

**UNIVERZITA J. SELYEHO – SELYE JÁNOS EGYETEM**

**FAKULTA EKONÓMIE A INFORMATIKY –  
GAZDASÁGTUDOMÁNYI ÉS INFORMATIKAI KAR**

**ELEKTRONICKÝ DENNÍK PRE SLEDOVANIE OSOBNÝCH  
AKTIVÍT –**

**ELEKTRONIKUS NAPLÓ SZEMÉLYES AKTIVITÁSOK  
NYOMONKÖVETÉSÉRE**

**Bakalárska práca – Szakdolgozat**

**Ollé Kevin**

**2021**

**UNIVERZITA J. SELYEHO – SELYE JÁNOS EGYETEM**

**FAKULTA EKONÓMIE A INFORMATIKY –  
GAZDASÁGTUDOMÁNYI ÉS INFORMATIKAI KAR**

**ELEKTRONICKÝ DENNÍK PRE SLEDOVANIE OSOBNÝCH  
AKTIVÍT –**

**ELEKTRONIKUS NAPLÓ SZEMÉLYES AKTIVITÁSOK  
NYOMONKÖVETÉSÉRE**

**Bakalárska práca – Szakdolgozat**

Študijný program:	Aplikovaná informatika
Tanulmányi program:	Alkalmazott informatika
Názov študijného odboru:	18. informatika
Tanulmányi szak megnevezése:	18. informatika
Vedúci záverečnej práce:	PaedDr. Krisztina Czakóová, PhD.
Témavezető:	PaedDr. Czakóová Krisztina, PhD.
Školiace pracovisko:	Katedra informatiky
Tanszék megnevezése:	Informatikai Tanszék

**Ollé Kevin**

**Komárno, 2021**



Univerzita J. Selyeho  
Fakulta ekonómie a informatiky

---

## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Kevin Ollé  
**Študijný program:** Aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)  
**Študijný odbor:** 18. - informatika  
**Typ záverečnej práce:** Bakalárska práca  
**Jazyk záverečnej práce:** maďarský  
**Sekundárny jazyk:** slovenský

**Téma:** Elektronikus napló személyes aktivitások nyomkövetésére

**Anotácia:** A szerző feladata egy olyan e-napló kifejlesztése, mely lehetőséget nyújt a felhasználónak személyes tevékenységeket beiktatni naplószerűen, és azokat adott szempontok szerint szűrni, rendezni, kiemelni, csoportosítani. Legyen lehetőség figyelmeztető üzenetek beállítására is adott tevékenységekhez vagy határidőkhöz kötve. A felhasználói felület legyen áttekinthető, rendezett, grafikailag támogatott.

### Kľúčové

**slová:** Blazor, elektronikus napló, adatbázis kezelés, dinamikus weblapok

**Vedúci:** PaedDr. Krisztina Czakóová, PhD.

**Katedra:** KINF - Katedra informatiky

**Vedúci katedry:** Ing. Ondrej Takáč, PhD.

**Dátum zadania:** 29.05.2020

**Dátum schválenia:** 19.07.2020

Ing. Ondrej Takáč, PhD.  
vedúci katedry

## Čestné vyhlásenie – Becsületbeli nyilatkozat

Čestne vyhlasujem, že diplomovú prácu som vypracoval samostatne, s použitím uvedenej literatúry a pod odborným vedením vedúceho práce.

Kijelentem, hogy a szakdolgozatot önállóan dolgoztam ki a feltüntetett szakirodalom felhasználásával és témavezetőm szakmai irányításával.

Komárno, 2021

.....

vlastnoručný podpis – sajátkezű aláírás

## **Köszönetnyilvánítás**

Szeretnék köszönetet mondani témavezetőmnek, PaedDr. Czakóová Krisztina, PhD. tanárnőnek, aki hasznos tanácsaival és észrevételeivel segítséget nyújtott a szakdolgozatom elkészítésében.

Köszönöm a NETWORK 24 s. r. o. csapatának, hogy a diákmunkám során sikerült megismernem a Blazor Framework –öt, amiben ez a munka készült.

## **Abstrakt**

Cieľom bakalárskej práce bolo vyvinúť elektronický denník, v ktorom môže užívateľ presne sledovať svoje denné a týždenné udalosti. Je možnosť pridať udalosti alebo zoradiť udalosti podľa ich priority. Práca tiež poukazuje na fungovanie blazorského rámca. Práca obsahuje 4 kapitoly a prílohu CD. Prvá kapitola poskytuje pohľad do sveta elektronických denníkov. Druhá kapitola pojednáva o použitých technológiách. Nasledujúca kapitola pojednávajú o fungovaní jednotlivých stránok webstránky a o problémoch a riešeniach, ktoré sa vyskytli počas vývoja. V poslednej kapitole sumarizujeme skúsenosti získané počas vývoja, ďalšie návrhy rozvoja a diskutujeme aj o zverejnení projektu na internete.

**Kľúčové slová:** Blazor, elektronický denník, správa databáz, dynamické webové stránky

## **Absztrakt**

A szakdolgozat célja egy elektronikus napló kifejlesztése volt, amelyben a felhasználó pontosan tudja követni a napi, illetve heti eseményeit. Eseményeket tud hozzáadni, illetve az eseményeket prioritásuk alapján tudja rendezni. A munka egyben rámutat a Blazor Framework működésére is. A munka 4 fejezetet tartalmaz, illetve egy CD mellékletet. Az első fejezet betekintést ad az elektronikus naplók világába. A második fejezet felhasznált technológiákat tárgyalja. A további fejezet pedig az egyes oldalak működését és a fejlesztés során felmerült problémákat és megoldásokat taglalja. Az utolsó fejezetben összegezzük a fejlesztés során felmerült tapasztalatot, tovább fejlesztési javaslatokat, illetve tárgyalva van a projekt interneten való publikálása is.

**Kulcsszavak:** Blazor, elektronikus napló, adatbázis kezelés, dinamikus weblapok

## **Abstract**

The aim of the thesis was to develop an electronic diary in which the user can accurately keep track of his daily and weekly events. You can add events or sort events by their priority. The work also points to the operation of the Blazor Framework. The work contains of 4 chapters and a CD attachment. The first chapter provides an insight into the world of electronic diaries. The second chapter discusses about the used technologies. The following chapter discuss about the operation of each page of site and the problems and solutions encountered during development. In the last chapter we summarize the experience gained during the development, further development suggestions, and the publication of the project on the Internet is also import topic.

**Keywords:** Blazor, electronic diary, database management, dynamic web pages



# Tartalomjegyzék

<b>Bevezetés .....</b>	<b>8</b>
<b>1 Elektronikus naplók lehetőségei.....</b>	<b>9</b>
<b>2 Munka célja.....</b>	<b>11</b>
<b>3 Elektronikus napló személyes aktivitások nyomonkövetésére .....</b>	<b>12</b>
3.1 Felhasznált technológiák .....	12
3.2 A napló adatbázis felépítése .....	14
3.3 Adatbázis kezelés, e-mail küldés és Back-end .....	16
3.3.1 Shared mappa, avagy Blazor és a Razor komponensesek működése, oldalak megjelenése ...	23
3.4 Kezdő oldal .....	26
3.5 Heti nézet .....	32
3.6 Havi nézet.....	34
3.7 Események listája.....	41
3.8 Új esemény hozzáadása.....	51
3.9 Prioritások hozzáadása és listája .....	52
3.10 Felhasználó adatainak beállítása .....	55
<b>4 Eredmények .....</b>	<b>59</b>
4.1 Problémák a fejlesztés során .....	59
4.2 Projekt publikálása az interneten.....	60
4.3 Továbbfejlesztési lehetőségek.....	60
<b>Resumé.....</b>	<b>62</b>
<b>Szakirodalom .....</b>	<b>66</b>
<b>Mellékletek .....</b>	<b>67</b>

## Bevezetés

A mai rohanó világban nagyon fontos az idő menedzselése, a nap minden percét felhasználni minél hatékonyabban. A mai okos telefonok könnyedén kapcsolódnak az internethez, ahol megannyi kalendár applikációt elérnek a felhasználók. Legismertebb ilyen applikációk közé sorolják a Google calendart, Asanat, Thunderbirdöt és megannyi más cég által fejlesztett megoldást.

Nagyon sok ember használ a mai világban valamilyen elektronikus naplót vagy kalendárt. Ezeket az applikációkat főleg a gazdasági életben alkalmazzák előszeretettel, ahol fontos nyomon követni az egyes tárgyalásokat. De használják ilyen alkalmazásokat akár az iparban is, ahol a segítségükkel megtudják tervezni a jövőbeni termelést vagy a termékeik szállítását. Használják elektronikus naplókat továbbá az informatikában is, főleg a jelen helyzetben, amikor a fejlesztő csapatok otthonról kényszerülnek dolgozni és valahogy egyeztetniük kell a csapattal.

A jelenlegi piacon nagyon sok elektronikus kalendárból vagy naplóból választhatnak az emberek. Nagyon sok alkalmazás van, ami egy másikat másol funkcionalitásban, de vannak, amik csak kinézetben különböznek egy másiktól, viszont vannak olyan alkalmazások is, amik mernek újítani. Minden kalandárnak vagy naplónak tartalmaznia kell az alap funkciókat, amik nélkül elképzelhetetlen lenne az ilyen applikációk működése.

Az ilyen funkciók a következők:

1. Események tarolása, menedzselése.
2. Események megjelenítése grafikus felületen, felhasználó barát módon.
3. Prioritási szintek rendelése az eseményekhez, amik alapján a felhasználót értesíti a rendszer a közelgő eseményekről.

A piacon levő elektronikus kalendárok és naplók nagy többsége csak ezeket a fő funkciókat tartalmazza, viszont vannak olyan applikációk, amik a működésükbe belefűzik a mesterséges intelligenciát is. Ezekben az applikációkban tulajdonképpen a mesterséges intelligencia figyeli az egyes tevékenységeinket és azok folyamatos ismétlése után megtanulja azt, hogy mikor mit csinálunk és a jövőben automatikusan beírja az eseményeket a naptárba anélkül, hogy a felhasználónak akár kattintania kellene.

# 1 Elektronikus naplók lehetőségei

Napjaink informatikai megoldásai lehetővé teszik, hogy a fejlesztők szabadra engedjék a képzeleteiket és teljesen új, hatékony rendszereket hozzanak létre. Ezek a rendszerek többnyire webes applikációk. Nagyon sok applikáció csak kinézetben tér el társaitól, viszont a sok applikáció közt vannak nagyon innovatív megoldással megvalósított kalendár applikációk. Ilyen például a *Google Calendár*, melyet ma már bármilyen eszközről elérhetjük és nagyon hatékonyan és gyors működést biztosít a felhasználó számára.



**1. ábra:** Google calendar logó; Forrás:

[https://hu.wikipedia.org/wiki/Google\\_Calendar#/media/F%C3%A1jl:Google\\_Calendar\\_icon\\_\(2020\).svg](https://hu.wikipedia.org/wiki/Google_Calendar#/media/F%C3%A1jl:Google_Calendar_icon_(2020).svg)

Az *Asana* [1] is egy nagyon innovatív rendszer. Főleg szoftver fejlesztő informatikus számára lett kifejlesztve. Nagyon hatékony grafikus felületet biztosít, ahol akár a fejlesztők egy SCRUM táblát is ki tudnak maguknak alakítani. A SCRUM [2] egy fejlesztési modell, amiben a megrendelő is a csapat részese, gyakoriak a köztes szállítások működő funkcionalitással. A köztes állapotokat folyamatosan validálják a megrendelő alapján. Átlátható tervezés és modularizáció jellemző a SCRUM-ra. Gyakoriak a meetingek, amelyeken mindenki véleményét és haladását meghallhatják, a legkisebb funkció működésére is odafigyelnek. A főbb szerepkörök a SCRUM fejlesztési modellben:

1. Scrum Master – felügyeli a szoftverfejlesztési folyamatokat.
2. Product Owner – főleg a projektbeni döntéshozók szerepét képviseli.
3. Team – nagyjából 7 fejlesztőből áll és lefedi a teljes munkafolyamatot a munkájuk.

Az *Asana* továbbá, különböző listák segítségével személyre szabott listákat tudnak létrehozni, például az új funkciók soron követésére vagy bugok és hibák javítását is tudják ezek alapján követni.



**2. ábra:** Asana logó; Forrás: [https://en.wikipedia.org/wiki/Asana\\_\(software\)#/media/File:Asana\\_logo.svg](https://en.wikipedia.org/wiki/Asana_(software)#/media/File:Asana_logo.svg)

Számunkra a projekt megtervezése során a legfontosabb szempont a minél hatékonyabb kezelhetőség és átláthatóság volt a cél. Alapkövetelménynek számított, hogy a projektet (webes applikációt) a felhasználó bárholonnan bármilyen eszközről, amin van egy webböngésző képes legyen elérni. A projektet lényegében a *Google Calendár* inspirálta, annak letisztult dizájnja és funkcionalitása. A legfontosabb funkció a minél átláthatóbb megjelenítés volt, külön napi, heti, hónapos nézet megalkotása. A legfontosabb eltérés a többi piacon levő applikációval szemben talán a felhasznált technika. Lényegében a projekt a kevés *Blazor Frameworkben* írt elektronikus naplók egyike, mely rámutat a *Blazor Framework* alaptulajdonságaira.

## 2 Munka célja

Munkánk célja egy webes applikáció – elektronikus napló elkészítése, amely a betáplált tevékenységeinket összegzi heti vagy havi nézetben, ránézésre színekkel láthatóvá teszi az egyes tevékenység típusokat, azok pontos időpontját. Bár van számos webes applikáció a tevékenységek naplózására, de nincs tudomásunk olyan elektronikus naplóról, ami kimondottan *Blazor Framewrokre* épülne.

Kezdeti cél volt, hogy lokális vagy webes applikáció készüljön. Végül egy webes applikáció készült, aminek az adatait egy *MySQL* adatbázis tárolja.

Legfontosabb cél a fejlesztés során az volt, hogy a felhasználó áttekinthető grafikus felületen tudja nyomon követni az elment eseményeit. A grafikus felület megalkotására *Bootstrapet* és tiszta *CSS-t* használtunk.

Továbbá fontos volt számunkra, hogy a felhasználó egyszerűen hozzá tudjon adni újabb tevékenységet a meglévőkhöz, azokat különböző szempontok szerint szűrni, ezzel is bővítve a felhasználó lehetőségeit a tevékenységei menedzselésében. Maguk az eseményeket bármelyik attribútumuk alapján képes legyen a felhasználó szűrni, illetve növekvő vagy csökkenő sorrendbe állítani azokat. Minden új eseménynél a felhasználó megadhatja az esemény prioritási szintjét, ami a grafikus felületen színek segítségével figyelmezteti a felhasználót, továbbá a felhasználó megadhatja az esemény kezdeti idejét és befejezését, és leírást fűzhet az új eseményhez. A hozzáadott tevékenységet első lépésben egy dátum ellenőrzés után majd foglalás alapján lehet hozzáadni a rendszerhez. A rendszer először ellenőrzi, hogy a menteni kívánt esemény kezdeti dátuma nincs e múltban, majd ha ez a feltétel igaz, akkor a rendszer ellenőrzi, hogy a menteni kívánt intervallumra nincs e már esemény foglalva. Itt, ha intervallumütközés van, akkor a felhasználó döntheti el, hogy át állítja az időt vagy menti az időütközés ellenére is az új eseményt.

Kifejlesztésre került a projekt összes kezdeti célja. Grafikus felületen tudja a felhasználó követni az aznapi, a heti és a havi eseményeit. A rendszerben tud menteni új eseményeket, illetve a már meglévőket tudja módosítani, keresni köztük és rendezni növekvő vagy csökkenő sorrendbe. A felhasználó saját felhasználási ötletei alapján tud hozzáadni új prioritási szinteket, amikhez színeket is tud hozzárendelni. A grafikus felületen a prioritások színei jelzik a felhasználó számára, hogy az egyes események mennyire fontosak a számára.

Maga a projekt széles körben felhasználható, jelen állapotában csak egy felhasználót képes kezelni, akinek továbbá minden reggel elküldi e-mailben az aktuális eseményeket.

### 3 Elektronikus napló személyes aktivitások nyomonkövetésére

Az egész projekt elején az alapprobléma az volt, hogy egy lokális, az adott gépen futtatható vagy webes applikációt szeretnénk-e létrehozni. Végül a webes applikáció mellett döntöttünk. Köszönhetően a szakmai gyakorlatnak az egyetemen sikerült nagyon sok programozási technikát megismerni és elsajátítani, mint a webes applikációk. Diákmunka során sikerül megismerkedni a *Blazor Frameworkkel*, amit a Microsoft alatt működő *.NET Foundation* szervezet fejleszt. Tehát a fejlesztői környezet adott volt, adatok tárolására MySQL adatbázist használtunk, ahonnan az adatokat különböző lekérdezések segítségével tudtunk visszakérni.

