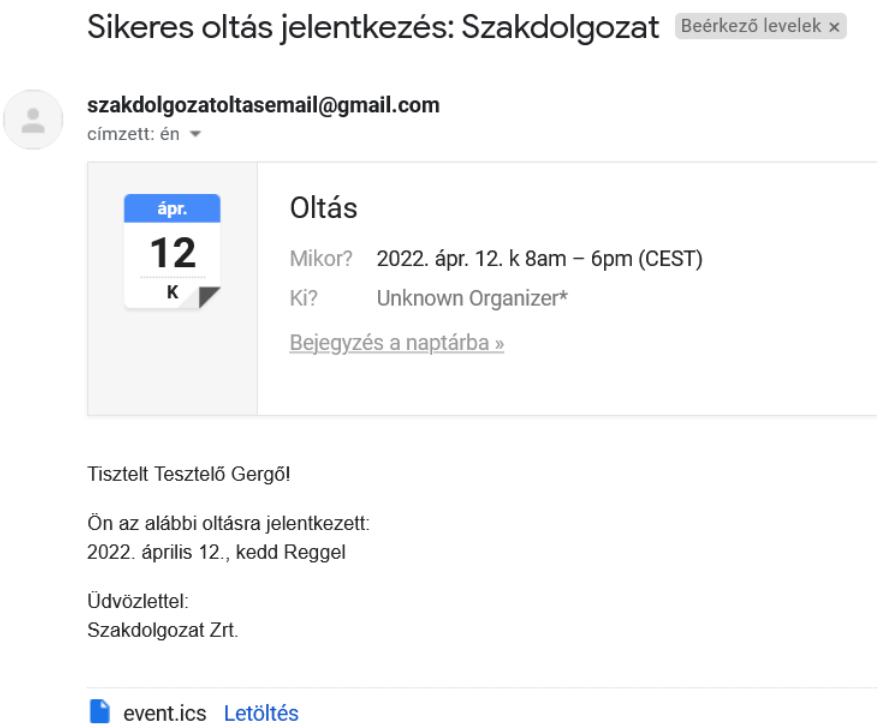
A metódus elején létrehozásra kerül egy `MailMessage` példányt, majd egyes tulajdonságai értéket kapnak: email tárgya, tartalma, címzettje. Ezt követően egy `SmtpClient` kerül példányosításra, majd meg kell adni neki a port számot, a hoszt nevét, illetve a belépési paramétereket. Tekintettel arra, hogy szükség van egy létező email fiókra ezért a munkámhoz létrehoztam egy külön GMAIL-fiókot, annak érdekében, hogy ne a privát email címemet kelljen használnom. Az STMP.Credentials megadását követően az email küldésre kerül.

`SendEmailVaccineApplication(string to, DateTime date, string time, string name)`

Sikeres oltásjelentkezést követően a felhasználó kap egy emailt az oltásról, amiben szerepel mellékletként egy .ics naptárfájl. Ezt a fájl ki tudja importálni a saját naptár alkalmazásába (Outlook, Apple Naptár, Google Naptár). A naptárfájl kiegészül egy emlékeztető jelzéssel, ami 24 órával az oltás előtt jelez a páciensnek.

A metódus paraméterei: `to` = a címzett email címe, `date` = oltás dátuma, `time` = az oltási alkalom `PartOfTheDay` tulajdonságának az értéke, `name` = páciens neve.

A metódus elején létrehozásra kerül egy `CalendarEvent` példány, majd megadásra kerülnek a naptáresemény fontosabb tulajdonságai. Ezt követően egy emlékeztető (Alarm) példány kerül létrehozásra, ami 24 órával a naptáresemény előtt jelez. Legvégül az `MailMessage` példányhoz kerül hozzáfűzésre melléklet formájában. Ezt követően a `MailMessage` tulajdonságai hasonló módon kerülnek beállításra, mint a korábban említett `SendEmailForgotPasswordAndRegistration` metódusban. Végül az email küldésre kerül.



