**C# Asztali Alkalmazás**

**Vizsgaremek 2022**

**Sipos Ferenc**

Az applikáció egy étterem belső működésének megkönnyítéséhez lett létrehozva.

Adatbázisban (MySQL) lévő adatok lekérésére egy (RESTful) ASP.NET Core Web API-t használ, HttpClient segítségével.

Felhasznált technológiák:

* WPF
* MVVM (Model-View-ViewModel) Architektúra
* .NET 5
* Material Design (XAML)
* Entity Framework Core

**Rövid leírás:**

**.NET:**



**MySQL:** Egy nyílt forráskódu, ingyenes, relációs adatbázis kezelő rendszer, SQL nyelvre alapozva. Kezelése egyszerű, jól teljesíti a neki szánt feladatokat.

**ASP.NET Core:** Egy nyílt forráskódu, ingyenes, Web keretrendszer, az eredeti ASP.NET-nek a továbbfejlesztett változata. Ez mostmár magába foglalja a korábban külön fejlesztett ASP.NET MVC Web alkalmazás keretrendszert, és az ASP.NET Web Api-t is.

Az applikáció által használt Web API ezen alapszik, ez egy HTTP alapú kiszolgáló ami adatokat ad vissza a (mi esetünkben) MySQL szerverből, különböző végpontok meghívását követően.

**REST:** Egy architekturális módszer/felépítés, melynek ha egy API eleget tesz, akkor hívhatjuk RESTful API-nak. Az alapvető megkötése az, hogy az API nem tárolhat semmilyen státuszt/adatot a felhasználókról, hanem minden szükséges információt (ki a küldő, mit szeretne módosítani) egy HTTP kérés kell tartalmazzon (levéve a terhet az API-ról), ezáltal egy RESTful API könnyedén ki tud szolgálni egyszerre több millió felhasználót is.

**WPF:** Egy nyílt forráskódu, grafikus felhasználói felületek készítéséhez használatos osztálykönyvtár, hasonló a WinForms-hoz. Legnagyobb különbség köztük hogy ez XAML leírónyelvet használ a felhasználói felületek kialakításához, ami nagy hangsúlyt fektet a felhasználói felület és az üzleti és megjelenítési logika elkülönítésére.

**MVVM (Model-View-ViewModel):** Egy olyan programtervezési minta amely a felhasználói felületet (**V**iew) különíti el teljesen az üzleti és megjelenítési logikától. A felhasználói felület “Databinding”-al kötődik az elkülönített megjelenítési logikához(**V**iew**M**odel), amely különböző adatszerkezeteket és **M**odellek egy egy példányát tartalmazza.

A felépítés előnyei:

* Egy applikáción belül teljesen el van különítve az UI és a Model, ha a Model logikájának változnia kell, ez megtörténhet anélkül hogy az UI-hoz hozzányúlnánk.
* XAML alapú könyvtárak/keretrendszerek (WPF,UWP,WinUI,MAUI,Xamarin stb..) alapból támogatják, sőt, köré vannak tervezve.
* Egy applikáción belül dolgozhat UI/UX designer csak az UI-n, akinek nem kell értenie a Backend részhez, ez fordítva is igaz.(Databinding teszi lehetővé)
* Kétirányú Databinding miatt nem kell mindig újra és újra manuálisan frissíteni az UI-t, amint a Databindinghoz használt tulajdonság frissült, a ViewModel szól erről a változásról, így az UI le tudja frissíteni önmagát.
* UI-n lévő vezérlők be és kikapcsolását a ViewModel(Megjelenítési logika) vezérli, általában egy Model segítségével, pl.egy egyszerű “CanExecute” methodus megírásával vezérelhető hogy egy gomb kattintható legyen, vagy ne.
* Mivel az applikáció minden része külön külön “modul”-okból áll, sokkal könyebben újra felhasználható minden része, és sokkal könyebben tesztelhető mert nem függenek egymástól.
* Ha a felépítés megfelelően van alkalmazva:
  + Egy model nem tartalmaz semmilyen referenciát a View vagy ViewModel-re.
  + Egy ViewModel nem tartalmaz semmilyen referenciát a View-ra.
  + Egy View csak Databinding-al kapcsolódik a ViewModelhez, és az UI mögötti kód csak és kizárólag UI-hoz kapcsolódó kódot tartalmaz.