Mindezek után szükséges volt pár fontos dolgot lefektetni. Ezek a következők voltak:

1. Áttekinthető design úgy mobil eszközökön, mint személyi számítógépeken.
2. Gyors és egyszerű kezelés minden felhasználó számára.
3. Hasznos funkcionalitás.
4. Események egyszerű nyomon követése.
5. Események között lehessen szűrni, keresni, illetve valamilyen szempont szerint csoportosítani.
6. Eseményekhez lehessen figyelmeztető üzeneteket beállítani.
7. Havi, heti nézet implementálása.
8. Színek használata, amivel a felhasználó be tudja állítani az esemény prioritását

#### 3.1 Felhasznált technológiák

##### Blazor

A Blazor a széles körben elterjedt *ASP.NET* fejlesztői framework része. Kiegészíti a *.NET* fejlesztői platformot eszközökkel és függvény könyvtárakkal ahhoz, hogy minél jobb webes applikációkat tudjanak a fejlesztők készíteni.

Interaktív *web UI* készítését teszi lehetővé *C#* nyelv segítségével anélkül, hogy *JavaScriptet* kéne használni. Egy Blazor app (applikáció) újra felhasználható web UI komponensekből épül fel, amelyek implementálására *C#-ot*, *HTML-t*, és *CSS-t* alkalmaz. A kliens és a szerver kód is *C#-ban* van írva azért, hogy egyszerűen lehessen kódot és függvény könyvtárakat megosztani projektek között.

A Blazor képes a kliens oldali *C#* kódot direkt módon a böngészőben futtatni, felhasználva a *WebAssemblyt*. Alternatív megoldásképpen képes a kliens oldali *C#* kódot

szerveren futtatni. Kliens felhasználó felület történéseit, eseményeit visszaküldi a szervernek a *SignalR* valós idejű üzenetküldési technológiát alkalmazva. Ha az esemény végrehajtódott, akkor a szükséges felhasználói felület változásokat visszaküldi és összesíti a *DOM* elemekkel. A *DOM* (Document Object Model) egy programozási interfész, ami reprezentálja az összes elemét egy HTML oldalnak, mint *nod*-okat egy fa struktúrába rendezve. *DOM*-ot használva az elemeket tudjuk frissíteni, törölni vagy új elemeket tudunk hozzáadni az oldalhoz.

A Blazor lehetővé teszi kódok és függvény könyvtárak megosztását, köszönhetően a *.NET Standard*nak. Lehetővé teszi ugyanazon kód vagy függvény könyvtár használatát szerver oldalon, böngészőben vagy bárhol, ahol *.NET* kódot szeretnénk írni vagy futtatni.

A Blazor lehetővé teszi továbbá az egyszerű JavaScript együttműködést. C# kódból egyszerűen tudunk hívni JavaScript *API*-kat és könyvtárakat. Egyszerűen használható a robosztus JavaScript ökoszisztéma a kliens oldali felhasználó felületen, miközben C# programozási logikáját alkalmazzuk.

*Visual Studio* és *Visual Studio Code* a Microsoft által fejlesztett két integrált fejlesztői környezet lehetővé teszi a *Blazor*-ben való fejlesztést Windows, Linux vagy akár macOS operációs rendszerek alatt. Ha valaki mégis más fejlesztői eszközt alkalmaz, akkor annak ott vannak a *.NET command-line tools*-ok, amik lehetővé teszik a fejlesztést más editorokban.

A Blazor legnagyobb előnye a nyílt forrású ingyenes fejlesztői lehetőség. Része a nyílt forrású *.NET* platformnak, ami mögött egy erős közösség áll 3 700 céggel és több mint 60 000 programozó személyében. [3]

## MySQL

A legelterjedtebb nyílt forráskódú SQL adatbázis menedzsment rendszer, amit az Oracle Corporation fejleszt, forgalmaz és támogat. Egy adatbázis nem más, mint adatok megfelelő struktúra alapján történő tárolása. Adatbázisban tárolt adat lehet egy egyszerű bevásárló lista, de lehet fényképek gyűjteménye vagy egy vállalati hálózat megszámlálhatatlan adatja. Ahhoz, hogy hozzáadni, elérni vagy dolgozni tudjunk adatokkal, amiket adatbázisban tárolunk, ahhoz szükségünk van egy adatbázis menedzselő rendszere, mint a MySQL szerver. A számítógépek nagyon gyorsan és hatékonyan tudják az adatokat feldolgozni, így egy adatbázis menedzselő rendszer központi szerepet játszhat, mint önálló segédprogram, de lehet része más applikációnak is.

Az MySQL egy nyílt forráskódú szoftver, amit bárki tud fejleszteni vagy egyes funkcióit megváltoztatni. Bárki letudja tölteni a MySQL szoftvert az internetről anélkül, hogy fizetnie

kellene érte. Ha szeretnénk, bármikor tudjuk tanulmányozni a forráskódját, és ha szükségünk van rá, akkor azt a saját igényeink szerint tudjuk módosítani.

A *MySQL Database Server* (adatbázis szerver) nagyon gyors, skálázható és egyszerű használatot biztosít a fejlesztők, illetve az átlag felhasználót számára is. Könnyedén fut bármilyen applikáció vagy web szerver mellet bármilyen számítógépen. Ha esetleg dedikálni szeretnénk egy eszközt a MySQL szerver számára, akkor személyre szabhatjuk az eszköz memóriáját, CPU használatát és I/O kapacitását az igényeink alapján. Mára már a konstans fejlesztésnek köszönhetően gazdag funkcionalitást biztosít mind szerver és mind kliens oldalon.

Ma már nagyon sok MySQL szoftvert érhetünk el a piacon. A *MySQL Server*-nek nagyon sok praktikus funkciója van, amik a közösség segítségével lettek kifejlesztve. Ennek hála nagyon sok programozási nyelv támogatja a *MySQL Database Server*-t. [4]

### 3.2 A napló adatbázis felépítése

A naplónk MySQL adatbázist használ, amit a fejlesztés során a WAMP program által biztosított lokális megoldás szolgált ki. A napló elkészítése után a *freemysqlhosting.net* által biztosított ingyenesen használható adatbázis szolgáltatást alkalmaztuk, hogy a napló képes legyen működni az interneten.

Az adatbázis egy *calendar* nevű sémában három darab táblát tartalmaz: a **calendar**, **user** és **priority** táblát.

A **calendar** tábla az események elmentéséért felelős. A tábla 6 darab oszlopból áll. Az *ID int* típusú adatot képes fogadni, egyben ez a tábla elsődleges kulcsa, emellett nem fogad nulla értéket és adatok hozzáadása után automatikusan inkrementálódik az értéke, ezzel megszámozva sorba az eseményeket. A *Date* oszlop dátumot képes fogadni, a *TimeFROM* és *TimeTO TIME* típust. A *Date* jelzi, hogy az esemény milyen dátumhoz van rendelve, a *TimeFROM* és a *TimeTO* pedig az esemény kezdetét és végét menti. Az *Event VARCHAR* típusú oszlop, mely az esemény nevét menti. A *Description* szintén *VARCHAR* típusú oszlop, mely az eseményhez tartozó leírást menti. A *Priority* ismét *VARCHAR* típusú oszlop, mely az eseményhez tartozó prioritási szintet menti. Ennek a táblának a kialakítása a következő:

```
CREATE TABLE `sys`.`calendar` (  
  `ID` INT NOT NULL AUTO_INCREMENT,  
  `Date` DATETIME NOT NULL,  
  `TimeFROM` TIME NOT NULL,
```



```
`TimeTO` TIME NOT NULL,
`Event` VARCHAR(255) NOT NULL,
`Priority` VARCHAR(255) NOT NULL,
`Description` VARCHAR(255) NOT NULL, PRIMARY KEY (`ID`));
```

Az **user** tábla 5 darab oszlopból áll, itt is megtalálható az *ID* nevű oszlop, ami ismét az elsődleges kulcs szerepét tölti be és minden mentés után az értéke inkrementálódik. A *Firstname*, *Lastname*, *Email* oszlopok *VARCHAR* típusúak és a felhasználó vezetéknévét, keresztnévét és e-mail címét mentik. A *Birthday* oszlop a felhasználó születési dátumát menti *DATETIME* típusba. Ennek a táblának a kialakítása a következő:

```
CREATE TABLE `sys`.`user` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Firstname` VARCHAR(255) NOT NULL,
  `Lastname` VARCHAR(255) NOT NULL,
  `Email` VARCHAR(255) NOT NULL,
  `Birthday` DATETIME NOT NULL, PRIMARY KEY (`ID`));
```

A **priority** tábla 3 oszlopot tartalmaz. Az *ID* oszlop ismét ugyan azt a szerepet tölti be, mint az első kettő tábla esetében. A másik két oszlop a *Priority* és *Color*, mindkettő *VARCHAR* típust ment. A *Priority* oszlop menti a prioritás nevét, a *Color* pedig hexadecimális értékben menti a prioritás színét, ezzel jelezve annak fontosságát. Ennek a táblának a kialakítása a következő:

```
CREATE TABLE `sys`.`priority` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Priority` VARCHAR(255) NOT NULL,
  `Color` VARCHAR(255) NOT NULL, PRIMARY KEY (`ID`));
```

(Murach, 2019)

calendar		
ID	int AI	PK
Date	datetime	
TimeFROM	time	
TimeTO	time	
Event	varchar(255)	
Priority	varchar(255)	
Description	varchar(255)	

user		
ID	int AI	PK
Firstname	varchar(255)	
Lastname	varchar(255)	
Email	varchar(255)	
Birthday	datetime	

priority		
ID	int AI	PK
Priority	varchar(255)	
Color	varchar(255)	

3. ábra: Adatbázis táblák grafikusán ábrázolva; Forrás: saját kép

### 3.3 Adatbázis kezelés, e-mail küldés és Back-end

A projekt (napló) felépítése az OOP (objektum orientált programozás) elvét követi. A metódusok többsége egy *C#* osztályban van megírva, amiből aztán objektumok származtatásával oldottuk meg a változók és metódusok használatát. A *DataBase.cs* *C#* osztályt további két alosztályra osztottuk: a *DataBase* és *DataBaseServices* nevű osztályokra. A *DataBase* osztály tartalmazza a változók *get*; *set*; megvalósításait. Ahhoz, hogy a Blazorben használni lehessen a *DataBase* osztályban implementált metódusokat és változókat, ahhoz használni kellett a *CalendarBC.Data* névteret, illetve a *Startup.cs* fájlban a *ConfigureServices* metódushoz hozzá kellett adni a következő sort:

```
Services.AddSingleton<DataBaseServices>();
```

A *DataBase* publikus osztály tartalmazza a változók automatikusan implementált tulajdonságait:

```
public class DataBase
{
    public int ID { get; set; }
    public DateTime Date { get; set; } = DateTime.Now;
    public TimeSpan TimeTO { get; set; }
    public TimeSpan TimeFROM { get; set; }
    public string Event { get; set; }
    public string PriorityE { get; set; }
    public string Description { get; set; }
    public string Priority { get; set; }
    public string CPriority { get; set; }
    public string ISmail { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public DateTime BirthDay { get; set; } = DateTime.UtcNow;
}
```

(Sharp, 2018)

A *DataBaseServices* osztály tartalmazza a projektben használt metódusok nagy többségét. Ebben az osztályban vannak megvalósítva a MySQL adatbázishoz tartozó adat lekérések, adat frissítések vagy az adatok törlése. Az itt definiált metódusok a *DataBase* osztályban definiált

adatmezőkön végeznek műveleteket. Ahhoz, hogy MySQL adatbázissal kapcsolatos műveleteket tudjunk végezni, ahhoz először is telepíteni kellett a *MySQL.Data* nevű *NuGet* csomagot, amit az Oracle adott ki.

A projekt Back-endje tartalmaz még két e-mail küldésre szolgáló szolgáltatást, amik az *IHostedService* interfészt valósítja meg. Az interfész meghatározza a metódusokat az objektum számára, amit a gazdagép menedzsel. Az interfész megvalósításához két metódust kell használni: a *StartAsync()* és *StopAsync()* nevű metódust. Az előbbi akkor indul el, amikor az applikációt futtató gazdagép kész elindítani a szolgáltatást, az utóbbi pedig akkor fut le, amikor az applikáció minden hiba nélkül leáll. [5]

Ahhoz, hogy az e-mail küldő szolgáltatások futni tudjanak, meg kell hívni őket a *Startup.cs* fájlban:

```
services.AddSingleton<IHostedService, MorningEMAIL>();  
services.AddSingleton<IHostedService, EMAILService>();
```

Mindkét szolgáltatás egy *C#* osztályban van implementálva, ami az *IHostService* nevű interfészt valósítja meg. A *MorningEMAIL* osztály három darab *C# Taskot* tartalmaz. Az első a *StartAsync()*, amiben elindul a *Send\_MAIL()* metódus, majd a függvény visszatérési értéke jelzi a szerver számára, hogy a művelet sikeresen lefutott:

```
public Task StartAsync(CancellationToken cancellationToken)  
{  
    Task.Run(Send_MAIL, cancellationToken);  
    return Task.CompletedTask;  
}
```

Ezt a metódust követi a *StopAsync()*, ami csak akkor fut, amikor az applikáció minden hiba nélkül leáll. Lényegében, ez állítja meg az e-mail küldő szolgáltatást:

```
public Task StopAsync(CancellationToken cancellationToken)  
{  
    Console.WriteLine("Sync Task stopped");  
    return null;  
}
```

Végül a *Send\_MAIL()* metódus zárja az osztályt. A mai társadalomban bevett szokás, hogy egy átlag felhasználó ébredés után kezébe veszi a telefonját és elkezdi tervezni a napját, értesül az aktuális hírekről, esetleg belájkol néhány macskás képet valamelyik szociális média oldalon. A *Send\_MAIL()* metódus pontosan az ilyen igényeket hivatott kiszolgálni. Minden nap reggel (esetünkben pontosan hat órakor) elküldi a felhasználó számára az aznapi

eseményeket, időrendi sorrendben. Első lépésként ahhoz, hogy a módszer működni tudjon, használni kell a *System.Net* és *System.Net.Mail* namespace-eket:

```
public Task Send_MAIL()  
{  
    while (true)  
    {
```

A `Send_MAIL()` módszer, amikor elindul, elindul egy `while` ciklus is, ami addig fut, amíg a bemeneti értéke igaz lesz:

```
var client = new SmtpClient()  
{  
    Port = 587,  
    Host = "smtp.gmail.com",  
    EnableSsl = true,  
    UseDefaultCredentials = false,  
    Credentials = new  
    NetworkCredential("ggmasterxgg@gmail.com", "exjwjqjdrknlzlk")  
};  
DateTime Today = DateTime.UtcNow;  
conn.Open();  
MySqlDataAdapter getMail = new MySqlDataAdapter("SELECT  
sql7385385.user.Email FROM calendar.user;", conn);  
DataTable dat = new DataTable();  
getMail.Fill(dat);  
foreach (DataRow row in dat.Rows)  
{  
    var tmp = row.Field<string>(0);  
    userMail = tmp;  
}  
conn.Close();
```

Első lépésben az *e-mail kliens* beállítása történik egy *Smtp kliens* segítségével. A *client* változóban a *Host* és *Port* rész jelzi, hogy az e-mail egy Google általi *Gmail* típusú e-mail címre lesz elküldve. Az `EnableSsl = true` a biztonságos kapcsolatot jelzi, a *Credentials* pedig a küldő felet hitelesíti. Miután ez kész, megtörténik a fogadó fél e-mail címének lekérése az adatbázisból az *userMail* nevű *string* típusú változóba:

```

conn.Open();
MySQLCommand cmd = new MySQLCommand("SELECT * FROM
sql7385385.calendar WHERE Date='" + Today.ToString("yyyy-
MM-dd") + "' ORDER BY TimeFROM ASC;", conn);
int TEvenets = Convert.ToInt32(cmd.ExecuteScalar());
conn.Close();

```

Miután a metódus visszakérte a fogadó fél e-mail címét, elindul egy adatbázis lekérdezés, ami egy *integer* típusú *TEvenets* változóba lekéri, hogy a *MySQLCommand*-ban levő adatbázis lekérdezés menyit adott vissza az elmentett adatok közül:

```

if (TEvenets != 0)
{
    DataTable dt = new DataTable();
    MySQLDataAdapter da = new
MySQLDataAdapter("SELECT * FROM sql7408148.calendar WHERE
Date='" + Today.ToString("yyyy-MM-dd") + "' ORDER BY TimeFROM
ASC;", conn);

    da.Fill(dt);
    string Body = "";
    foreach (DataRow row in dt.Rows)
    {
        var TimeFROM = row.Field<TimeSpan>(0);
        var TimeTO = row.Field<TimeSpan>(1);
        var Event = row.Field<string>(2);
        var Description = row.Field<string>(3);
        Body += TimeFROM + " - " + TimeTO + " : "
+ Event + " \n\r" + Description + "\n\r";
    }
}