11. ábra: Email a sikeres oltás jelentkezésről

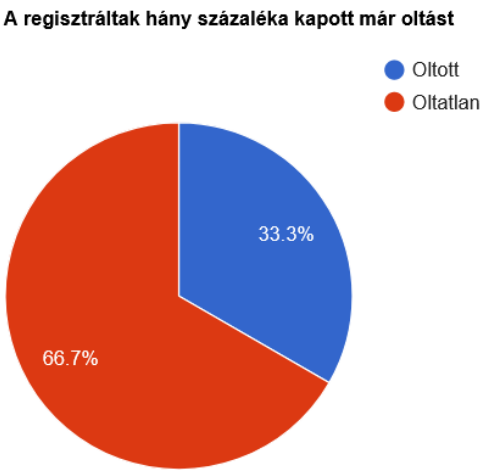
8. Funkcionális tesztelés

Funkció megnevezése	Elvárt működés	Tesztelés időpontja	Eredmény	Tesztelő
Google ReCAPTCHA	Ha nincs bepipálva, akkor sikertelen a bejelentkezés/ regisztráció	2022. 03. 24.	OK	Szabados Gergely
Regisztráció	Email formátum ellenőrzés, adatok beszúrás az adatbázisba, új felhasználó létrehozás	2022. 03. 24.	OK	Szabados Gergely
Automatikus email küldés	Email a sikeres a regisztrációról + benne ideiglenes jelszó	2022. 03. 24.	OK	Szabados Gergely
Bejelentkezés + ideiglenes jelszó	Bejelentkezés + ha még nem változtatták meg az ideiglenes jelszót, akkor átirányít a jelszóváltoztatás oldalra	2022. 03. 24.	OK	Szabados Gergely
Kijelentkezés	Kijelentkezés	2022. 03. 24.	OK	Szabados Gergely
Nyelvváltás	A kiválasztott nyelvre változik minden felirat, szöveg, gomb, diagram	2022. 03. 24.	OK	Szabados Gergely
Szerepkör szerinti funkciók látszódnak	Csak a felhasználó szerepkörének megfelelő funkciók látszódnak	2022. 03. 24.	OK	Szabados Gergely
Páciens				
Oltásjelentkezés	Oltásjelentkezés bejegyzése az adatbázisba + férőhelyek számának csökkentése	2022. 03. 24.	OK	Szabados Gergely
Email a jelentkezésről	Oltásjelentkezést követően automatikus email küldése az időpontról	2022. 03. 24.	OK	Szabados Gergely
Oltás törlése	Törlődik a páciens aktív oltása	2022. 03. 24.	OK	Szabados Gergely
Fiók törlése	Az adatbázisból törlődik a felhasználó + az összes oltási adata + jelentkezése (ha volt)	2022. 03. 24.	OK	Szabados Gergely
Meglévő oltások megjelenítése	A páciens meglévő oltásai listázása	2022. 03. 24.	OK	Szabados Gergely
QR kód + ellenőrző oldal	Ha a páciens védettnek minősül, akkor letöltheti a QR kódot, ami egy védettséget igazoló oldalra vezet, ahol megtekinthetőek a korábbi oltások	2022. 03. 24.	OK	Szabados Gergely

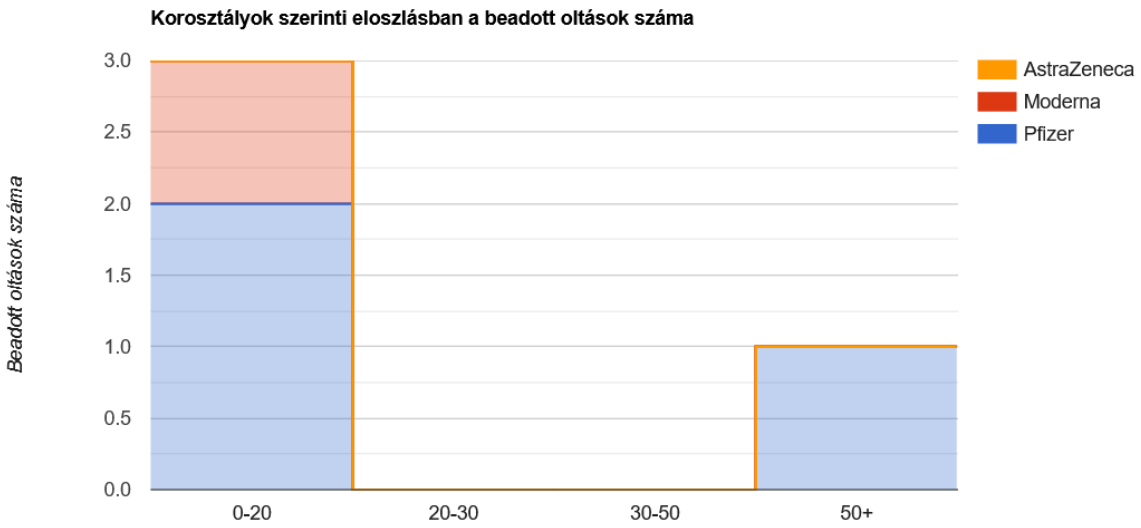
Doktor				
Páciensek listázása	A doktor beosztásával megegyező oltási időpontra jelentkezett páciensek listázása + nevük mellett pedig az oltás bejegyzésére egy gomb	2022. 03. 24.	OK	Szabados Gergely
Admin				
Felhasználók szerkesztés	Admin és Doktor szerepkörű felhasználók listázása + CRUD	2022. 03. 26.	OK	Szabados Gergely
Vakcinahelyszínek szerkesztése	Vakcinahelyszínek listázása + teljes körű szerkesztése (CRUD)	2022. 03. 26.	OK	Szabados Gergely
Vakcinaidőpontok szerkesztése	Vakcinaidőpontok listázása + teljes körű szerkesztése (CRUD)	2022. 03. 26.	OK	Szabados Gergely
Oltási adatok - Diagramok				
Vonaldiagrammon ábrázolni az elmúlt 10 napban a naponta beadott oltások számát		2022. 03. 30.	OK	Szabados Gergely
Kördiagrammon bemutatni, hogy a regisztráltak hány százaléka kapott már oltást		2022. 03. 30.	OK	Szabados Gergely
Diagrammon bemutatni, hogy az adott korosztályokban hány darab oltást adtak be		2022. 03. 30.	OK	Szabados Gergely
Vonaldiagrammon ábrázolni az elmúlt 10 napban az újonnan regisztráltak számát		2022. 03. 30.	OK	Szabados Gergely