Akkor az alkalmazás akármelyik része egyszerűen módosítható, külön-külön is.

A felépítés hátrányai:

* A koncepció elsőre nehéznek tűnik, egyszerűbb alkalmazásoknál nem feltétlenül a legjobb választás mert több időbe telhet felépíteni az alkalmazás vázát mint megírni a logikáját.
* Nem annyira elterjedt mint a társai (MVC,MVP) , leginkább XAML-t használó keretrendszerekbe/könyvtárakba (WPF,UWP,WinUI,MAUI,Xamarin stb...) van implementálva alapból a támogatása.
* Hibás felépítés esetén akár hátráltathatja is a fejlesztést.

**.NET 5:** Egy nyílt forráskódú fejlesztési keretrendszer/platform (régebben .NET Core), ami a .NET Framework-öt váltotta le. Támogatott programozási nyelvek (5-ös verziónál):

* C# 9.0 (vagy kisebb)
* F# 5.0 (vagy kisebb)
* VB.NET 16.0 (vagy kisebb)

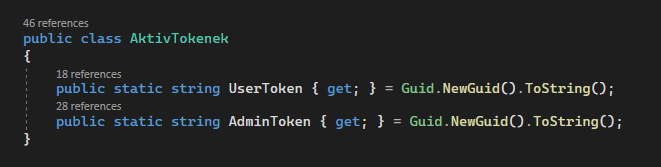
**Material Design:** Egy ingyenes,nyílt forráskódú felhasználói felület könyvtár, egyik legismertebb grafikus felület csomag WPF-hez. Az projecten belül NuGet csomagként van implementálva.

**Entity Framework Core:** Egy nyílt forráskódú adat hozzáférésre készített technológia. Az API ezzel végzi el az adatbázisban az utasításokat. Objektumokkal/Modellekkel lehet dolgozni, amit az Entity Framework átalakít SQL utasításokká és elvégzi azokat.

**Fejlesztői dokumentáció:**

**API:**

**Authorizáció:**



* Amint az API elindul, 2 statikus property-be generál 1-1 GUID-t.
* Ez nem lesz egyedi azonosító, csupán arra szolgál hogy ne tudjon nem bejelentkezett személy adatokat lekérni.
* Sikeres bejelentkezésnél a felhasználó joga alapján átküldjük neki a megfelelő GUID-t.
* Az endpointokban header-be várjuk “auth:” néven a tokent amelyikeket le akarjuk védeni.
* Minden más azonosító (hogy ki kéri, mit változtasson) az objektumokon/modelleken található amit a felhasználó kliense küldeni fog.

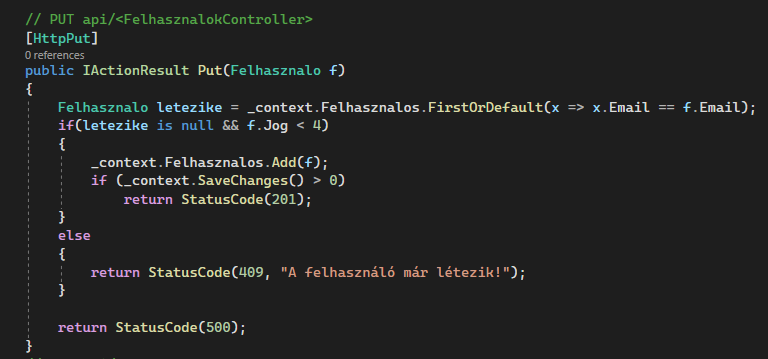
**Kontrollerek:**

Egy kontroller élettartalma egy HTTP kérés teljesítéséig tart. Minden alkalommal amikor egy felhasználó/kliens meghív egy endpointot([HttpGet],[HttpPut],[HttpPost] stb..) valamelyik kontrollerbe:

* Az API felépít egy új kontrollert
* A kontrolleren belül készül egy új DBContext(esetünkben burgeretteremContext) ami tartalmazza az adatbázisunk legfrisebb adatait.
* Egy DBContext-et általában egy kéréshez ajánlott használni.
* A DBContext adatait módosítja (új felhasználó hozzáadása, felhasználó adat módosítás) és elmentjük a változtatásokat rajta.
* Ezután alakítja át az Entity Framework a módosításokat SQL parancsokká és hajtja végre azokat.