```

Ha a *TEvenets* változó értéke a lekérdezés után nem egyenlő nullával, akkor a metódus elkezd lekérdezni az aznapi eseményeket az adatbázisból. A lekérdezés egy *DataTable* -be kéri vissza az adatokat, amik az: eseményekhez tartozó név, leírás, kezdete és vége. A *DataTable* sorait egy *foreach* ciklus tölti fel. Ez a ciklus minden egyes fordulás után a *string* típusú *Body* változóba elkezd beleírni az eseményeket:

```

var mail = new MailMessage()

```

```

{
    From = new MailAddress("ggmasterxgg@gmail.com"),
    Subject = "Hello, here is your schedule for Today",
    Body = Body
};
mail.To.Add(new MailAddress(userMail));
client.Send(mail);

```

Ha a metódus végzett az e-mail testének feltöltésével, akkor megtörténik az egész e-mail deklarálása a `new MailMessage()` segítségével, amiben a `From` jelzi, hogy kitől jön az e-mail. A `Subject` az email tárgyat állítja be, a `Body` pedig az e-mail testét, tehát ahova az események kiírása történik. Ezután az e-mailhez hozzáadódik a felhasználó e-mail címe és végül megtörténik az e-mail elküldése. Ha a metódus elején a `TEvents` változó értéke mégis egyenlő lenne nullával, akkor a feltétel vizsgálat `else` ága kezd el futni. Ekkor a felhasználó csak egy rövid üzenetet kap, hogy az adott napra semmi dolga sincs:

```

else
{
    var mail = new MailMessage()
    {
        From = new MailAddress("ggmasterxgg@gmail.com"),
        Subject = "Hello, here is your schedule for
Today...",
        Body = "Oh anyway today is a free day if i see
it correctly...\r\nEnjoy your free time."
    };
    mail.To.Add(new MailAddress(userMail));
    client.Send(mail);
}

```

Végül definiáltuk, hogy a metódus minden eltelt nap után lefusson:

```

DateTime nextStop = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, (DateTime.Now.AddDays(1)).Day, 6, 0, 0);
var timeToWait = nextStop - DateTime.Now;
var millisToWait = timeToWait.TotalMilliseconds;
Thread.Sleep((int)millisToWait);

```

A *nextStop* változó tartalmazza, hogy mikor fusson újra a kódrészlet, jelen esetben a következő napon reggel hatkor. A *timeToWait* változó tartalmazza, hogy mennyit kell várakozni a kódrészlet újra futtatásával, jelen esetben *nextStop* változó értéke mínusz az aktuális dátum és idő. Ezt követően a *millisToWait* változó megkapja a *timeToWait* értékét milliszekundumokra átalakítva. Végül a `Thread.Sleep()` függvény bemeneti értéként megkapja a *millisToWait* értékét integer típusra alakítva. A `Thread.Sleep()` a bemeneti értéknyi ideig leállítja a függvény futását.

A másik e-mail küldő szolgáltatás az *EMAILService* a C# osztályban van implementálva, szintén az *IHostedService* interfész alapján. Ez a szolgáltatás minden eltelt óra után ellenőrzi, hogy a következő órában lesz-e valamilyen esemény, ha lesz akkor küld egy e-mail az eseményről. A szolgáltatás hasonlóan működik, mint a *MorningMAIL* szolgáltatás. Első lépésként beállítja az e-mail küldéshez szükséges dolgokat, majd lekéri a felhasználó e-mail címét az adatbázisból. Következő lépésben az *Events* DataBase típusú tömb megkapja a `Get_EVENTS()`-tól az eseményeket. A `Get_EVENTS()` metódus első lépésben megnyitja az adatbázis kapcsolatot, majd lekéri azokat az eseményeket az adatbázisból, amik az aktuális idő és az aktuális idő plusz egy óra között vannak.

```

MySQLConnection connGet = new
MySQLConnection("server=sql7.freemysqlhosting.net;port=3306;da
tabase=sql7385385;user=sql7385385;password=PC914Uj4Ta");

    DateTime Today = DateTime.UtcNow;
    List<DataBase> cld = new List<DataBase>();
    DataTable dt = new DataTable();
    DateTime timeNow = DateTime.Now;
    DateTime timeNowPlusOne =
DateTime.Now.AddHours(1);

    MySqlDataAdapter da = new MySqlDataAdapter("SELECT
* FROM sql7385385.calendar WHERE Date='" + Today.ToString("yyyy-
MM-dd") + "' AND (TimeFROM BETWEEN '" +
timeNow.ToString("HH:mm") + "' AND '" + timeNowPlusOne + "')
ORDER BY TimeFROM ASC;", connGet);

```

A lekért adatokat belerakja egy listába majd visszaadja a listát tömbbé alakítva.

```

da.Fill(dt);
foreach (DataRow a in dt.Rows)
{

```

```

cld.Add(new DataBase
{
    ID = (int)a["ID"],
    Date = (DateTime)a["Date"],
    TimeFROM = (TimeSpan)a["TimeFROM"],
    TimeTO = (TimeSpan)a["TimeTO"],
    Event = a["Event"].ToString(),
    PriorityE = a["Priority"].ToString(),
    Description = a["Description"].ToString()
});
}
return cld.ToArray();

```

Miután az *Events* tömb megkapta az adatait, lefut egy *if* feltétel vizsgálat, hogy a tömb hosszúsága nem egyenlő nullával. Ha a feltétel igaz, akkor egy *foreach* cikluson belül a *Body* változó megkapja az események adatait. A ciklus után az e-mail formázása történik, majd az e-mail küldése.

```

if (Events.Length != 0)
{
    foreach (var tmp in Events)
    {
        Body += tmp.Event + "\n" + tmp.TimeFROM + " - " +
tmp.TimeTO + " \n\r" + tmp.Description + "\n\r";
    }
    var mail = new MailMessage()
    {
        From = new MailAddress("ggmasterxgg@gmail.com"),
        Subject = "Hello, you have an event in the next
hour",
        Body = Body
    };
    mail.To.Add(new MailAddress(userMail));
    client.Send(mail);
}

```

Végül definiálva van, hogy a metódus minden eltelt óra után lefusson:

```

DateTime nextStop = DateTime.Now.AddHours(1);
var timeToWait = nextStop - DateTime.Now;
var millisToWait = timeToWait.TotalMilliseconds;
Thread.Sleep((int)millisToWait);

```

(Sharp, 2018; Troelsen and Japikse, 2020; Himschoot, 2020; Murach, 2019).



### 3.3.1 Shared mappa, avagy Blazor és a Razor komponenssek működése, oldalak megjelenése

Ahhoz, hogy az oldal egy web-böngészőben is használható legyen, ahhoz az alap HTML szerkezetre szükség van. Ezt a HTML szerkezetet az *\_Host.cshtml* fájl tartalmazza. Ebben a fájlban van definiálva a karakter kódolás, itt vannak linkelve az egyes stílus fájlok, illetve a JavaScript kiterjesztésű fájlokra is itt történik a hivatkozás. A `<body>...</body>` tag-ek között vannak a Blazor működéséhez elengedhetetlen tag-ek:

```
<app>
    <component type="typeof (App) " render-
mode="ServerPrerendered" />
</app>
```

`<app>...</app>` tag alapján tudja a *Framework*, hogy a *razor* komponenseket hova kell majd betölteni, illetve, hogy milyen projektről van szó. Jelen esetben egy szerver applikációról, ahol a renderelés a szerver oldalon történik. Ezt követi egy rövid *script*, ami akkor jelenik meg, ha esetleg valamilyen hiba történik az oldal vagy a szerver működésében:

```
<div id="blazor-error-ui">
    <environment include="Staging,Production">
        An error has occurred. This application may no longer
        respond until reloaded.
    </environment>
    <environment include="Development">
        An unhandled exception has occurred. See browser dev
        tools for details.
    </environment>
    <a href="" class="reload">Reload</a>
    <a class="dismiss">✕</a>
</div>
```

Két fajta hiba üzenet van alapból definiálva. Az első produkciós hiba, ami figyelmeztet, hogy van valami hiba és az applikáció addig nem működik, amíg újra nem töltjük. A második hiba akkor jelenik meg, ha valamilyen fejlesztői hiba jelentkezne. Ekkor figyelmeztet, hogy egy nem várt kivétel következett be és nézzük meg a webböngészőnknek a fejlesztői eszközét. Mindkét

hiba esetén a hiba üzenet után megjelenik egy **Reload** gomb, ami újra tölti az oldalt, illetve egy **X** gomb, amivel bezárhatjuk a hibát.

A Blazor *Framework Razor* komponensekből állítja össze az egyes oldalakat. Minden Blazor projekt tartalmaz egy *Shared* mappát. Ez a mappa tartalmazza azokat a fájlokat, amikből a *Framework* összeállítja az adott oldal megjelenését. *MainLayout.razor* fájl írja le, hogy az oldal hogyan nézzen ki. A fájlban lévő komponensek a *LayoutComponentBase*-ből vannak származtatva:

```
@inherits LayoutComponentBase.
```

Jelen esetben a *Mainlayout.razor* fájl tartalmazza azt, hogy az oldal, hogyan jelenítse meg a navigációs menüt, illetve az egyes *Razor* komponensek hova legyenek majd betöltve. Az oldalakon a navigációs menü mindig fixen fent helyezkedik el egy sávban, amin úgynevezett *NavLink*-ek vannak, amik az egyes oldalakra hivatkoznak. Minden *NavLink*hez tartozik egy felirat is, amire ha rákattintunk, akkor átirányít minket a megfelelő oldalra, illetve egy *Open Iconic* ikon, ami grafikusán is jelzi, hogy melyik oldalról van szó. A navigációs menü után egy vízszintes választó vonal helyezkedik el, amit a *@Body* változó követ. Ebbe a változóba tölti be a Blazor az egyes komponensek tartalmát és így jelennek majd meg azok a felhasználó böngészőjében:

```
<div class="main">
    <div class="fixed-top px-4" style="background-
color:deepskyblue; z-index:2">
        <div class="row">
            <NavLink class="nav-link text-white" href=""
Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-
hidden="true"></span> Home
            </NavLink>
            <NavLink class="nav-link text-white"
href="MonthlyView">
                <span class="oi oi-calendar" aria-
hidden="true"></span> Calendar
            </NavLink>
            <NavLink class="nav-link text-white"
href="WeeklyView">
```

```

        <span class="oi oi-grid-four-up" aria-
hidden="true"></span> Weekly View
    </NavLink>
    <NavLink class="nav-link text-white"
href="EventList">
        <span class="oi oi-list" aria-
hidden="true"></span> Event List
    </NavLink>
    <NavLink class="nav-link text-white"
href="Priorities">
        <span class="oi oi-pencil" aria-
hidden="true"></span> Priorities
    </NavLink>
    <NavLink class="nav-link text-white" href="User">
        <span class="oi oi-cog" aria-
hidden="true"></span> User
    </NavLink>
</div>
</div>
<hr/>
<div class="content px-4">
    @Body
</div>
</div>

```

A *@Body* változóba a projektben található *Pages* mappa tartalma van betöltve. Ez a mappa tartalmazza az összes *Razor* komponenst, amik mindig a megfelelő oldal megjelenésért felelősek. Egy *Razos* komponsbén nem csak az adott oldal megjelenése van implementálva, hanem itt vannak implementálva továbbá az oldal funkcionalitásáért felelős metódusok is. Ilyenre példa, hogy az *Index*, vagyis a kezdőlapón itt van implementálva az aktuális idő és dátumot mutató *Bootstrap* kártya, vagy az események mentéséért felelős metódusok, melyek az *AddEvent* nevű *Razor* komponensben vannak implementálva. (Litvinavicius, 2019; Himschoot, 2020)

### 3.4 Kezdő oldal

Minden weboldal vagy webes applikáció esetében szükség van egy *Index* oldalra. Ez az adott weboldal honlapja (kezdő oldal). Jelen esetben az *Index* oldal tartalmaz egy aktuális időt, dátumot és napot mutató *Bootstrap* kártyát, illetve egy táblázatot, amiben az aznapi eseményeinket tudjuk soron követni. Az oldal, amikor betöltődik, meghívásra kerül az `OnInitializedAsync()` függvény.

Az `OnInitializedAsync()` függvény akkor hívódik meg, amikor a komponens készen áll ahhoz, hogy elinduljon és megkapta az összes kezdeti paraméterét a szülőtől a renderfában. Felül tudjuk írni, ha aszinkron művelet szeretnénk, hogy végrehajtsa és szeretnénk, ha a komponens frissüljön, ha a művelet befejeződött. [6]

Az `OnInitializedAsync()`-ben implementálva vannak az *Index* oldal működéséhez és megjelenéséhez elengedhetetlen metódusok. Az *Events* nevű tömb itt kapja meg az aznapi eseményeket a `DataBaseServices.Get_TODAY_EVENTS()` metódus segítségével:

```
public Task<DataBase[]> Get_TODAY_EVENTS()
{
    MySqlConnection connGet = new
    MySqlConnection("server=sql7.freemysqlhosting.net;port=3306;da
    tabase=sql7385385;user=sql7385385;password=PC914Uj4Ta");
    DateTime Today = DateTime.UtcNow;
    List<DataBase> cld = new List<DataBase>();
    DataTable dt = new DataTable();
```

A `Get_TODAY_EVENTS()` metódus a *DataBaseServices* osztály metódusa. Amikor ez a metódus elindul, megtörténik egy lokális adatbázis kapcsolat deklarálása, illetve pár segédváltozó deklarálása, mint a *Today DateTime* típusú változó, ami aktuális dátumot kapja meg értéként, valamint egy *cld* nevű lista és egy *dt* nevű *DataTable*.

```
MySqlDataAdapter da = new MySqlDataAdapter("SELECT * FROM
sql7385385.calendar WHERE Date='" + Today.ToString("yyyy-MM-
dd") + "';", connGet);
da.Fill(dt);
foreach (DataRow a in dt.Rows)
{
```

```

        cld.Add(new DataBase {
            ID = (int)a["ID"],
            Date = (DateTime)a["Date"],
            TimeFROM = (TimeSpan)a["TimeFROM"],
            TimeTO = (TimeSpan)a["TimeTO"],
            Event = a["Event"].ToString(),
            PriorityE = a["Priority"].ToString(),
            Description = a["Description"].ToString()
        });
    }

    return Task.FromResult(cld.ToArray());
}

```

Az aznapi események lekérése egy *DataAdapter* segítségével történik, amiben a lekérés van definiálva. A lekérés egy egyszerű SELECT műveletet hajt végre, ami lekérdezi az összes olyan sort, ahol a *Date* oszlop értéke megegyezik a *Today* változó értékével a **calendar** nevű táblából. A lekért adatok ezután belekerülnek a *cld* listába és a függvény visszatérési értéként visszaadja a *cld* listát tömbbé alakítva.

Következő lépésben az *User* tömb kapja meg a felhasználó adatait a *DataBaseServices.GetUser()* metódus segítségével:

```

public Task<DataBase[]> GetUser()
{
    MySqlConnection connGet = new
    MySqlConnection("server=sql7.freemysqlhosting.net;port=3306;da
    tabase=sql7385385;user=sql7385385;password=PC914Uj4Ta");

    List<DataBase> cld = new List<DataBase>();
    DataTable dt = new DataTable();
    MySqlDataAdapter da = new MySqlDataAdapter("SELECT * FROM
    sql7385385.user", connGet);

    da.Fill(dt);
    foreach (DataRow a in dt.Rows)
    {
        cld.Add(new DataBase {
            ID = (int)a["ID"],

```

```

        FirstName = a["Firstname"].ToString(),
        LastName = a["Lastname"].ToString(),
        Email = a["Email"].ToString(),
        BirthDay = (DateTime)a["BirthDay"]
    });
}

return Task.FromResult(cld.ToArray());
}

```

A `Get_USER()` metódus működése ugyanaz, mint a `Get_TODAY_EVENTS()` metódusé, csak itt a lekérés az **user** táblából történik és a felhasználó adatait adja vissza a *cld* listában, tömbbé alakítva.