7. táblázat: Funkcionális tesztelés

8.1 Oltási statisztikák (Google Charts diagramok) tesztelése

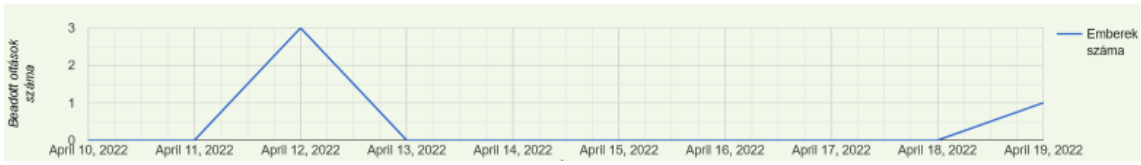


12. ábra: A regisztráltak hány százaléka kapott már oltást



13. ábra: Korosztályok szerinti eloszlásban a beadott oltások száma

14. ábra: Regisztrációk száma az elmúlt 10 napban



15. ábra: Beadott oltások száma az elmúlt 10 napban

9. Továbbfejlesztési lehetőségek

Az elkészült rendszer segítségével drasztikusan csökkenthető egy oltáskampányon belül az adminisztrációs munka, azonban a felhasználói élmény növelése érdekében célszerű a jövőre továbbfejleszteni az alábbi módokon:

- SMS autentikáció

  Bejelentkezés során kétfaktoros hitelesítéssel nagyobb biztonság garantálható, így ki lehet szűrni az illetéktelen próbálkozásokat.

- Mobiltelefonos támogatás:

  Tekintettel arra, hogy a mobiltelefonok széles körben használtak, ezért célszerű lenne a jövőben mobiltelefonokra alkalmazást készíteni, amelyben például lehetne internet nélkül használni digitális oltási igazolvány formájában.

- Látássérülteket segítő felület

  A látássérültek segítése érdekében a webalkalmazás fejleszthető olyan funkciókkal, mint például nagy kontrasztos kinézet, illetve választható betűméret.

- Űrlap a felhasználói visszajelzésekre:

  Véleményem szerint a felhasználói visszajelzések nagyon lényegesek, hiszen ezek alapján tudjuk a végfelhasználók érdekeit szolgálni. Egy űrlap segítségével elmondhatják véleményüket a webalkalmazásról vagy bármilyen egyéb észrevételt elmondhatnak ebben az űrlapban.

- TAJ szám valódiságának ellenőrzése az OEP/NEAK rendszer segítségével

10. Következtetések

Témaválasztás során elsődleges szempont volt számomra, hogy létező problémákra találjak megoldásokat, mint például az automatikus email küldés vagy a nagyobb funkcionalitású oltási időpont kereső. Fontos volt számomra, hogy egy frissen kiadott keretrendszerben készíthessem el a feladatomat annak érdekében, hogy a megszerzett tudás kiegészítse a korábban megszerzett webfejlesztési ismereteimet.