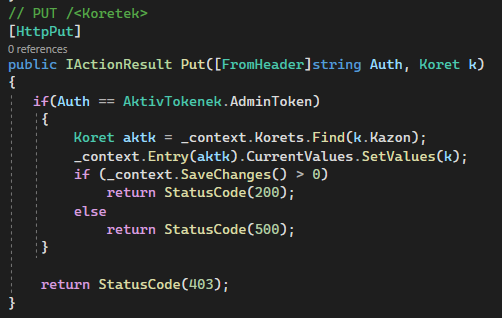
**Nem szükséges Auth (Regisztráció)**





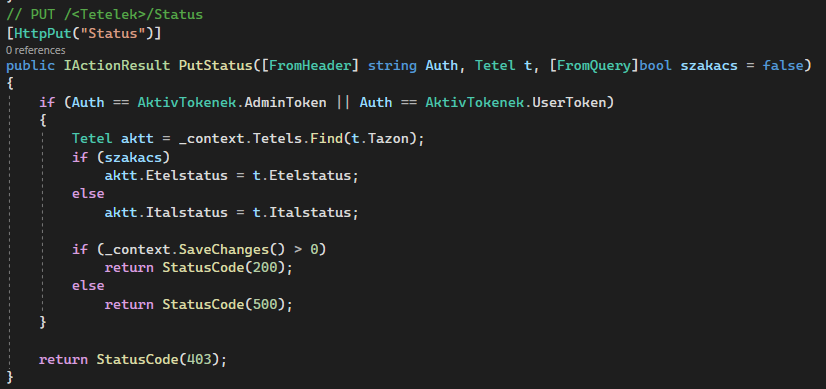
* IActionResult-al tér vissza (a többi endpoint is), ami lehet StatusCodeResult vagy ObjectResult.
* Ez az endpoint nem vár authorizációs stringet a headerben, mert ezt fogják használni a még be nem jelentkezett felhasználók.(Az admin regisztrációra külön endpoint van ami már kéri az admin GUID-ot.)
* A kérés body-jából vár egy Felhasznalo típusú objektumot (JSON formában), az ASP.NET Web Api automatikusan átalakítja a bejövő JSON-t .NET objektumokká, és a kimenő objektumokat JSON-é.
* A DBContext-ünk (burgeretteremContext) felhasználói között megnézzük van-e ilyen email-el rendelkező felhasználó.
* Ha nincs és a paraméterben kapott felhasználó joga kisebb mint 4 (szóval nem admin)
* Hozzáadjuk a DBContext felhasználói közé és elmentjük a változtatásokat ami visszatér a változtatott adatmennyiséggel.
* Ha több mint 0, szóval változott a DB-ben az adat, akkor visszatérünk egy 201-es státusszal (Created).
* Ha már létezik ilyen felhasználó akkor visszatérünk egy 409es státusszal(Conflict).
* Ha akármi más történik (pl. Mentésnél 0 a visszatért érték, szóval nem változott semmi) akkor visszatérünk egy 500as státusszal(Internal Server Error).

**Admin szükséges (Köret módosítása)**



* IActionResult-al tér vissza (a többi endpoint is), ami lehet StatusCodeResult vagy ObjectResult.
* Ez az endpoint vár Authorizációs stringet a headerből, amit leellenőrzünk az API futtatásakor generált Admin GUID-vel.
* A body-ból vár egy köretet JSON formátumban.
* Ha az Auth token megegyezik a generált Admin tokennel.
* Megkeressük a DBContext köretjei között a módosítani kívánt köretet (a kapott köret azonosítója alapján).
* Átállítjuk minden értékét a kapott köretére.
* Elmentjük a változtatásokat, ha több mint 0, szóval változott a DB-ben az adat, akkor visszatérünk 200as státusszal(OK).
* Ha nem több mint 0 akkor visszatérünk 500as státusszal(Internal Server Error).
* Ha a kapott token nem egyezik meg a generálttal akkor visszatérünk egy 403as státusszal(Forbidden).