Ahogy lefutott az `OnInitializedAsync()` függvény, elindul egy időzítő. Ahhoz, hogy a projekt „real-time” legyen, ahhoz létrejön egy időzítő, ami minden eltelt másodperc után meghívja a `Refresh()` függvényt, amiben a *currentDate* *DateTime* típusú változóba lekéri az aktuális dátumot és időt:

```

@{var timer = new Timer(new TimerCallback(_ =>
    {
        Refresh();
        InvokeAsync(() =>
        {
            StateHasChanged();
        });
    }), null, 1000, 1000);
}

```

A C#-ba beépített `System.Threading` függvény-könyvtárban található *Timer* osztály alapján működik az oldalon lévő időzítő. Létrejön egy *Timer* objektum, aminek mind a négy attribútuma fel van használva ahhoz, hogy legyen egy olyan időzítő, ami minden másodpercben frissíti az oldal egyes részeit. A *Timer* első attribútuma a *TimerCallback*, ez tartalmazza, hogy az időzítő adott idő elteltével, milyen utasításokat hajtson végre. Jelen esetben, végre hajtja a `Refresh` függvényt, utána a `StateHasChange` beépített függvény jelzést küld a megfelelő komponensnek vagy komponenseknek, hogy esetleges adatbéli változás történt, és ha szükséges, akkor újra „rendereli” az oldalt, vagy csak adott komponenseket. A második paraméter null értékű, mivel nem követi semmilyen objektum állapotát. A harmadik

paraméterben 1000-t kap, ez jelzi az időzítőnek, hogy a létrehozása után mennyit kell várnia. Ezt a paramétert „Due Time” vagy „Delay Time”-nak nevezik. Végül a negyedik paraméterben ismét 1000-t kap, ami jelzi, hogy a *TimerCallback* tartalma minden eltelt másodperc után végrehajtódjon.

Az *Index* oldal két *Bootstrap* kártyából épül fel. A jobb oldali kártya fejléce tartalmazza a felhasználó köszöntését. A kártya teste pedig egy „real-time” órát, valós dátumot és az aktuális napot tartalmazza:

```
<div class="card-header py-3">
    @if (User != null)
    {
        @foreach (var tmp in User)
        {
            <h6 class="m-0 font-weight-bold text-primary">Hello
@tmp.FirstName @tmp.LastName!</h6>
        }
    }
</div>
```

A *Bootsrtap* kártya fejlécében csak akkor jelenik meg a felhasználó neve, ha a *User* tömb kapott adatokat. Ha a *User* tartalmaz adatokat, akkor egy *foreach* ciklus segítségével vannak elérve az egyes attribútumok a kiíratáshoz:

```
<div class="card-body">
    <h1 class="display-1 font-weight-bold text-primary nav
justify-content-center">@currentDate.ToString("HH:mm:ss")</h1>
    <h1 class="display-1 font-weight-bold text-primary nav
justify-content-center">@currentDate.ToString("yyyy-MM-
dd")</h1>
    <h1 class="display-1 font-weight-bold text-primary nav
justify-content-center">@currentDate.DayOfWeek.ToString()</h1>
</div>
```

A kártya testében, három sorba, az aktuális idő, ezt követően az aktuális dátum és végül az aktuális nap van kiíratva.

A második *Bootstrap* kártya az oldalon egy HTML táblát tartalmaz, amibe az aznapi események jelennek meg felsorolva, az eseményhez tartozó idők alapján. Az események az

*Events* nevű tömbben tárolódnak. Az események kiírásánál először ellenőrizve van, hogy az *Events* tömb tartalmaz-e adatot. Ezt követően három segédváltozó definiálása történik, amik mind *TimeSpan* típusúak, ezek a: *time*, *time1* és *endTime* változók:

```
<tbody>
    @if (Events != null)
    {
        TimeSpan time = new TimeSpan(0, 0, 0);
        TimeSpan time1 = new TimeSpan(0, 0, 0);
        TimeSpan endTime = new TimeSpan(23, 0, 0);
```

Következő lépésben egy *foreach* ciklus végig megy az *Events* tömbön és elkezdődik a kiírása az aktuális napi eseményeknek. A cikluson belül a *time1* segédváltozó megkapja a soron lévő esemény *TimeFROM* változójának értékét, ami után egy *if* feltétel vizsgálat következik, ami csak akkor fut le, ha az aktuális esemény *TimeFrom* változója nem egyenlő 00:00:00-val. Ekkor egy *while* ciklus kezd el futni, ami elkezd feltölteni a táblát órákkal addig, amíg el nem jut az aktuális esemény kezdeti idejéhez:

```
foreach (var events in Events)
{
    time1 = events.TimeFROM;
    if (!TimeSpan.Equals(events.TimeFROM, new TimeSpan(0, 0,
0)))
    {
        while (TimeSpan.Compare(time, events.TimeFROM.Add(new
TimeSpan(-1, 0, 0))) < 1)
        {
            <tr class="text-black-50"
style="background-color:lightgreen">
                <td>@time</td>
                <td></td>
            </tr>
            time += TimeSpan.FromHours(1);
        }
    }
}
```



Ha az `if` feltétel vizsgálat befejeződött és eljutott a kód az első eseményhez, akkor elkezdődik az események kiíratása. Do-While ciklus segítségével történik az esemény kiíratása, három sorba, amiknek a háttérszíne is megváltozik az eseményhez tartozó prioritás színére. Különben a tábla minden sora világoszöld színt kap, amivel jelzi, hogy ott nincs semmi esemény. Az első sorba kerül az eseményhez tartozó kezdeti idő és neve, a másodikba megy az esemény leírása, a harmadikba pedig az esemény befejezési ideje és ismét az esemény neve. Miután kiírt egy eseményt, a *time1* változó értéke egyenlő lesz az aktuális esemény órájával nulla percel és nulla másodpercel, a *time* változó értéke pedig megkapja az aktuális eseményhez tartozó befejezési idő óráját és percét. A Do-While ciklus addig fut, amíg az eseményhez tartozó *TimeTO* változó értéke nagyobb nem lesz, mint a *time1* változó értéke:

```
do
{
    <tr class="text-white" style="background-
color:@events.PriorityE">
        <td>@events.TimeFROM</td>
        <td>@events.Event</td>
    </tr>
    <tr class="text-white" style="background-
color:@events.PriorityE">
        <td colspan="2">@events.Description</td>
    </tr>
    <tr class="text-white" style="background-
color:@events.PriorityE">
        <td>@events.TimeTO</td>
        <td>@events.Event</td>
    </tr>
    time1 = new TimeSpan(events.TimeTo.Hours,0,0);
    time = new TimeSpan(events.TimeTO.Hours,
events.TimeTO.Minutes, 0);
} while (TimeSpan.Compare(time1, events.TimeTO)>1);
```

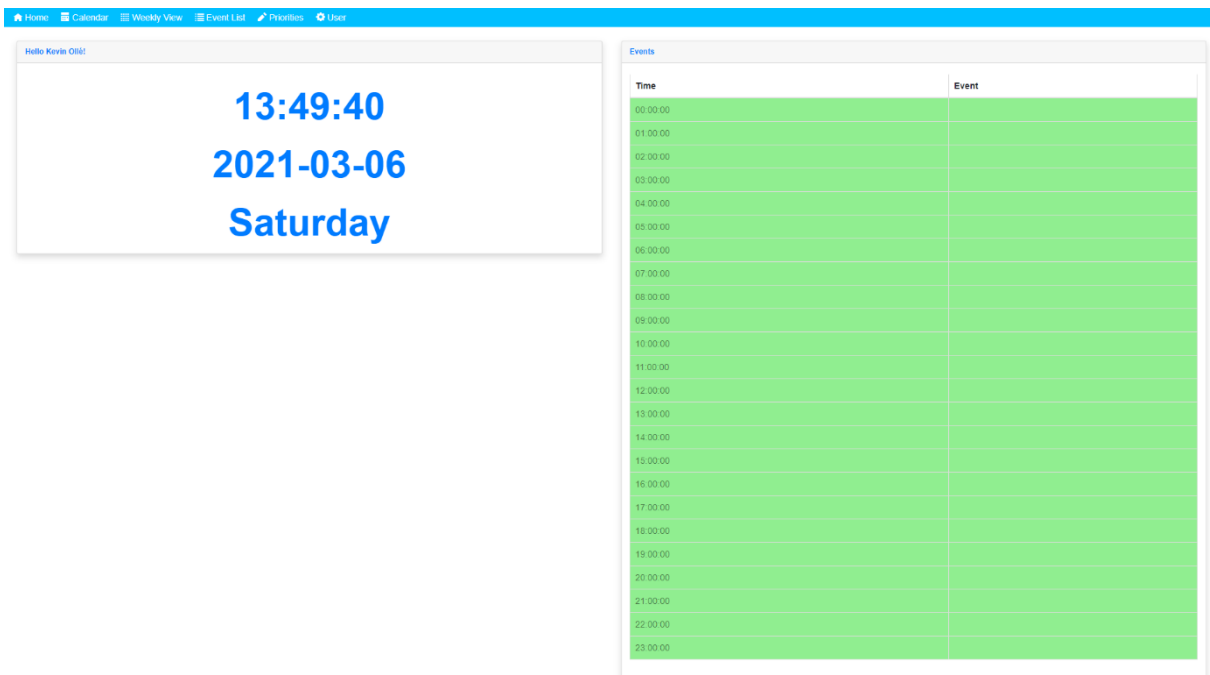
Ha a Do-While ciklus befejezte a futását, akkor a *time* változó ellenőrzése következik, hogy a perc része egyenlő e nullával, ha nem akkor beállítódik a megfelelő idő:

```
if (time.Minutes != 0)
```

```
{
    time = new TimeSpan(time.Hours + 1, 0, 0);
}
```

Ha esetleg nem lenne több esemény, amit ki szeretnénk írni, akkor elkezdődik egy `Do-While` ciklus, amiben a `time` változó értékének kiírása folyik addig, amíg a `time` és az `endTime` változó értéke meg nem egyezik:

```
do
{
    <tr class="text-black-50" style="background-
color:lightgreen">
        <td>@time</td>
        <td></td>
    </tr>
    time += TimeSpan.FromHours(1);
} while (TimeSpan.Compare(time, endTime) < 1);
```



**4. ábra:** Kezdőlap megjelenése; Forrás: saját kép

(Murach, 2019; Himschoot, 2020; Matsinopoulos, 2020)

### 3.5 Heti nézet

Mivel a projekt egy elektronikus napló, ezáltal elengedhetetlen része egy heti nézet, ahol napra pontosan tudjuk követni az eseményeket. Ezen az oldalon az események oszlopokba

rendezve vannak grafikusán megjelenítve, ahol az egyes oszlopok az egyes napokat jelzik. Amikor betölt az oldal, elindul az `OnInitializedAsync()` függvény, amiben az *Events* nevű *DataBase* típusú tömb megkapja az eseményeket az adatbázisból a `DataBaseServices.Get_WEEK_EVENTS()` metódus segítségével.

A metódus ugyanúgy működik, mint a `Get_TODAY_EVENTS()` metódus a kezdőlapon. Ugyanúgy egy listába vannak visszakérve az események adatai egy *MySqlDataAdapter* segítségével, viszont a lekérdezés máshogy néz ki. Ismét visszakér mindent a *calendar* tömbből, de csak akkor, ha a *date* oszlopban tárolt dátum megegyezik az aktuális héthez tartozó dátumokkal:

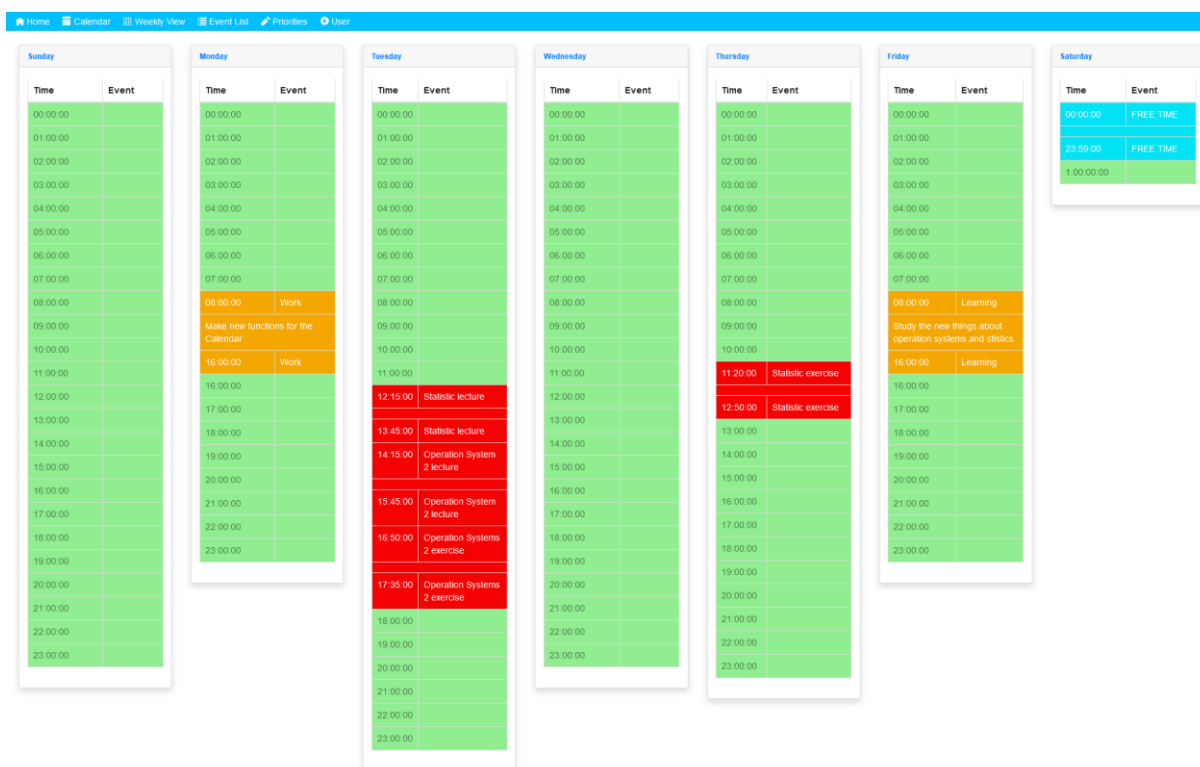
```
SELECT * FROM sql7385385.calendar WHERE  
YEARWEEK(date)=YEARWEEK(NOW()) "
```

Az oldal megjelenése nagyon egyszerűen van kialakítva. A back-end részen deklarálva van egy *days* string típusú tömb, amiben a hét napjai vannak tárolva.

```
public string[] days = { "Sunday", "Monday", "Tuesday",  
"Wednesday", "Thursday", "Friday", "Saturday" };
```

Az oldal megjelenítésében egy `for` ciklus segít, ami nullától hétig halad egyesével és minden egyes lépés után kirajzol egy *Bootstrap* kártyát, ami a *days* tömb *i*-edik értékét kapja meg címként. A kártya testében pedig egy HTML táblázat kerül, amiben időre pontosan kiíratásra kerülnek az események. A HTML tábla két oszlopot tartalmaz. Az egyik oszlop *Time* névvel jelzi az időt az *Events* oszlop pedig jelzi az esemény nevét. Ha az *Events* tömb nem tartalmaz eseményt, akkor a HTML tábla minden egyes cellája világoszöld színt kap, különben egy adott eseményhez tartozó cella az eseményhez tartozó prioritás színét kapja meg. A tábla feltöltése az eseményekkel ugyan azt a megoldást követi, mint a kezdőlap esetében, csak a `foreach` ciklusban van egy `if` feltétel vizsgálat, ami ellenőrzi, hogy a soron levő esemény a megfelelő oszlopba kerüljön:

```
if (events.Date.DayOfWeek.ToString() == days[i])
```



5. ábra: Heti nézet megjelenése; Forrás: saját kép

(Murach, 2019; Himschoot, 2020; Matsinopoulos, 2020)

### 3.6 Havi nézet

Bármelyiknek, ha beszélünk egy kalendár applikációról, annak rögtön a havi nézetes, napokra pontosan foglalható eseményes megoldás jut eszébe. A projekt sokáig nem tartalmazta ezt a nézetet, mert első körben más fajta megközelítésben szeretnénk volna megoldani ezt az oldalt, de aztán végül megvalósításra került. Első körben, amikor nekiláttunk megoldani ezt az oldalt, akkor tudatosult bennünk, hogy a *Blazor* direkt módon nem képes elérni a *DOM* objektumokat, tehát *C#* kódból nem tudtuk elérni a különböző *HTML* elemeket. Fejlesztés helyett, kutatással telt itt egy kis idő, hogy hogyan is lehetne ezt a problémát áthidalni. Végül arra jutottunk, hogy kihasználjuk a *Blazor* és a *Javascript* kapcsolatát, tehát valami *C#* és *JavaScript* ötvözte megoldás született végül.