Célkitűzéseim között volt .Net 6.0 keretrendszer fontosabb funkciónak megismerése: authorizáció, authentikáció, szerepkörök szerinti funkciók implementálása, többnyelvűség fejlesztése, illetve a Frontend fejlesztési képességeim fejlesztése. Munkám során az egyik legnehezebb feladatnak az authorizáció implementálása bizonyult, azonban komolyabb internetes kutatást követően rátaláltam a JSON web Tokens-re, melynek dokumentációját elolvasva sikerült áthidalnom a problémát.

Továbbá fontos célkitűzésem volt számomra, hogy jobban felhasználjam az interneten rendelkezésre álló tudást, azaz önállóan keressek és tanuljak új dolgokat. Véleményem szerint a szakdolgozatomban és a magam felé kitűzött céljaimat sikerült elérnem és a munkám elkészítése során olyan rutint és tapasztalatokat szereztem, amivel a későbbiekben hatékonyabban tudom növelni tudásomat, mind a webfejlesztésen belül, mind más technológiákban.

11. Összefoglalás

Szakdolgozatomban bemutatásra került egy olyan rendszer tervezése és fejlesztése, ami járvány esetén megkönnyíti a vakcinabeadás adminisztrációját, illetve annak operatív működését. Elsőként a munkám elején bemutatásra került, hogy a jelenlegi vakcinaregisztrációs rendszernek milyen hiányosságai és hibái vannak, illetve, hogy én ezeket milyen módon oldom meg.

Másodikként az alkalmazott technológiákat és szoftvereket mutattam be, majd ezt követően bemutattam, hogy a rendszernek általános és funkcionális követlényeknek kell megfelelnie, azaz hogyan nézzen ki, illetve, hogy milyen funkciókkal rendelkezzen.

Munkám harmadik része a megvalósításról szól, amelyet az adatbázis bemutatásával kezdtem és ezt követően ismertettem, hogy milyen módon valósítottam meg a követelményekben definiált elvárásokat és funkciókat.

A megvalósítást egy funkcionális teszteléssel zártam, annak érdekében, hogy megbizonyosodjak a rendszer megfelelő működéséről.

12. Summary

In my dissertation, I presented the design and development of a system that facilitates the administration of vaccine administration and its operational operation in the event of an epidemic. At the beginning of my work, I presented the shortcomings and flaws of the current vaccine registration system and how I will solve them.

Secondly, I presented the technologies and softwares I used, and then I showed how the system should meet general and functional requirements, what it should look like and what functions it should have.

The third part of my work is about the implementation, which I started with the presentation of the database and then described how I implemented the requirements and functions defined in the requirements.

I closed the implementation with a functional test to make sure the system was working properly.

13. Ábrajegyzék

14. Táblázatjegyzék

15. Irodalomjegyzék

[1] Barcza, Zs., Formanek-Balku, E., Gyenesei, A., Herzceg, R., Kásler, M., Kiss, Z., Kollár, L., Miseta, A., Molnár, A. G., Müller, C., Pályi, B., Surján, Gy., Surján, O., Vokó, Z., Wittmann, I.: Nationwide effectiveness of five SARS-CoV-2 vaccines in Hungary - The HUN-VE study, (https://www.clinicalmicrobiologyandinfection.com/article/S1198-743X(21)00639-X/fulltext), utoljára megtekintve: 2021.11.25.

[2] COVID-19 Dataset by Our World in Data, (https://github.com/owid/covid-19-data), utoljára megtekintve: 2021.12.05.

[3] COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University, (https://github.com/CSSEGISandData/COVID-19), utoljára megtekintve: 2021.12.05.

[4] Az ASP .NET programozási nyelv, (http://nyelvek.inf.elte.hu/leirasok/ASP.NET/index.php?chapter=1), utoljára megtekintve: 2021.10.02.

[5] Common Language Runtime (CLR) overview, (https://docs.microsoft.com/en-us/dotnet/standard/clr), utoljára megtekintve: 2021.10.23.

[6] Overview of ASP.NET Core MVC, (https://docs.microsoft.com/hu-hu/aspnet/core/mvc/overview?view=aspnetcore-5.0), utoljára megtekintve: 2021.11.15.

[7] Szabó-Resch, M. Zs.: MVC, ASP Intro előadás, (https://users.nik.uni-obuda.hu/prog4/Eloadas_HU/P4HU_EA_08_ASPMVC.pptx), utoljára megtekintve: 2021.11.20.