**Admin vagy User szükséges (Tétel státusz módosítása)**

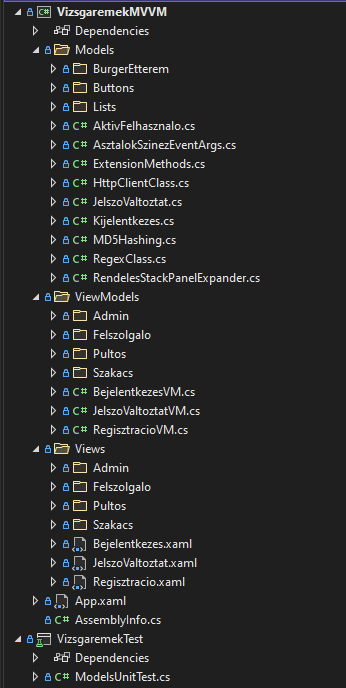


* IActionResult-al tér vissza (a többi endpoint is), ami lehet StatusCodeResult vagy ObjectResult.
* Ez az endpoint vár Authorizációs stringet a headerből, amit leellenőrzünk az API futtatásakor generált GUID-kkel.
* A bodyból egy tételt vár JSON formátumban.
* Queryből (?szakacs=) ha kaphat esetleges true értéket, ha nem kap akkor a default false.
* Összehasonlítjuk a kapott Auth stringet a generált GUID-kkel, ha valamelyikkel egyezik akkor elvégezzük a továbbiakat.
* A DBContext tételei közül megkeressük a módosítani kívánt tételt(a kapott tétel azonosítójából).
* Ha queryből a szakács át lett állítva true-ra akkor:
  + A tétel ételstátuszát változtatjuk
* Ha nem lett:
  + A tétel italstátuszát változtatjuk
* Elmentjük a változtatásokat, ha több mint 0, szóval változott a DB-ben az adat, akkor visszatérünk 200as státusszal(OK).
* Ha nem több mint 0 akkor visszatérünk 500as státusszal(Internal Server Error).
* Ha a kapott token nem egyezik meg a generáltakkal akkor visszatérünk egy 403as státusszal(Forbidden).

**Az API Controllerjei/Endpointjai mind hasonlóan működnek.**

**WPF Alkalmazás**

**A project felépítése**

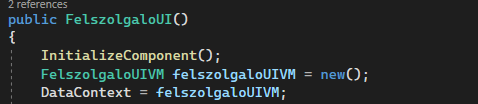


* **Views:** Itt található minden ami a grafikus/felhasználói felületért felel, nem tartalmaz semmilyen üzleti logikát.
* **ViewModels:** Minden View-hoz tartozik egy ViewModel, ezek a View absztrakciói, ez tartalmazza a megjelenítési logikát, és Property-ket amikhez a View Databindol.
* **Models:** Ezek tartalmazzák az üzleti logikát, úgymond építőelemei az applikációnak, amiket a ViewModelbe rakosgatunk általában Propertyként hogy Databindolható legyen.

**Példa az applikáció működésére**

**View(FelszolgaloUI):**

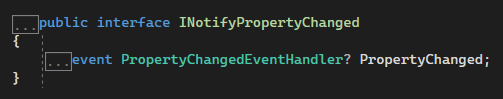
**Ablak mögötti kód:**

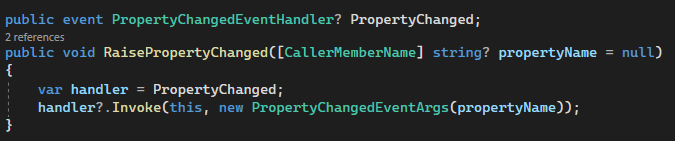


* Készítűnk a ViewModeljéből egy példányt
* Beállítjuk az ablak DataContext-jének
* Ezek után minden megjelenítési logika a VM-ben történik
* Ablak mögötti kód csak UI-val kapcsolatos kódot tartalmaz