Az oldalhoz tartozó back-end a kezdőlapra már megismert időzítővel kezdődik. Mikor betölt az oldal, meghívódik az `OnInitializedAsync()` függvény, amiben a *dates DataBase* típusú tömb megkapja a dátumokat a `DataBaseServices.Get_DATES()` metódus segítségével. A `Get_DATES()` metódus a **calendar** táblából lekéri a *Date* oszlop adatait, ismétlődés nélkül. A *month DataBase* típusú tömb megkapja az összes adatot a **calendar** táblából. Az események és a dátumok lekérdezése után következik a prioritások

lekérése a *priority DataBase* típusú tömbbe, a `DataBaseServices.Get_PRIORITY()` metódus segítségével. Az összes adat lekérése után elindul a `COLOR()` függvény, illetve meghívódik egy *JavaScript* függvény a `ChangeColor()` függvény, ami bemeneti értéként megkapja a *cellID* változót. A `Refresh()` függvény az időzítőben minden másodperc elteltével meghívódik, ugyanazt az implementációt hajtja végre, mint az `OnInitializeAsync()` függvény:

```
protected override async Task OnInitializedAsync()
{
    month = await DataBaseServices.Get_EVENTS();
    dates = await DataBaseServices.GetDATES();
    priority = await DataBaseServices.Get_PRIORITY();
    COLOR();
    await JSRuntime.InvokeVoidAsync("ChangeColor", cellID);
}

protected async Task Refresh()
{
    month = await DataBaseServices.Get_EVENTS();
    dates = await DataBaseServices.GetDATES();
    priority = await DataBaseServices.Get_PRIORITY();
    COLOR();
    await JSRuntime.InvokeVoidAsync("ChangeColor", cellID);
}
```

A `ChangeColor()` *JavaScript* függvény a kalendár celláinak színeiért felel. Első lépésben minden cellát, ami a *Calendar* osztályt tartalmazza, beszínezi a háttérét világoszöldre. Ezután a *res* változó deklarálása történik, ami értéként megkapja a bemeneti változó szóköz alapján történő feldarabolását. Miután a változó deklarálás megtörtént, jön egy `for` ciklus, ami nullától a *res* tömb utolsó eleméig megy. A ciklusban azok a cellák, amiknek a neve egyenlő a tömb aktuális értékével, azoknak a háttérszíne narancssárgára állítódik. Végül a függvény azzal záródik, hogy azok a cellák, amik a *CalendarE* osztályt tartalmazzák, azok háttérszíne világosszürke lesz:

```
function ChangeColor(tmp) {
    $(".Calendar").css("background", "lightgreen");
```

```

var res = tmp.split(" ");
for (i = 0; i < res.length; i++) {
    $('td[name=' + parseInt(res[i]) + "]").css({
'background': 'orange' });
}
$('.CalendarE').css("background", "lightgrey");
}

```

A *cellID* változó a `COLOR()` függvényben kap értéket. A `COLOR()` függvényben első lépésben egy *DateTime* típusú *Date* nevű lista deklarálása történik, amit az *tmpID* üres *string* típusú változó deklarálása követ. Miután ez megvan, elkezd futni egy *foreach* ciklus a *month* tömbön. A *for* ciklusban az *tmpID* változó kezd el értéket kapni a következő módon: szóköz + aktuális adathoz tartozó dátum éve + aktuális adathoz tartozó dátum hónapja + aktuális adathoz tartozó dátum napja. Miután a ciklus futása befejeződött, a *cellID* változó megkapja az *tmpID* változó értékét:

```

private void COLOR()
{
    List<DateTime> Date = new List<DateTime>();
    string tmpID = "";
    foreach (var tmp in month)
    {
        tmpID += " " + tmp.Date.Year + " " + tmp.Date.Month
+ " " + tmp.Date.Day;
    }
    cellID = tmpID;
}

```

Minden cellához tartozik egy *onclick* esemény, ami akkor hívódik meg, ha rákattintunk a cellára. Az *onclick* esemény a `SetDate()` nevű függvényt hívja meg az oldalon. A `SetDate()` függvényben először a *getDate string* típusú változó kap értéket a `Get_Date()` *JavaScript* függvény alapján, ami *string* értéket ad majd vissza. A *JavaScript* oldalon deklarálva van egy *data* nevű változó. Ezt követi egy eseményfigyelő, ami a *data* változó értékét beállítja annak a cellának az *id*-jét, amire rákattintottunk. Végül jön a `Get_Date()` függvény deklarálása, ami visszaadja a *data* változót:

```

var data;

$(document).on("click", "#calendar td", function (e) {
    data = $(this).attr('id');
});

function Get_Date() {
    return data;
}

```

Miután a *getDate* változó megkapta értékét, jön egy lokális változó deklarálása *dt* névvel, *dateTime* típussal. A *dt* változó a deklarálás során rögtön kap is értéket, méghozzá a *getDate* értéket kapja meg, *DateTime* típusú konvertálva. Következő lépésben a *monthTMP DateTime* típusú tömb kapja meg azokat az eseményeket, amikhez tartozó dátum megegyezik a *dt* változó értékével a *DataBaseServices.Get\_EVENTbyDATE()* függvény által.

A *Get\_EVENTbyDATE()* metódus bemeneti értéként egy *DateTime* típusú változót vár, majd a bemeneti érték szerint visszakéri a listába azokat az eseményeket, amikhez tartozó dátum megegyezik a bemeneti értékkel:

```

public Task<DataBase[]> Get_EVENTbyDATE(DateTime date)
{
    MySqlConnection connGet = new
    MySqlConnection("server=sql7.freemysqlhosting.net;port=3306;da
    tabase=sql7385385;user=sql7385385;password=PC914Uj4Ta");
    List<DataBase> cld = new List<DataBase>();
    DataTable dt = new DataTable();
    MySqlDataAdapter da = new MySqlDataAdapter("SELECT * FROM
    sql7385385.calendar WHERE calendar.Date='" +
    date.ToString("yyyy-MM-dd") + "';", connGet);
    da.Fill(dt);
    foreach (DataRow a in dt.Rows)
    {
        cld.Add(new DataBase {
            ID = (int)a["ID"],
            Date = Convert.ToDateTime(a["Date"]),
            TimeFROM = (TimeSpan)a["TimeFROM"],

```

```

        TimeTO = (TimeSpan)a["TimeTO"],
        Event = a["Event"].ToString(),
        PriorityE = a["Priority"].ToString(),
        Description = a["Description"].ToString()
    });
}

return Task.FromResult(cld.ToArray());
}

```

Ha mindez hibátlanul lefutott, megnyílik egy *modal*, amiben a megfelelő események vannak kiírva egy HTML táblázatba:

```

protected async Task SetDate()
{
    getDate = await JSRuntime.InvokeAsync<string>("Get_Date");
    DateTime dt = DateTime.Parse(getDate);
    monthTMP = await DataBaseServices.Get_EVENTbyDATE(dt);
    await JSRuntime.InvokeVoidAsync("Edit_Modal");
}

```

A havi nézet egy *Bootstrap* kártyából épül fel. A kártya fejléce tartalmazza az aktuális hónapot, ami egy *DateTime* típusú *C#* változó és a beépített *ToString*(„MMMM”) függvény segítségével van kiírva (csak a dátum része):

```

<div class="card-header py-3">
    <h3 class="m-0 font-weight-bold text-
primary">@date.ToString("MMMM").ToUpper()</h3>
</div>

```

A kártya teste első lépésben egy **Previos month** és egy **Next month** gombot tartalmaz. Ezen gombok végzik el a következő vagy az előző hónapra történő ugrást:

```

<div class="form-group">
    <a style="color:white" class="btn btn-primary"
@onclick="PREV">
        &#8592; Previous month
    </a>
    <a style="color:white" class="btn btn-primary float-
right" @onclick="NEXT">
        &#8594; Next month
    </a>
</div>

```



```

        </a>
    </div>

```

A **Previous month** gomb kattintás esetén a `PREV()` függvényt hívja meg, ami a *date* változó értékéből levon egy hónapot. A **Next month** gomb kattintás esetén a `NEXT()` függvényt hívja meg, ami a **date** változó értékéhez ad hozzá egy hónapot:

```

protected async Task PREV()
{
    date = date.AddMonths(-1);
}

protected async Task NEXT()
{
    date = date.AddMonths(1);
}

```

A hónapváltó gombok után jön egy táblázat. A táblázat fejlécében a hét napjai vannak felsorolva. Vasárnapkal kezdődik a hét a *C#* adottságai miatt, mert a *DateTime* változó minden hetet az amerikai stílus alapján vasárnapkal kezd. A tábla teste hét oszlopból és hat sorból áll. A tábla megalkotása *C#* kóddal történik. Először három *DateTime* típusú változó deklarálása történik, a beszédes *firstDayOfMonth* és a *lastDayOfMonth* azaz a két változó, ami a hónap első és utolsó napját tartalmazza. Ezek után a *DateTime* típusú *tmp* változó, ami a *firstdayOfMonth* változó értékét kapja meg alaphól. Ezeket a változókat követi az *ID string* típusú, alaphól üres szöveget tartalmazó változó:

```

<tbody>
    @{
        DateTime firstDayOfMonth = new DateTime(date.Year,
        date.Month, 1);
        DateTime lastDayOfMonth =
        firstDayOfMonth.AddMonths(1).AddDays(-1);
        DateTime tmp = firstDayOfMonth;
        string ID = "";

```

A segédváltozók deklarálása után elkezdődik a tábla rajzolása. A tábla rajzolása két *for* számláló ciklussal valósul meg. Az első ciklus a tábla soraiért felelős, tehát ez nullától ötig megy, míg a második ciklus nullától hatig, ami pedig a tábla oszlopaiért felelős. A második ciklus egy *if* feltétel vizsgálattal kezd. Ha a *firstDayOfMonth* aktuális állása az aktuális héten

meg egyezik a *j* értékével és a *firstDayOfMonth* kisebb vagy egyenlő a *lastDayOfMonth* változó értékével, akkor lefut az *if* feltételben deklarált kód:

```
@for (int i = 0; i < 6; i++)
{
    <tr>
        @for (int j = 0; j < 7; j++)
        {
            if (((int)firstDayOfMonth.DayOfWeek == j) &&
                (firstDayOfMonth <= lastDayOfMonth))
            {
```

Ha a program belép az *if*-be, akkor első lépésben az *ID* változó értéket kap, ami a *firstDayOfMonth* év, hónap és napja lesz. Ezt a megoldást használtuk, hogy a későbbiekben minden cellának saját azonosítója legyen:

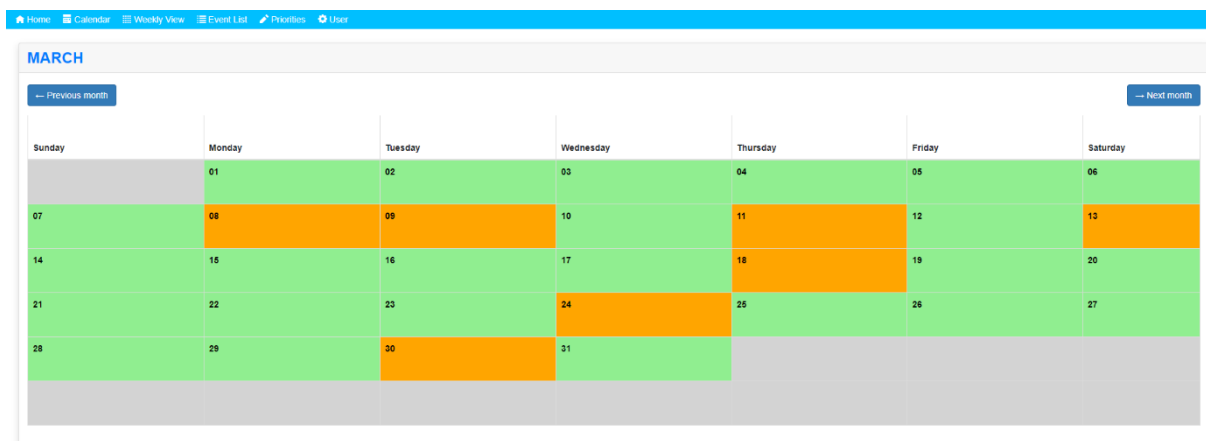
```
ID = firstDayOfMonth.Year.ToString() + " " +
firstDayOfMonth.Month.ToString() + " " +
firstDayOfMonth.Day;
```

Következő lépésben a cella létrehozása következik be. Minden cella először fehér színt kap, majd a későbbiekben kapja meg a zöld vagy narancssárga színt, attól függően, hogy van-e az adott napra valamilyen esemény. Ha nincs esemény, akkor a cella zöld színt kap, különben meg narancssárgát. Minden cella a *Calendar* osztályt kapja meg, a stílusuk pedig ugyan úgy egyenlő lesz 15%-os szélesség 70 pixeles magasság és fehér háttérszínnel. A cellák neve egyenlő lesz az *ID* változó aktuális értékével, a cellákhoz tartozó *Id* pedig egyenlő lesz a *firstDayOfMonth* változó értékével. Minden cella rendelkezik egy *onclick* eseménnyel, ami a *SetDate* névre hallgat. A cella tartalma egyenlő lesz a *firstDayOfMonth* változó *ToString(„dd”)* függvény segítségével visszaadott napjával, amit számként jelenít meg a *ToString()* függvénynek beadott „dd” érték miatt. Minden kirajzolt cella után a *firstDayOfMonth* változó értéke egy nappal növekszik:

```
<td class="Calendar" style="width: 15%; height: 70px;
background:white; font-weight: bold; color: black" name="@ID"
id="@firstDayOfMonth" @onclick="SetDate">
    @firstDayOfMonth.ToString("dd")
</td>
firstDayOfMonth = firstDayOfMonth.AddDays(1);
```

Ha esetleg az *if* feltétel vizsgálat hamis eredmény adna vissza, akkor pedig egy világosszürke háttérrel rendelkező cella kirajzolása hajtódik végre:

```
else
{
    <td class="CalendarE" style="width: 15%; height: 70px;
    background:lightgrey"></td>
}
```

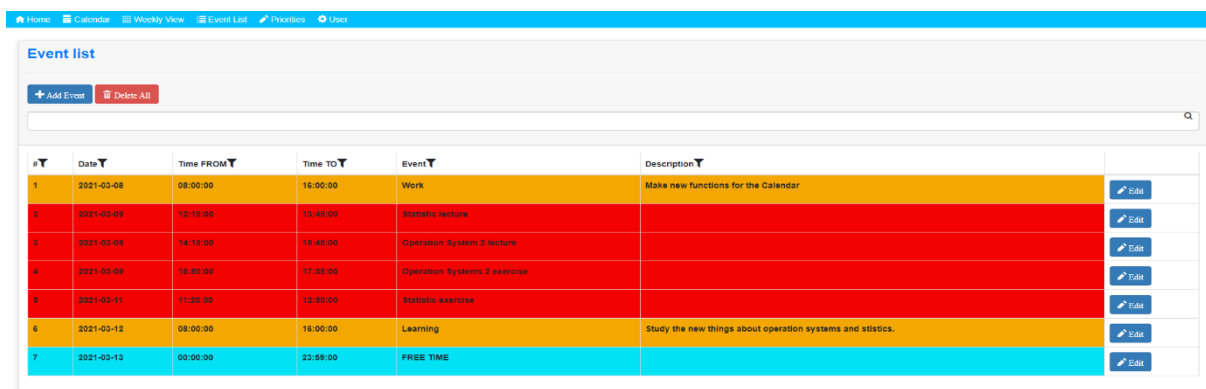


6. ábra: Havi nézet megjelenése; Forrás: saját kép

(Murach, 2019; Himschoot, 2020; Matsinopoulos, 2020; Murach, 2020)

### 3.7 Események listája

Az események listája az az oldal a projekten belül, ahol az események egy listában megjelennek. Itt a felhasználó tudja az eseményeket módosítani, szűrni vagy akár rendezni.