[8] Freeman, A.: Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC 3, Blazor, and Razor Pages. *Apress Media*, 2020

[9] Anuraj, P.: Integrating Google Charts in ASP.NET Core, (https://dotnetthoughts.net/integrating-google-charts-in-aspnet-core/), utoljára megtekintve: 2021.11.10.

[10] Gadge, S.: Google Charts API Using Database in ASP.Net, (https://www.c-sharpcorner.com/UploadFile/ca9151/google-charts-api-using-database-in-Asp-Net/), utoljára megtekintve: 2021.11.10.

[11] How to schedule a SQL Server backup, (https://solutioncenter.apexsql.com/how-to-create-a-sql-server-scheduled-backup/), utoljára megtekintve: 2021.10.14.

[12] RESTful webes API-tervezés, (https://docs.microsoft.com/hu-hu/azure/architecture/best-practices/api-design), utoljára megtekintve: 2021.10.14.

16. Melléklet

16.1 Idegen kulcs megszorítások

```
ALTER TABLE DoctorWorks
ADD CONSTRAINT FK_DoctorWorks_VaccinationDateId
FOREIGN KEY (VaccinationDateId) REFERENCES VaccinationDates(DateId);

ALTER TABLE RegisteredVaccinations
ADD CONSTRAINT FK_RegisteredVaccinations_PatientId
FOREIGN KEY (PatientId) REFERENCES Users(UserId);

ALTER TABLE VaccinationApplications
ADD CONSTRAINT FK_VaccinationApplications_VaccinationDateId
FOREIGN KEY (VaccinationDateId) REFERENCES VaccinationDates(DateId);

ALTER TABLE VaccinationApplications
ADD CONSTRAINT FK_VaccinationApplications_PatientId
FOREIGN KEY (PatientId) REFERENCES Users(UserId);

ALTER TABLE VaccinationDates
ADD CONSTRAINT FK_VaccinationDates_VaccinationLocationId
FOREIGN KEY (VaccinationLocationId) REFERENCES
VaccinationLocations(LocationId);

ALTER TABLE DoctorWorks
ADD CONSTRAINT FK_DoctorWorks_DoctorId
FOREIGN KEY (DoctorId) REFERENCES Users(UserId);
```

16.2 Bejelentkezés – Validate metódus

```
[ValidateReCaptcha]
        [HttpPost("login")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Validate(LoginModel model)
        {
            if (!ModelState.IsValid)
            {
                TempData["CaptchaNotValid"] = _localizer["CaptchaNotValid"];
                return View("login");
            }

            LoginResponseModel? responseModel = null;

            var json = JsonConvert.SerializeObject(model);
            var data = new StringContent(json, Encoding.UTF8,
"application/json");
            try
            {
                using (HttpClient client = new HttpClient())
                {
                    client.BaseAddress = new Uri("https://localhost:7039/");
                    client.DefaultRequestHeaders.Accept.Clear();
```

```
                    client.DefaultRequestHeaders.Accept.Add(new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"))
;
                    HttpResponseMessage response = await
client.PostAsync("api/User/Login", data);
                    if (response.IsSuccessStatusCode)
                    {
                        var jsonModel = await
response.Content.ReadAsStringAsync();
                        responseModel =
JsonConvert.DeserializeObject<LoginResponseModel>(jsonModel);
                    }
                }

                if (responseModel != null && responseModel.Role != null)
                {
                    if (responseModel.Role == "HasToChangeDefaultPassword")
                    {
                        TempData["Message"] = "You have to change your
password";
                        return RedirectToAction("ChangePassword");
                    }
                    var claims = new List<Claim>();
                    claims.Add(new Claim("UserId",
responseModel.UserId.ToString()));
                    claims.Add(new Claim(ClaimTypes.Name,
responseModel.FullName));
                    claims.Add(new Claim(ClaimTypes.Email, model.Email));
                    claims.Add(new Claim(ClaimTypes.Role,
responseModel.Role));
                    claims.Add(new Claim(ClaimTypes.Sid,
responseModel.Token));
                    var claimsIdentity = new ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme);
                    var claimsPrincipal = new
ClaimsPrincipal(claimsIdentity);
                    await HttpContext.SignInAsync(claimsPrincipal);
                    if (responseModel.Role == "Admin")
                    {
                        return RedirectToAction("Index", "Administrator");
                    }
                    return RedirectToAction("Index", responseModel.Role);
                }
                TempData["Error"] = "Error. Username or Password is not
valid";
                return View("login");
            }
            catch (Exception)
            {
                TempData["Message"] = _localizer["OutOfService"];
                return View("login");
            }
        }
```