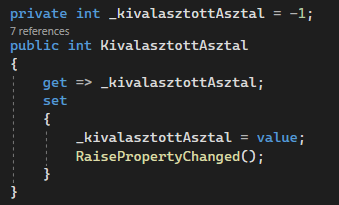
**ViewModel(FelszolgaloUIVM):**



* Egy ViewModel-t származtatni kell az INotifyPropertyChanged Interface-ből.
* Ez az Interface fog felelni az értesítésekért, amikor egy Property értéke változik.
* Az Interface egy PropertyChangedEventHandler-t kér implementálni 
* Amint ezt beraktuk a ViewModelünkbe, írunk egy eljárást amivel ezt az EventHandlert meg tudjuk hívni, és PropertyChangedEventArgs-ban átadni a Property nevét, aminek az értéke változott.



* Ezek után létre tudjuk hozni a Propertyket amik kellenek a View-hoz, a Property publikus kell legyen , és kell hozzá egy mező is (általában privát,ebben az esetben alapértelmezett értéke -1).

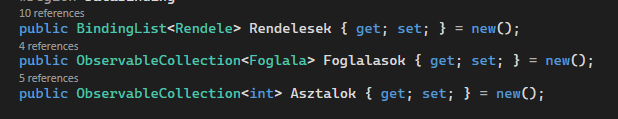


* Amikor a Property-ből adatot kér ki a View, az vissza fogja adni a mező értékét, és amikor változtatás történik akárhonnan akkor beállítja azt ÉS meghívja az előbb készített PropertyChanged eljárásunkat, ami jelez a Viewnak (vagy akárminek ami hallgat erre az EventHandler-re) hogy változott az érték, szóval kikéri a Property értékét.
* A RaisePropertyChanged meghívásánál nincs átadva semmi, ez az eljárás [CallerMemberName] miatt van, amivel kinyerhetjük automatikusan a Property nevét ami meghívta.
* A View-ban a DataBinding így néz ki, egy ComboBox SelectedValue-ja van hozzá bindolva.



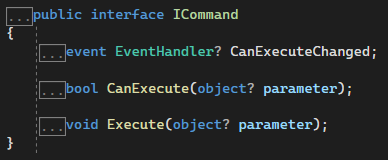
* Egy DataBinding lehet kétirányú, egyirányú(View->Kód) , egyirányú a kód felől vagy egyszeri.

**Listák databindinghoz**



* **2** fajta lista van használva az applikációban DataBindinghoz.
* A binding lehetséges mert mindkét fajta értesítést küld ha változott a benne lévő adat, azzal a különbséggel hogy:
  + BindingList:
    - IRaiseItemChangedEvents Interface-t tartalmazza, ez azt jelenti hogy akkor is szól a változásokról ha a benne lévő adatok valamelyik tulajdonsága változott(feltéve ha az adatszerkezet tartalmaz valamilyen fajta értesítő Interface-t pl. INotifyPropertyChanged)
  + ObservableCollection:
    - INotifyCollectionChanged Interface-t tartalmazza, ez csak akkor szól a változásokról ha a kollekció tartalma változott(pl. Hozzá lett adva egy új elem vagy törölve lett).