7. ábra: Események listája megjelenése; Forrás: saját kép

Az oldal back-endje a szokásos módon az `OnInitilizedAsync()` függvénnyel indul. Itt a *priority DataBase* típusú tömb kap adatokat a `DataBaseServices.Get_PRIORITY()` metódus által. A metódus visszaker minden tárolt adatot a *priority* táblából. Következő lépésben

az *sortedEvents* *DataBase* típusú tömb megkapja az eseményekhez tartozó adatokat a *DataBaseServices.Get\_EVENTS()* metóduson keresztül. Az időzítőben a *REFRESH()* függvény hajtódik végre, minden eltelt másodperc után. A függvényben ugyan azok a metódusok vannak hívva, mint az *OnIntializedAsync()* függvényben, illetve egy JavaScript függvény, ami a táblában történő keresésért felelős:

```
protected async Task Refresh()
{
    JSRuntime.InvokeVoidAsync("SEARCH");
    priority = await DataBaseServices.Get_PRIORITY();
    sortedEvents = await DataBaseServices.Get_EVENTS();
}

protected override async Task OnIntializedAsync()
{
    priority = await DataBaseServices.Get_PRIORITY();
    sortedEvents = await DataBaseServices.Get_EVENTS();
}
```

Az adatok kiírása a felhasználó számára egy HTML táblázatban történik meg. A tábla testébe egy *foreach* ciklus segítségével történik meg az adatot kiíratása. A ciklus végig megy az *sortedEvents* tömbön és az aktuális sorba kiírja az aktuális eseményhez tartozó adatokat, az utolsó oszlopba pedig rak egy gombot, amivel az adott eseményt lehet szerkeszteni. A gomb egy *onclick* eseményre meghívja a *SetEvent()* függvényt, az aktuális esemény *ID*-jével. A *SetEvent()* függvény a *t* objektumba beállítja a kiválasztott esemény adatait a *SingelOrDefault()* függvény segítségével:

```
<tbody id="EventTable">
    @{
        if (sortedEvents != null)
        {
            foreach (var Event in sortedEvents)
            {
                <tr style="background-
color:@Event.PriorityE">
                    <td><b>@Event.ID</b></td>
```

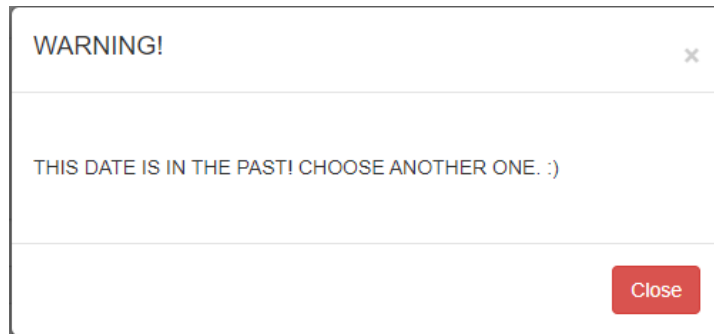
```

        <td><b>@Event.Date.ToString("yyyy-MM-
dd")</b></td>

        <td><b>@Event.TimeFROM</b></td>
        <td><b>@Event.TimeTO</b></td>
        <td><b>@Event.Event</b></td>
        <td><b>@Event.Description</b></td>
        <td style="background-color:white">
            <button class="btn btn-primary
oi oi-pencil" name="gomb" data-toggle="modal" @onclick="(e
=> SetEvent(Event.ID))" data-target="#EditModal">
                Edit
            </button>
        </td>
    </tr>
</tr>
    }
}
}
</tbody>
private void SetEvent(int id)
{
    t = events.SingleOrDefault(m => m.ID == id);
}

```

Miután a megfelelő *ID*-vel rendelkező eseményhez tartozó adatok vissza lettek töltve a *t* objektumba megnyílik egy *Bootstrap modal*, amiben a kiválasztott esemény adatait lehet megváltoztatni. Ha megváltoztattunk valamit, akkor a **Save** gomb segítségével el lehet menteni a változtatást. A gomb *onclick* eseményében a `SAVE()` függvény hívódik meg. A függvény első lépésben leellenőrzi, hogy a kiválasztott dátum kisebb-e mint az aznapi dátum. Ha kisebb, akkor meghívja a `Date_Modal JavaScript` függvényt, ami figyelmeztet arra, hogy a múltba nem lehetséges eseményt menteni.



8. ábra: Múltba történő foglalás figyelmeztetése; Forrás: saját kép

Ha a dátum nem kisebb, mint az aznapi dátum, akkor az *ISRES* integer típusú változóba meghívódik a `DataBaseServices.Is_RESERVABLE()` metódus az esemény foglalási idejével és dátumával. Maga a metódus lekéri a **calendar** táblából azt a sort, ahol a *TiemFROM* oszlop a bementi *timeStart* és *TimeEnd* között van és a *Date* oszlop egyenlő a bemeneti *date* változóval. A kapott értéket egy integer típusú változóban küldi vissza:

```
public int Is_RESERVABLE(DateTime timeStart, DateTime
timeEnd, DateTime date)
{
    conn.Open();
    MySqlCommand comm = new MySqlCommand("SELECT * FROM
sql7385385.calendar WHERE (sql7385385.calendar.TimeFROM
BETWEEN '" + timeStart.ToString("HH:mm:ss") + "' AND '" +
timeEnd.ToString("HH:mm:ss") + "') AND
(sql7385385.calendar.Date='" + date.ToString("yyyy-MM-dd")
+ "');", conn);
    int tmp1 = Convert.ToInt32(comm.ExecuteScalar());
    conn.Close();
    return tmp1;
}
```

Ha a metódus nullát ad vissza, akkor megtörténik az esemény adatainak frissítése a `DataBaseServices.Save_EVENT()` metódus segítségével. Ami lényegében a beadott *ID* és adatok alapján frissíti a megfelelő rekordot a **calendar** táblában:

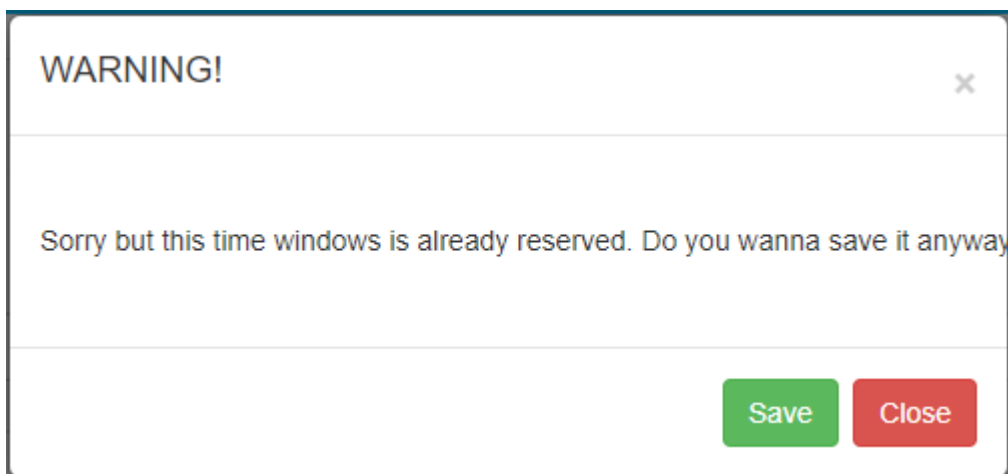
```
public Task Save_EVENT(int ID, DateTime Date, DateTime
TimeFROM, DateTime TimeTO, string Event, string PriorityE,
string Description)
{
```

```

conn.Open();
MySQLCommand comm = new MySQLCommand("UPDATE
sql7385385.calendar SET Date='" + Date.ToString("yyyy-MM-
dd") + "', TimeFROM='" + TimeFROM.ToString("HH:mm") + "',
TimeTO='" + TimeTO.ToString("HH:mm") + "',Event='" + Event
+ "', Priority='" + PriorityE + "', Description='" +
Description + "' WHERE ID=" + ID, conn);
comm.ExecuteNonQuery();
conn.Close();
return null;
}

```

Ha az *ISRES* változó nem egyenlő nullával, akkor arra az idő intervallumra már van esemény foglalva, így meghívja az *Alert\_Modal* JavaScript függvényt, ami megjelenít egy *Bootstrap modal*t egy figyelmeztetéssel, hogy az aktuális időszakra már van esemény mentve.



9. ábra: már foglalt időszáv figyelmeztetése; Forrás: saját kép

A *modal* tartalmaz egy **Save** gombot, amihez *onclcik* eseményhez a *SAVEIT()* függvény van rendelve, ezzel tulajdonképpen a felhasználó elmentheti idő ütközés esetén is az új eseményt és a *sortedEvents* újra megkapja a frissített adatokat az adatbázisból:

```

private void Save()
{
    int result = DateTime.Compare(t.Date,
DateTime.Now.Date);
    if (result > 0)
    {

```

```

        ISRES = DataBaseServices.Is_RESERVABLE(t2.Time2,
t2.Time3, t.Date);
        if (ISRES == 0)
        {
            DataBaseServices.Save_EVENT(t.ID, t.Date,
t2.Time2, t2.Time3, t.Event, t.PriorityE, t.Description);
            StateHasChanged();
        }
        else
        {
            JSRuntime.InvokeVoidAsync("Alert_Modal");
        }
    }
    else
    {
        JSRuntime.InvokeVoidAsync("Date_Modal");
    }
}

```

Ha esetleg törölni szeretnénk az adott eseményt, akkor a *modal* -on belüli **Remove** gombra kattintva, meghívódik a `Delete()` függvény, amiben meghívódik a `DataBaseServices.Delete_EVENT(t.ID)` metódus a megfelelő törölni kívánt eseményhez tartozó ID-vel. A metódus törli a **calendar** táblából a megfelelő ID-vel rendelkező rekordot:

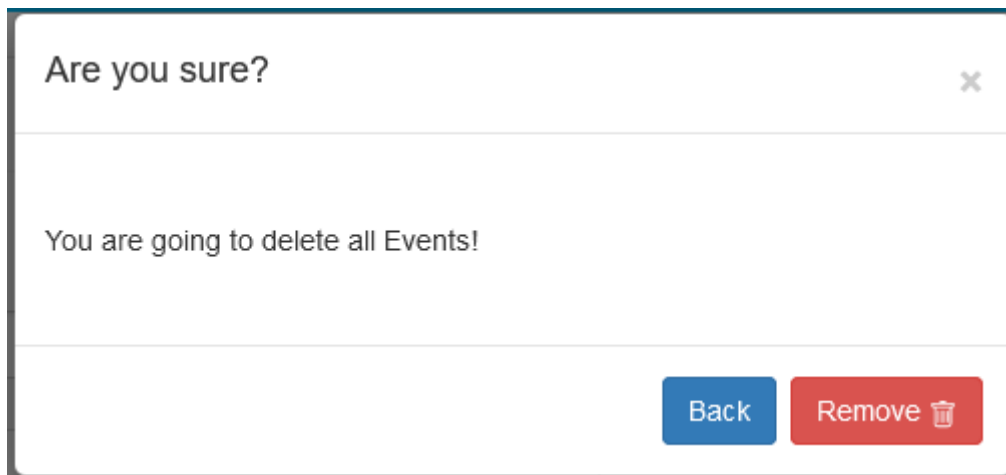
```

public Task Delete_EVENT(int ID)
{
    conn.Open();
    MySqlCommand comm = new MySqlCommand("DELETE FROM
sql7385385.calendar WHERE ID=" + ID, conn);
    comm.ExecuteNonQuery();
    conn.Close();
    return null;
}

```



A felhasználó több száz eseményt is el tud menteni és ezek az események mindaddig az adatbázisban vannak, amíg esetleges adatbázis váltás vagy a **Delete All** gomb megnyomása nem történik.



10. ábra: Összes esemény törlése esetén megjelenő figyelmeztetés; Forrás: saját kép

A **Delete All** gomb *onclick* eseményre meghívja a `Delete_ALL()` függvényt az oldalon és a `DataBaseServices.Delete_EVENTS()` metódus végrehajtja az eseményeket tároló tábla kiürítését:

```
private async Task Delete_ALL()
{
    DataBaseServices.Delete_EVENTS();
    sortedEvents = await DataBaseServices.Get_EVENTS();
}

public Task Delete_EVENTS()
{
    conn.Open();
    MySqlCommand comm = new MySqlCommand("TRUNCATE TABLE
sql7385385.calendar", conn);
    comm.ExecuteNonQuery();
    conn.Close();
    return null;
}
```

Ha a felhasználó mégsem szeretné kitörölni az összes eseményt, akkor az események között tud szűrni bármelyik attribútum alapján, ami egy eseményhez tartozik. Ilyen attribútum lehet az esemény neve, leírása vagy esetleg az eseményhez tartozó dátum vagy a foglalt időszak. Az események közti keresést egy JavaScript script oldja meg, ami hozzá van rendelve egy szöveg

típusú beviteli mezőhöz. Amikor betölt az oldal, akkor a `Refresh()` függvényben rögtön meghívódik a `SEARCH()` JavaScript függvény:

```
function SEARCH() {  
    var input, filter, found, table, tr, td, i, j;  
    input = document.getElementById("filter");  
    filter = input.value.toUpperCase();  
    table = document.getElementById("EventTable");  
    tr = table.getElementsByTagName("tr");
```

A függvényben kezdetben a változók deklarálása történik, illetve egyes változók kapnak is értéket. Az *input* változó megkapja a beviteli mezőt az *ID*-je alapján. A *filter* változó a beviteli mező értékét kapja meg, amit át is alakít nagybetűsre. A *table* változó az eseményeket tartalmazó táblát kapja meg az *ID*-je alapján. A *tr* változó pedig megkapja az összes `<tr>` tages HTML komponensét:

```
for (i = 0; i < tr.length; i++) {  
    td = tr[i].getElementsByTagName("td");
```

A változók deklarálása és az érték adása után egy *for* ciklus kezd el futni nullától egészen addig, amíg az *i* változó értéke kisebb, mint a *t* hossza. A ciklusban a *t* változó kap értéket, az aktuális sorban levő *td*-ket kapja meg:

```
for (j = 0; j < td.length; j++) {  
    if  
(td[j].innerHTML.toUpperCase().indexOf(filter) > -1) {  
        found = true;  
    }  
}
```

Miután a *td* változó megkapta az értékét, elkezd futni egy *for* ciklus *j* egyenlő nullától egészen addig, amíg a *j* változó értéke kisebb, mint a *td* hossza. A cikluson belül megtörténik egy feltétel vizsgálat. Ha a *td* aktuális értéke nagybetűkre alakítva tartalmazza a *filter* változó értékét, akkor a *found* változó *true* értéket kap. Ha nem tartalmazza, akkor a feltétel *-1*-es értéket küld vissza és ekkor nem történik semmi:

```
if (found) {  
    tr[i].style.display = "";  
    found = false;  
} else {  
    tr[i].style.display = "none";
```

```
}
```

Ha a feltétben a *found* változó *true* értéket kapott, akkor azok a sorok, amikben megtalálható a *filter* változó értéke, azok megjelennek a felhasználó számára és a *found* változó értéke átállítódik *false*-ra, és az első *for* ciklus tovább lép a tábla soraiban. Ha a *found false* értékkel érkezik meg ehhez a feltétel vizsgálatához, akkor az aktuális táblasor láthatatlanná válik a felhasználó számára és az első *for* ciklus egy lépéssel tovább lép.

Az események, miután betöltődik az oldal, az adatbázisban mentett *ID*-jük alapján vannak sorba rendezve a táblázatban. Viszont, ha más attribútum alapján szeretnénk rendezni a táblát, akkor azt is megtehetjük. A tábla az események oszlop címe alapján, az oszlop címekre történő kattintás után növekvő vagy csökkenő sorrendbe rendezhető:

```
<tr>
    <th @onclick="@ ( () => SORT ("ID")) ">#<i class="fa fa-
filter" style="font-size:20px"></i></th>
    <th @onclick="@ ( () => SORT ("Date")) ">Date<i class="fa
fa-filter" style="font-size:20px"></i></th>
    <th @onclick="@ ( () => SORT ("TimeFROM")) ">Time FROM<i
class="fa fa-filter" style="font-size:20px"></i></th>
    <th @onclick="@ ( () => SORT ("TimeTO")) ">Time TO<i
class="fa fa-filter" style="font-size:20px"></i></th>
    <th @onclick="@ ( () => SORT ("Event")) ">Event<i class="fa
fa-filter" style="font-size:20px"></i></th>
    <th @onclick="@ ( () =>
SORT ("Description")) ">Description<i class="fa fa-filter"
style="font-size:20px"></i></th>
    <th></th>
</tr>
```

A táblázat minden oszlopának címéhez *onclick* eseményéhez meg van hívva a *SORT()* függvény, ami bemeneti értékként megkapja, hogy az eseményeket melyik oszlop alapján kell növekvő vagy csökkenő sorrendbe rendezni. Ahhoz, hogy a *SORT()* függvény működni tudjon, először is kellett két változó, egy logikai típusú *IsSortedAscending* és egy szöveg típusú *CurrentSortColumn* változó. Mikor a függvény meghívódik, elindul egy feltétel vizsgálat. Ha a bementi oszlop neve nem egyezik meg a *CurrentSortColumn* változó értékével, akkor a *sortedEvents* tömb értékei az aktuális oszlop alapján növekvő sorrendbe lesznek sorolva. Ezt

követően a *CurrentSortColumn* megkapja értéként a bemeneti oszlop nevét, illetve az *IsSortedAscending* változó értéke igazra (*true*) állítódik:

```
if (columnName != CurrentSortColumn)
{
    sortedEvents = sortedEvents.OrderBy(x =>
x.GetType().GetProperty(columnName).GetValue(x,
null)).ToList();
    CurrentSortColumn = columnName;
    IsSortedAscending = true;
}
```

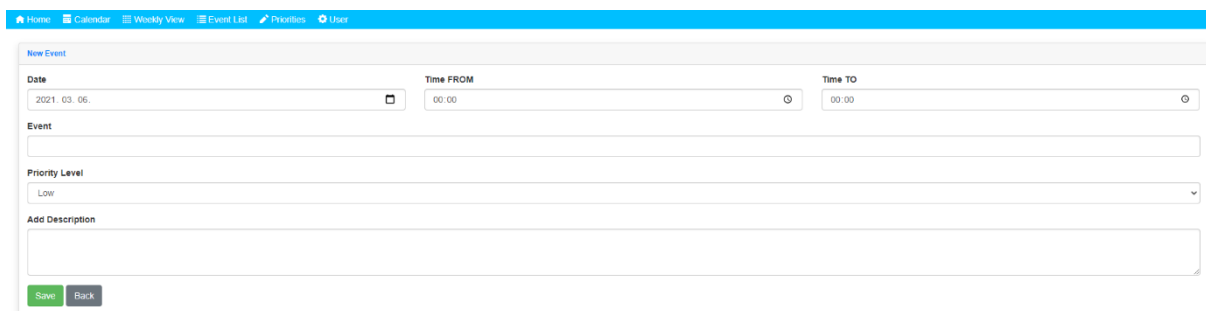
Ha a feltételvizsgálat hamis értéket ad vissza, akkor elindul az *if* függvény *else* ága. Ha a kód eljut ide, akkor megint egy feltétel vizsgálat következik, hogy az *IsSortedAscending* igaz-e. Ha igaz akkor a *sortedEvents* tömb csökkenő sorrendbe lesz rendezve, ha nem, akkor növekvő sorrendbe. Végül, ha a feltétel vizsgálat befejeződött, az *IsSortedAscending* változó megkapja aktuális értékének negáltját:

```
else
{
    if (IsSortedAscending)
    {
        sortedEvents =
sortedEvents.OrderByDescending(x =>
x.GetType().GetProperty(columnName).GetValue(x,
null)).ToList();
    }
    else
    {
        sortedEvents = sortedEvents.OrderBy(x =>
x.GetType().GetProperty(columnName).GetValue(x,
null)).ToList();
    }
    IsSortedAscending = !IsSortedAscending;}
```

(Murach, 2019; Himschoot, 2020; Matsinopoulos, 2020; Murach, 2020)

### 3.8 Új esemény hozzáadása

Ha a felhasználó új eseményt szeretne létrehozni, akkor az eseményeket listázó oldalon rá kell kattintania az **Add Event** gombra, ami átirányítja őt az **AddEvent** oldalra.