16.3 EmailLogic osztály

```
public class EmailLogic : IEmailLogic
    {
        public void SendEmailForgotPasswordAndRegistration(string to, bool
IsRegistration, string name, string password)
        {
            try
            {
                MailMessage message = new MailMessage();
                if (IsRegistration)
                {
                    message.Subject = "Sikeres regisztráció! -
Szakdolgozat";
                    message.Body = "<p>Tisztelt " +name + "!</p><p>Köszönjük
regisztrációját. Mellékelten küldjük a jelszavát, amit a bejelentkezést
követően meg kell változtatnia.<br>Jelszó: " + password +
"<br></p><p>Üdvözlettel:<br>Szakdolgozat Zrt.<br></p>";
                }
                else
                {
                    message.Subject = "Elfelejtett jelszó - Szakdolgozat";
                    message.Body = "<p>Tisztelt " + name + "!</p><p>Ön új
egyszer használatos jelszót kért az elfelejtettet jelszó gomb
segítségével.<br>Jelszó: " + password +
"<br></p><p>Üdvözlettel:<br>Szakdolgozat Zrt.<br></p>";
                }

                SmtpClient smtp = new SmtpClient();
                message.From = new MailAddress("asd@sdf.hu");
                message.To.Add(new MailAddress(to));
                message.IsBodyHtml = true; //to make message body as html
                smtp.Port = 587;
                smtp.Host = "smtp.gmail.com"; //for gmail host
                smtp.EnableSsl = true;
                smtp.UseDefaultCredentials = false;
                smtp.Credentials = new
NetworkCredential("szakdolgozatoltasemail@gmail.com", "Szakdoga123!");
                smtp.DeliveryMethod = SmtpDeliveryMethod.Network;
                smtp.Send(message);
            }
            catch (Exception) { }
        }

        public void SendEmailVaccineApplication(string to, DateTime date,
string time, string name)
        {
            try
            {
                var calendar = new Calendar();

                var icalEvent = new CalendarEvent
                {
                    Summary = "Oltás",
                    Description = "Covid-19 Vakcina",
                    Start = new CalDateTime(date.Year, date.Month, date.Day,
8, 0, 0),
```

```
                    End = new CalDateTime(date.Year, date.Month, date.Day,
18, 0, 0)
                };

                //// Create a reminder 24h before the event
                Alarm reminder = new Alarm();
                reminder.Action = AlarmAction.Display;
                reminder.Trigger = new Trigger(new TimeSpan(-24, 0, 0));
                icalEvent.Alarms.Add(reminder);

                calendar.Events.Add(icalEvent);
                calendar.AddTimeZone(new VTimeZone("Europe/Budapest"));
                var iCalSerializer = new CalendarSerializer();
                var result = iCalSerializer.SerializeToString(calendar);

                var bytesCalendar = Encoding.UTF8.GetBytes(result);
                MemoryStream ms = new MemoryStream(bytesCalendar);
                System.Net.Mail.Attachment attachment = new
System.Net.Mail.Attachment(ms, "event.ics", "text/calendar");

                MailMessage message = new MailMessage();
                SmtpClient smtp = new SmtpClient();
                message.From = new MailAddress("asd@sdf.hu");
                message.To.Add(new MailAddress(to));
                message.Subject = "Sikeres oltás jelentkezés: Szakdolgozat";
                message.IsBodyHtml = true; //to make message body as html
                message.Body = "<p>Tisztelt " + name + "!</p><p>Ön az alábbi
oltásra jelentkezett:<br>" + date.ToString("D",
System.Globalization.CultureInfo.CreateSpecificCulture("hu-HU")) + " " +
time + "<br></p><p>Üdvözlettel:<br>Szakdolgozat Zrt.<br></p>";

                message.Attachments.Add(attachment);
                smtp.Port = 587;
                smtp.Host = "smtp.gmail.com"; //for gmail host
                smtp.EnableSsl = true;
                smtp.UseDefaultCredentials = false;
                smtp.Credentials = new
NetworkCredential("szakdolgozatoltasemail@gmail.com", "Szakdoga123!");
                smtp.DeliveryMethod = SmtpDeliveryMethod.Network;
                smtp.Send(message);
            }
            catch (Exception) { }
        }
    }
```