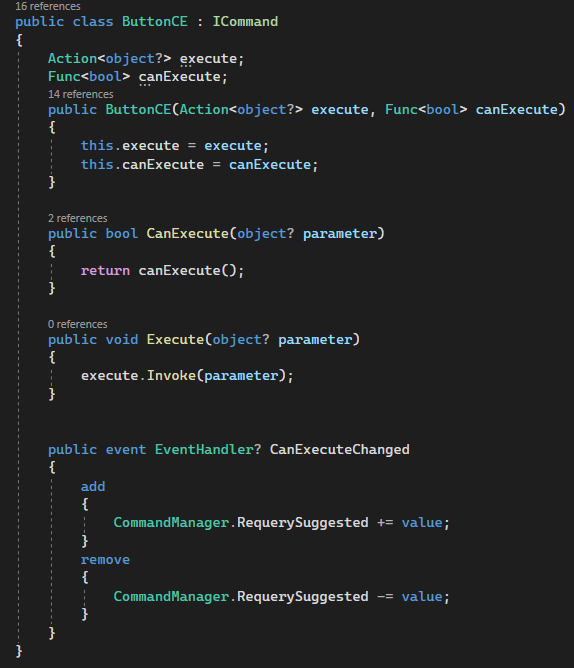
**ICommand Interface**



* Minden Gombot,vezérlőt ami kattintásra elvégez valamit egy olyan Modelhez bind-olunk amit az ICommand Interface-t implementálja.



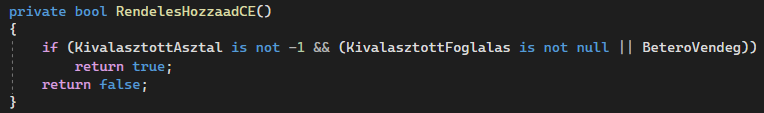
* A ButtonCE Model-ben csak a “váza” van a gombunknak, amikor létrehozzuk a ViewModel-ben nekünk kell funkciót adni neki, hogy mivel legyen létrehozva.



* Két privát adatszerkezet, ezekben fogjuk tárolni a konstruktorból megkapott funkciókat.
* Konstruktor:
  + Kapnia kell egy Action-t(void visszatérés) ami paraméterbe kaphat egy nullable Object-et ÉS nem ad vissza semmit, ez lesz az elvégezendő eljárás gomb kattintásnál.
  + Kapnia kell egy olyan Function-t (bool visszatérés) ami nem kap paraméterbe semmit, ez fogja eldönteni hogy a gombunk kattintható legyen-e.
* CanExecute és Execute funkciók:
  + CanExecute: minden esetben amikor meg akarjuk tudni hogy kattintható-e a gombunk ez fog lefutni, ami az előbb eltárolt funkciónkat fogja meghívni.
  + Execute: gomb/vezérlő kattintásnál ez fog lefutni, ami az előbb lementett void visszatérésű eljárásunkat fogja meghívni.
* CanExecuteChanged EventHandler: amikor ez az esemény meghívásra kerül, ez a modellünk lefuttatja a CanExecute funkciót, és megnézi hogy a gomb kattintható-e. Ezt az EventHandler-t feliratjuk a CommandManager.RequerySuggested eseményére, ami minden alkalommal meghívásra kerül amint a felhasználó kattint,szerkeszt vagy akármit csinál az UI-on ami befolyásolhatja a gombunk kattinthatóságát.



* Ebben az esetben a rendelés felvétele gombunk ButtonCE nevezetű objektum egy példányához van bind-olva, ami tartalmazza az ICommand Interface-t.
* Ez felel azért hogy: kattintásra mi történjen, lehessen-e kattintani a vezérlőt (amit másodpercenként többször is megnéz).
* A gomb példányosításánál átadunk neki paraméterbe 2 funkciót, az első ami kattintásra történni fog, a második az hogy mikor lehessen rákattintani.