11. ábra: Új esemény hozzáadása oldal megjelenése; Forrás: saját kép

Amikor az oldal elindul, meghívódik az `OnInitializesAsync()` függvény, amiben a *priority DataBase* típusú tömb megkapja a prioritásokhoz kapcsolódó adatokat a `DataBaseServices.Get_PRIORITY()` metódus segítségével.

Az oldal megjelenése nagyon egyszerűen van felépítve. Egy Bootstrap kártyán különböző beviteli mezők vannak, amiben a felhasználó megadhatja az eseményhez tartozó adatokat, mint az esemény neve, leírása, mikor kezdődik és meddig tart, illetve az eseményhez tartozó dátumot. Továbbá a felhasználó egy HTML *select* komponensből kiválaszthatja az esemény prioritási szintjét. A *select*-be a *priority* tömb elemei vannak *foreach* ciklus segítségével felsorolva. Végül a mezők után két gomb kapott helyet, a **Save** gomb, ami *onclick* eseményre meghívja a `Save()` függvényt, illetve a **Back** gomb, ami visszairányítja a felhasználót az **EventList** oldalra.

Ahhoz, hogy a **back-end** el tudja érni az egyes mezőkbe írt adatokat, ahhoz a `@bind` beépített parancs van használva. Minden mező a *DataBase* típusú *t* objektum megfelelő változóját kapja meg a `@bind` segítségével.

A `Save()` függvény ugyanúgy működik, mint az események listáján. Első lépésben ellenőrzi a dátumot és az alapján lép tovább, ami vagy egy dátumot figyelmeztető *modal*, hogy a dátum a múltban van, vagy tovább lép, amiben ellenőrzi, hogy az adott időintervallumra és dátumra van-e foglalás. Ha van foglalás, akkor felugrik egy *modal* figyelmeztetéssel, amin ugyan úgy van egy gomb, amivel idő és dátum ütközés esetén elmentheti a felhasználó az eseményt a `SAVEIT()` függvény segítségével. A függvényben a `DataBaseServices.Save_NEVENT()` metódus oldja meg az esemény mentését, majd a `NavigationManager.NavigateTo(„EventList“)` parancs visszairányítja a

felhasználót az események listájára. Ha pedig probléma nélkül menthető az esemény, vagyis az *ISRES* változó értéke egyenlő nullával, akkor meghívódik a `DataBaseServices.Save_NEVENT()` metódus, ami az *INSERT* paranccsal elmenti az új eseményt a *calendar* táblába.

A mentés után a `NavigationManager.NavigateTo(„EventList“)` parancs visszaküldi a felhasználót az események listájára:

```
public Task Save_NEVENT(DateTime Date, DateTime TimeFROM,
    DateTime TimeTO, string Event, string PriorityE, string
    Description)
{
    conn.Open();
    MySqlCommand comm = new MySqlCommand("INSERT INTO
    sql7385385.calendar
    (Date,TimeFROM,TimeTO,Event,Priority,Description) VALUES
    ('" + Date.ToString("yyyy-MM-dd") + "',''" +
    TimeFROM.ToString("hh:mm") + "',''" +
    TimeTO.ToString("hh:mm") + "',''" + Event + "',''" +
    PriorityE + "',''" + Description + "')", conn);
    comm.ExecuteNonQuery();
    conn.Close();
    return null;
}
```

(Murach, 2019; Himschoot, 2020; Matsinopoulos, 2020; Sharp, 2018)

### 3.9 Prioritások hozzáadása és listája

A **Priorities** oldalon a felhasználó követni tudja az általa megadott prioritásokat egy listában, illetve azokat tudja javítani, frissíteni vagy akár új prioritást is itt tud hozzáadni a

rendszerhez.

ID	Priority	Priority Color	
1	Low		<a href="#">Edit</a>
2	Medium		<a href="#">Edit</a>
3	High		<a href="#">Edit</a>

12. ábra: Prioritások oldal megjelenése; Forrás: saját kép

Az oldal két darab *Bootstrap* kártyából áll. Az első kártyán két beviteli mező van, amikben a prioritás nevét, illetve a hozzá tartozó szín kódot hexadecimális értékben adjuk meg. Mindkét mező a *DataBase* típusú objektum megfelelő változójával van `@bind`-elve. A kártya alján egy **Save** gomb kapott helyet, aminek az *onclick* eseményére meghívódik a `SAVEN()` függvény:

```
protected async Task SAVEN()
{
    DataBaseServices.Save_PRIORITY(t.Priority,
    t.CPriority);
    priority = await DataBaseServices.Get_PRIORITY();
}
```

A függvény a `DataBaseServices.Save_PRIORITY()` metódus segítségével elmenti az új prioritást a **priority** táblába, az `INSERT` adatbázis parancs segítségével. Miután a mentés megtörtént, a *priority* *DataBase* típusú tömb adatai frissítése következik:

```
public Task Save_PRIORITY(string Priority, string
CPriority)
{
    conn.Open();
    MySqlCommand comm = new MySqlCommand("INSERT INTO
sql7385385.priority (Priority, Color) VALUES ('" + Priority
+ "',''" + CPriority + "')", conn);
    comm.ExecuteNonQuery();
}
```

```

        conn.Close();
        return null;
    }

```

A második kártyában egy HTML tábla helyezkedik el, amiben a *priority DataBase* típusú tömb elemei vannak kiírva, egy *foreach* ciklus segítségével. Minden táblasor utolsó oszlopában egy **Edit** gomb helyezkedik el. Ha a felhasználó ráklikkel, akkor a függvényben a *t DataBase* objektum megkapja az aktuális prioritás adatait a *SingleOrDefault()* függvény segítségével, illetve a *currentColor string* típusú változó megkapja a kiválasztott prioritás aktuális színét, és megnyílik egy Bootstrap *modal*. A *modal*-ban a kiválasztott prioritást tudja a felhasználó javítani vagy törölni. A *modal* alján három gomb helyezkedik el, a **Back**, ami minden művelet nélkül bezárja a *modalt*, a **Remove**, ami *onclick* eseményre meghívja a *Delete()* függvényt:

```

private async Task Delete()
{
    DataBaseServices.Delete_SPRIORITY(t.ID);
    priority = await DataBaseServices.Get_PRIORITY();
}

```

A függvény a *DataBaseServices.Delete\_SPRIORITY()* metódus segítségével törli a kiválasztott prioritást a *priority* táblából, a *DELETE* parancs segítségével:

```

public Task Delete_SPRIORITY(int ID)
{
    conn.Open();
    MySqlCommand comm = new MySqlCommand("DELETE FROM
sql7385385.priority WHERE ID=" + ID, conn);
    comm.ExecuteNonQuery();
    conn.Close();
    return null;
}

```

Továbbá a *modal* rendelkezik egy **Save** gombbal is, ami *onclick* eseményre a *SAVE()* függvényt hívja meg:

```

private async Task SAVE()

```



```

{
    DataBaseServices.Save_SPRIORITY(t.ID, t.Priority,
    t.CPriority, currentColor);
    priority = await DataBaseServices.Get_PRIORITY();
}

```

A függvényben a `DataBaseServices.Save_SPRIORITY()` metódus frissíti az UPDATE parancs segítségével azoknak az eseményeknek a prioritási színét a *CPriority* bemeneti értékre, ahol a prioritái szín megegyezik a *curcolor* bementi értékkel. Továbbá frissíti a megfelelő *ID*-vel rendelkező prioritást a bemeneti adatok alapján:

```

public Task Save_SPRIORITY(int ID, string Priority, string
CPriority, string curcolor)
{
    conn.Open();
    MySqlCommand comm = new MySqlCommand("UPDATE
sql7385385.calendar SET Priority='"+CPriority+" WHERE
Priority='"+curcolor+"';UPDATE sql7385385.priority SET
Priority='" + Priority + "', Color='" + CPriority + "'
WHERE ID=" + ID, conn);
    comm.ExecuteNonQuery();
    conn.Close();
    return null;
}

```

Mind a `SAVE()` és mind a `Delete()` függvény továbbá tartalmazza a `priority = await DataBaseServices.Get_PRIORITY()` sort, ami frissíti a *priority* tömb adatait, hogy a felhasználó mindig a megfelelő adatokat lássa.

(Murach, 2019; Himschoot, 2020; Matsinopoulos, 2020; Sharp, 2018)

### 3.10 Felhasználó adatainak beállítása

A napló egyszerre csak egy felhasználót tud kezelni. Nem került beépítésre bejelentkezési illetve regisztrációs rendszer. A felhasználó hozzáadása, illetve annak adatainak módosítása az *User* oldalon történik.

The screenshot shows a web application interface. At the top, there is a navigation bar with links: Home, Calendar, Weekly View, Event List, Priorities, and Users. Below this, there are two main sections. The first section is titled 'Setup - please give your data' and contains a form with input fields for 'FirstName', 'LastName', 'Email', and 'BirthDay'. The 'BirthDay' field is pre-filled with '2021. 03. 06.' and has a calendar icon. A green 'Save' button is at the bottom of the form. The second section is titled 'User data' and contains a table with three columns: 'Name', 'Email', and 'BirthDay'. The table has one row of data: 'Kevin Ollé', 'olle.kevin09@gmail.com', and '1998-6-9'.

13. ábra: Felhasználó adatainak beállítása oldal megjelenése; Forrás: saját kép

Amikor az oldal megnyílik, elindul az `OnInitializedAsync()` függvény, amiben az *user DataBase* típusú tömb megkapja a felhasználó adatait a `DataBaseServices.GetUser()` metódus segítségével. A metódus egy *SELECT* utasítással mindent lekér az *user* adatbázis táblából. Az oldal két *Bootstrap* kártyából épül fel. Az első kártya címe a „**Setup – please give your data**”, és a kártya testében beviteli mezők vannak, amikbe megadhatja a felhasználó az adatait.

Első részben *row form-group* osztály van használva, ami tulajdonképpen egy csoportot alkot az adott sorba. Ezt követi egy *col-12* osztály, ami a teljes szélességébe fogja szétvágni a `<div class="col-12">...</div>` tagek között lévő *DOM* elemeket. Ezt a stílust követik a **FirstName** és **LastName** megadásához használt beviteli mezők. Azaz, itt tudja a felhasználó megadni a keresztnévét, illetve vezetéknévét. A beviteli mezők szöveget várnak beviteli értéként.

A második részben ugyanúgy *row form-group* osztály van használva viszont a *col-12*-es osztály helyett *col-6* van alkalmazva, ezzel elérve, hogy a két beviteli mező egymás mellett legyen. A felhasználó itt tudja megadni az e-mail címét egy szöveg típusú mezőbe, illetve a születési dátumát, dátum típusú mezőbe. Címkék segítségével pedig mutatva van a felhasználónak, hogy melyik mezőbe melyik adatát írja. Ahhoz, hogy a **back-end** el tudja érni a mezőkbe írt adatokat, ahhoz a `@bind` parancs van használva, amikbe a *DataBase* osztályból származtatott *t* nevű objektum egyes változói vannak írva:

```
<div class="row form-group">
  <div class="col-12">
    <label class="uk-form-label">FirstName</label>
    <input type="text" class="form-control"
id="firstname" @bind="t.FirstName" />
  </div>
</div>
```

```

<div class="row form-group">
    <div class="col-12">
        <label class="uk-form-label">LastName</label>
        <input type="text" class="form-control"
id="lastname" @bind="t.LastName" />
    </div>
</div>

<div class="row form-group">
    <div class="col-6">
        <label class="uk-form-label">Email</label>
        <input type="text" class="form-control"
id="email" @bind="t.Email" />
    </div>
    <div class="col-6">
        <label class="uk-form-label">BirthDay</label>
        <input type="date" class="form-control" id="bday"
@bind="t.BirthDay" />
    </div>
</div>

```

Végül, a kártya alján egy **Save** mentés gomb van elhelyezve, ami az *onclick* eseményre meghívja a `SAVE()` metódust:

```
<a class="btn btn-success text-white" @onclick="SAVE">Save</a>
```

A `SAVE()` metódus meghívja a `DataBaseServices.Save_USER()` metódust, ami a felhasználó adatainak mentéséért felel:

```

public Task Save_USER(string Firstname, string Lastname,
string Email, DateTime Bday)
{
    conn.Open();
    MySqlCommand comm = new MySqlCommand("TRUNCATE TABLE
sql7385385.user; INSERT INTO sql7385385.user (Firstname,
Lastname, Email, Birthday) VALUES ('" + Firstname + "', '" +
Lastname + "', '" + Email + "', '" + Bday.ToString("yyyy-
MM-dd") + "')", conn);
}

```

```

        comm.ExecuteNonQuery();
        conn.Close();
        return null;
    }

```

A metódus bemeneti értéként három szöveg típusú változót és egy *DateTime* típusút vár. Ezek jelzik a felhasználó adatait. Maga a metódus először megnyitja az adatbázis kapcsolatot, ezt követi egy `MySQLCommand` deklarálás, ami lényegében maga az adatbázis mentés. Adatok mentése előtt a **user** tábla `TRUNCAT`-elése történik, azaz kitörli az összes adatot a táblából, majd az `INSERT INTO` paranccsal elmenti a megfelelő adatot a megfelelő oszlopba. A `MySQLCommand` deklarálása után a parancs lefuttatása történik az `ExecuteNonQuery()` parancs segítségével. Miután a parancs végrehajtása megtörtént, a metódus lezárja az adatbázis kapcsolat és nulla értékkel visszatér.

A `SAVE` metódus továbbá a mentés után újra lekéri a felhasználó adatait az *user* tömbbe, ekkor már a tömbben az új frissített adatot fognak tárolódni.

Az oldalon levő második *Bootstrap* kártyába a felhasználó adatai jelennek meg. A tábla három oszlopból áll a *Name*, *Email* és *Birthday* oszlopokból. A felhasználó adatait pedig egy `foreach` ciklus írja ki, a megfelelő adatot a megfelelő sorba:

```

<table class="table table-bordered" id="dataTable"
width="100%" cellspacing="0">
    <thead>
        <tr>
            <th>Name</th>
            <th>Email</th>
            <th>BirthDay</th>
        </tr>
    </thead>
    <tbody>
        @if (user != null)
        {
            @foreach (var tmp in user)
            {
                <tr>

```

```
 @tmp.FirstName @tmp.LastName</td>  @tmp.Email</td>  @tmp.Birthday.Year- @tmp.Birthday.Month-@tmp.Birthday.Day </td> </tr> } } </tbody> </table> | | |
```

(Murach, 2019; Himschoot, 2020; Matsinopoulos, 2020; Sharp, 2018)

## 4 Eredmények

Kezdeti tesztelések után a felhasználó felület nagyon áttekinthetőnek bizonyult. Egyszerűen használható, átlátható és mobil eszközökön is egyszerűen használható. A felület megalkotásánál törekedtünk az átláthatóság megtartására, de mégis meghagyni az alapvető nézeteket, amik a projektet egy elektronikus naplóvá teszik. A projekt funkcionalitása teljes mértékben kielégíti az elvárásokat, amik egy elektronikus naplóval szemben felállíthatunk.

### 4.1 Problémák a fejlesztés során

A fejlesztés során folyamatos tesztelésnek volt alávetve a projekt. Folyamatosan jöttek elő kisebb-nagyobb hibák, mint például a *DOM* elemek elhelyezkedésében való csúszások, vagy az, hogy az események szerkesztéséhez megnyíló *modal* rossz eseményt tölt vissza, de ezek a hibák rögtön javításra kerültek.

A legszembeötlőbb hiba a havi nézet oldalon található. Mivel a *Blazor* nem képes direkt módon elérni a *DOM* elemeket, ezért a tábla színezése *JavaScript* segítségével történik. Alapból, amikor betölt az oldal, az összes cella zölden jelenik meg és csak egy másodperc után ugranak be a narancssárga cellák, amik az esemény-foglalásokat jelzik.

Egyetlen egy hiba javítása nem történt meg. Amikor a felhasználó szerkeszteni szeretné valamelyik eseményt, akkor az idő választó mezőbe mindig 00:00 érték töltődik vissza. Ez a probléma is visszavezethető valamilyen szinten a *Blazor* hiányosságára, hogy a *@Bind* paranccsal nem képes átállítani az időválasztó mező értékét.

## 4.2 Projekt publikálása az interneten

Mivel a projekt egy webes applikáció, ezáltal elengedhetetlen, hogy publikálva legyen az interneten. A projekt egy *Blazor* szerver applikáció, ami *MySQL* adatbázist használ. A publikációra a **Heroku** felhő alapú szolgáltatást alkalmaztuk [7]. Maga a publikáció nagyon egyszerűen működik. A **Herokun** történő publikáció első lépésében hozzá kell kapcsolni a **GitHub** repot, ahol a projektet tároljuk, a **Heroku** rendszeréhez, majd ki kell választani a számunkra megfelelő *build packot*, ami jelen esetben **jincod** által fejlesztett *dotnetcore-buildpack* [8] volt. Ezt követően a **Heroku** felületén pontos elérési útvonalat kell adni a projekt fájlhoz, ami *\*.csproj* kiterjesztéssel rendelkezik. Végül a **Build** gomb megnyomásával pillanatok alatt elérhetővé válik a projektünk bárki számára az interneten. A **MySQL** adatbázist pedig a *freemysqlhosting* [9] rendszerében béreltük, az ingyenes változatot, ami 5 MB-nyi helyet biztosít ingyen a felhasználók számára. Tesztelések során a publikált rendszer nagyon hatékonyan működött. Igaz, néha a szerver kifagyott de ez egy kompromisszuma az ingyenes szolgáltatásoknak. Az 5 MB-nyi adatbázis tárhely is elégnek bizonyult.

## 4.3 Továbbfejlesztési lehetőségek

Elsődleges továbbfejlesztési lehetőség lenne, hogy a projekt egyszerre több felhasználót is tudjon kezelni. Ehhez szükség van egy kidolgozott regisztrációs és bejelentkezési rendszerre, ami tudja azonosítani a felhasználókat, illetve így az adatbázist is ki kell majd bővíteni további táblákkal.

Továbbá hasznos lenne egy dinamikus rendszer kifejlesztése, amely megfigyeli a felhasználót, hogy mit mikor csinál és a jövőben automatikusan menti a megszokott időben végrehajtott eseményeket. Ilyenre példa lehet, hogy a felhasználó heteken keresztül bejegyzí, hogy hétfőtől péntekig reggel nyolctól délután négyig dolgozik, de időhiány miatt a következő hétre nem jegyzi be. Ekkor a rendszer visszanézné az eseményeket és a múltban folyamatosan ugyan arra az idő intervallumra mentett eseményeket újra elmentené a felhasználó számára.

## Befejezés

Az elkészült projekt lehetőséget nyújt egy felhasználó számára, hogy az eseményeit naplószerűen vezesse. Ezekhez az eseményekhez leírásokat tudjon megadni, továbbá prioritási szinteket tudjon minden eseményhez hozzárendelni. A projekt továbbá hatékony és átlátható grafikus felületet biztosít a felhasználó számára, ahol órára, sőt percre pontosan tudja követni az elmentett eseményeit. Az eseményeket tudja követni heti havi vagy akár az aktuális napra szétírva. Az eseményeket továbbá egy oldalon felsorolva is tudja böngészni, javítani, keresni az események közt.

A továbbfejlesztési lehetőségek megvalósítása után a projekt még hatékonyabb lehet, mivel több felhasználót is ki tud majd szolgálni és így nem csak egy személyre lesz szabva a rendszer.

A felhasznált technológiák behatóbb ismerete révén nagyobb betekintést nyert a szerző a webes applikációk működésébe, és ezen tudás alapján a további kitűzött fejlesztési célok elérése rövid időn belül megvalósulhat.

## Resumé

V prých kapitolách bakalárskej práce je zhrnutie poznatkov o elektronických denníkoch, o použitých technológiách počas vývoja, o budovaní databázy, respektíve o zrealizovaní backend projektu. Nasledujúce kapitoly sú venované budovaniu a fungovaniu stránok. V poslednej kapitole sú pojednávané problémy objavujúce sa počas vývoja, postup publikácie projektu na internete, respektíve ďalšie možnosti vývoja. Bakalárska práca opisuje vývoj elektronického denníka. V počiatočnej fáze vývoja bolo treba stanoviť postup, podľa ktorého bude práca realizovaná.

Postupy práce a stanovené kritériá boli nasledovné:

1. Prehľadný dizajn na mobilných prostriedkoch a osobných počítačoch.
2. Obsluha pre používateľa je ľahká a jednoduchá.
3. Užitočná funkcionálna.
4. Jednoduché sledovanie udalostí.
5. Možnosť vyhľadávania, filtrovania podľa špecifických parametrov.
6. Možnosť nastavenia upozornenia k jednotlivým udalostiam.
7. Implementácia týždenného a mesačného náhľadu.
8. Užívateľ môže nastaviť prioritu k udalostiam za použitia farieb.

Projekt je webová aplikácia založená na Blazor Framework-u, a na ukladanie dát využíva MySQL databázu. Po dlhšom uvážení sme vybrali realizáciu formou webovej aplikácie preto, lebo je dostupná na akomkoľvek zariadení. Práca bola realizovaná ako serverová aplikácia. Takáto realizácia umožňuje, aby na strane klienta sa vykonávalo iba zobrazovanie a samotný výpočet prebiehal na strane servera. Projekt dodržiava pravidlá objektovo orientovaného programovania. Každá jedna funkcia je pozbieraná v C# triede.

Na zobrazenie boli použité *Bootstrap* a *CSS*. Každé jedno zobrazenie stránky prebieha na základe pravidiel daného Frameworku. V Blazore každý projekt obsahuje zdieľaný priečinok, v ktorej definujeme, kam sa umiestnia jednotlivé Razor komponenty na stránke. Okrem toho projekt obsahuje jeden *\_Host.cshtml* súbor, ktorý volá jednotlivé k zobrazeniu potrebné CSS štýly, respektíve k funkcionálite potrebné JavaScript súbory.

Keď používateľ otvorí projekt, na úvodnej stránke ho vítajú dve Bootstrap karty. Ľavá karta ukazuje aktuálny dátum a čas. Pravá karta na minútu presne ukazuje graficky zobrazené udalosti daného dňa vo forme zoznamu. Back-end stránka je veľmi jednoduchá.



Keď sa načíta stránka, sú volané dve metódy, z ktorých prvá načíta aktuálne udalosti dňa z databázy, druhá načíta dáta používateľa. Keď sa to dokončí, spustí sa časovač, v ktorom sa aktuálny dátum a čas obnovuje, respektíve tu sa volá aj metóda zodpovedná za poslanie e-mailu. Tu sa ukazuje nevýhoda posielanie e-mailu, pretože iba vtedy sa vykoná, ak má používateľ otvorenú webovú stránku. Ďalšie možnosti zlepšenia predstavuje .NET Framework konzolová aplikácia, ktorá beží na serveri súbežne s kalendárom, a každé ráno odošle používateľovi udalosti toho dňa.

Na stránke týždenného náhľadu sa zobrazuje sedem Bootstrap kariet. Každá karta predstavuje deň v týždni. Zaujímavosťou je, že prvá karta ukazuje udalosti nedele, a to preto, lebo v C# jazyku typ „dátum“ má anglickú formu, kde sa týždeň začína nedeľou. Zobrazenie kariet sa riadi rovnakou zásadou ako na úvodnej stránke, t.j. udalosti sú graficky zobrazené vo forme zoznamu, pričom každá udalosť sa umiestni do stĺpca podľa jej prislúchajúcemu dátumu.

V *back-end-e* sa volá iba jedna metóda, ktorá dopytuje udalosti z databázy na základe dátumu aktuálneho týždňa. Počas zobrazenia každá udalosť sa umiestni do príslušného stĺpca na základe dátumu.

Na stránke mesačného náhľadu je typické zobrazenie kalendára. Dve tlačidlá napomáhajú v listovaní mesiacov. Zobrazenie dní v mesiaci sa deje v HTML tabuľke. Každá bunka tabuľky predstavuje deň v mesiaci. Táto stránka presne poukazuje na nevýhodu Blazoru, že nie je schopný priamo pristupovať k HTML elementu. Tieto nedostatky sme odstránili pomocou JavaScript-u. Každá bunka má vlatné *ID*, ktoré je dané pomocou dátumu.

Po spustení stránky *back-end* zbehne metóda, ktorá si vypýta všetky udalosti z databázy. Po uskutočnení tohto dopytu sa všetky dátumy zapisujú do jednej premennej typu string, ktorú následne na stránke spracuje JavaScript. Script v prvom kroku rozbalí premennú, aby získal jednotlivé dátumy. Ak dátum a *ID* niektorej bunky sa zhodujú, tak túto bunku zafarbí na oranžovo, čím upozorní používateľa, že v daný deň má nejakú udalosť. Keď používateľ klikne na niektorú bunku, tak sa otvorí *modal*, ktorý zobrazí existujúcu udalosť.

Používateľ vie sledovať udalosti na stránke zoznamu udalostí. Tu sú udalosti vymenované v *Bootstrap* tabuľke. Farba riadkov tabuľky sa mení podľa priority daných udalostí. V tabuľke sú vymenované udalosti podľa ich *ID*, pod ktorým sú uložené v databáze. Po kliknutí na hociktorú hlavičku stĺpca tabuľky sa udalosti zoradia zostupne alebo vzostupne, podľa daného stĺpca. Každý riadok tabuľky obsahuje jedno tlačidlo **Edit**, ktoré po kliknutí používateľom otvorí *modal*, ktorý umožňuje zmenu atribútov udalostí alebo zmazať danú udalosť, a nakoniec stlačením tlačidla **Save** uložiť udalosť.

Program pred uložením kontroluje dátum, či nie je z minulosti. Ak áno, oznámi to používateľovi pomocou *modalu*. Ak je dátum z budúcnosti, tak program ďalej kontroluje, či vo zvolenom intervale už existuje udalosť. Ak je interval obsadený, tak používateľ je upozornený. To je už na používateľovi, či uloží prekrývajúcu sa udalosť alebo zmení čas. Nad tabuľkou sa nachádza pole na vyhľadávanie, ktoré umožňuje používateľovi vyhľadávanie medzi udalosťami. Na stránku sa dostali aj ďalšie dve tlačidlá. Prvé slúži na zmazanie všetkých udalostí. V prípade, že používateľ stlačí toto tlačidlo, tak sa zobrazí *modal* na overenie, či naozaj chce zmazať všetky udalosti. Druhé tlačidlo presmeruje používateľa na druhú stránku, kde vie pridať novú udalosť k ostatným udalostiam v systéme.

Keď chce používateľ pridať novú udalosť v systéme, tak musí stlačiť **Add Event** tlačidlo na stránke zoznamu udalostí, čo ho presmeruje na novú stránku, kde môže pridať novú udalosť. Používateľa na stránke víta *Bootstrap* karta, na ktorej sa nachádza viacero vstupných políček. Tu vie používateľ zadať prislúchajúci dátum, časový rozsah udalosti, názov a popis udalosti. Ďalej sa dá pomocou rolovacieho menu zvoliť úroveň priority udalosti. K udalosti priradená priorita slúži hlavne na zjednodušenie zobrazovania, keďže udalosti budú mať rozličnú farbu pozadia. Ak používateľ vyplnil všetky polia, stlačením tlačidla **Save** uloží novú udalosť. Pred uložením program ešte skontroluje dátum, či nie je z minulosti. Ak áno, oznámi to používateľovi pomocou *modalu*. Ak je dátum z budúcnosti, tak program ďalej kontroluje, či v zvolenom intervale už existuje udalosť. Ak je interval obsadený, tak je používateľ nato upozornený. To je už na voľbe používateľa, či uloží prekrývajúcu sa udalosť alebo zmení čas.

Používateľ môže priradiť ku každej udalosti úroveň priority. Tieto priority vie sám zobrazovať v systéme. Na stránke priority sú dve *Bootstrap* karty. V hornej karte vie používateľ zadať názov a farbu priority, a uložiť ju stlačením tlačidla **Save**. Na spodnej karte sú v zozname vymenované uložené úrovne priority. V poslednom stĺpci zoznamu stlačením tlačidla **Edit** môže používateľ upraviť uloženú prioritu.

Systém dokáže zatiaľ obslúžiť iba jedného používateľa. Používateľ musí nastaviť v systéme svoj e-mail a dátum narodenia. Toto vie vykonať na osobitnej stránke. E-mail adresa, ktorú používateľ nastaví, bude použitá na odoslanie oznámenia o dennej udalosti. Projekt otvára možnosť implementácie prihlasovacieho a registračného systému, pomocou ktorého sa budú dať rozlíšiť údaje jednotlivých používateľov.

Posledná kapitola pozostáva z troch podkapitol. V prvej podkapitole sú zhrnuté problémy objavujúce sa počas vývoja. Vyskytli sa menšie aj väčšie problémy. Menšie problémy sa vyskytli kvôli zlému umiestneniu HTML elementov, väčším problémom bola stránka mesačných náhľadov. Bolo treba nájsť nato riešenie, nakoľko Blazor nevie priamo pristúpiť k

HTML elementu. V druhej podkapitole opisujeme, akým spôsobom bol projekt publikovaný na internete. Na publikáciu projektu bol použitý bezplatný balíček **Heroku**, databáza bola prenajatá zo stránky **Free MySQL Hosting**. Tretia podkapitola opisuje ďalšie inteligentné možnosti vývoja pre zlepšenie služieb projektu.

## Szakirodalom

### Könyv alapú források:

- Litvinavicius, T. (2019). *Exploring Blazor: Creating Hosted, Server-side, and Client-side Applications with C#* (1st ed.). Apress, 2019, 210 p. ISBN 978-1-4842-5446-2.
- Himschoot, P. (2020). *Microsoft Blazor: Building Web Applications in .NET* (2nd ed.). Apress, 2020, 303 p. ISBN 978-1-4842-5927-6.
- Troelsen, A. and Japikse, P. (2020). *Pro C# 8 with .NET Core 3: Foundational Principles and Practices in Programming* (9th ed.). Packt, 1281 p. ISBN 978-1-4842-5755-5.
- Murach, J. (2019). *Murach's MySQL* (3rd ed.). Murach, 628 p. ISBN 978-1-9438-7236-7.
- Murach, J. (2020). *Murach's JavaScript and jQuery* (4th ed.). Murach, 752 p. ISBN 978-1-9438-7262-6.
- Matsinopoulos, P. (2020). *Practical Bootstrap: Learn to Develop Responsively with One of the Most Popular CSS Frameworks* (1st ed.). Apress, 516 p. ISBN 978-1-4842-6070-8.
- Sharp, J. (2018). *Microsoft Visual C# Step by Step (Developer Reference)* (9th ed.). Microsoft Press, 832 p. ISBN 978-1-5093-0776-0.

### Internetes források:

- [1] <https://asana.com/?noredirect> (letöltve: 2020.12.05)
- [2] <https://www.scrum.org/resources/what-is-scrum> (letöltve: 2021.04.03)
- [3] <https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor> (letöltve: 2020.12.05.)
- [4] <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html> (letöltve: 2020.12.05.)
- [5] <https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.ihostedservice?view=dotnet-plat-ext-5.0> (letöltve: 2020.04.04.)
- [6] <https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.componentbase.oninitializedasync?view=aspnetcore-5.0> (letöltve: 2020.10.09.)
- [7] <https://www.heroku.com/> (letöltve: 2020.12.05)
- [8] <https://github.com/jincod/dotnetcore-buildpack> (letöltve: 2020.12.05)
- [9] <https://www.freemysqlhosting.net/> (letöltve: 2020.12.05)

## Mellékletek

CD:

- a) Szakdolgozat PDF verziója (OlleKevin.pdf)
- b) Forráskódokat tartalmazó fájlok (Forráskódok mappa)
- c) Adatbázis működéséhez szükséges fájlok (Adatbázis mappa)
- d) Link a webes applikáció eléréséhez: <https://calendarbc.herokuapp.com/>