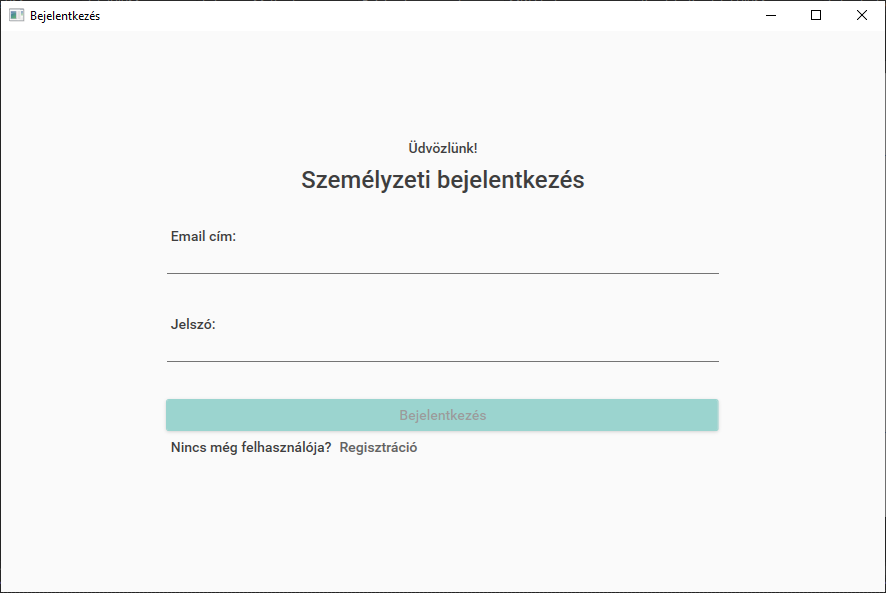


**A ButtonCE példányunkba le van mentve ez a két funkció, akkor lehet rákattintani amikor a RendelesHozzaadCE true-val tér vissza (ez másodpercenként többször is lefut) és kattintásra a RendelesHozzaad fog végrehajtódni.**

**Felhasználói dokumentáció**

Az alkalmazás 4 felhasználói felületet tartalmaz: Felszolgáló, Szakács, Pultos, Admin.

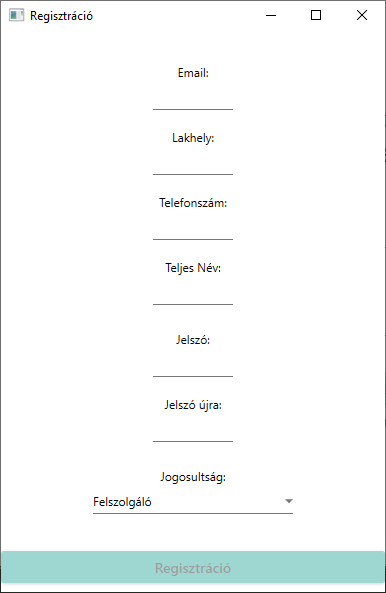
**Bejelentkezés**



A felület részletei:

* Email mező
* Jelszó mező
* Bejelentkezés gomb
* Regisztráció gomb

A Bejelentkezés gomb aktívvá tételéhez mindkét beviteli mezőt ki kell tölteni. A gomb megnyomásával a RESt API lesz meghívva ami sikeres bejelentkezés esetén visszaküldi a bejelentkezett felhasználót. A Regisztráció gombra kattintás során egy új Dialóg ablak nyílik meg.



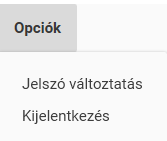
A felület részletei:

* Email mező
* Lakhely mező
* Telefonszám mező
* Teljes név mező
* Jelszó mező
* Jelszó újra mező
* Jogosultság választó
* Regisztráció gomb

A regisztráció gomb aktívvá tételéhez a mezők sikeres kitöltése szükséges. A jelszó és jelszó újra mezőknek meg kell egyeznie. A jogosultságtól függően kapjuk meg később a megfelelő felületet.

Sikeres bejelentkezés után a megfelelő felület fog megjelenni.

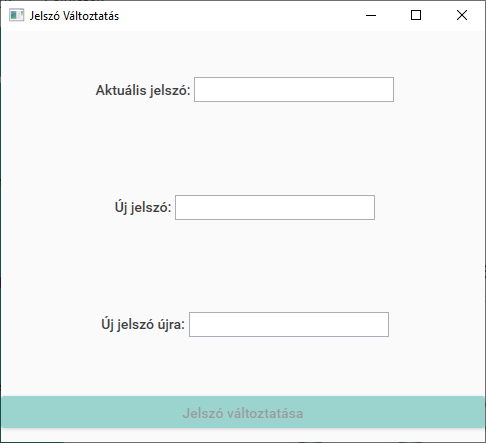
Felső menü:



Balra: Felszolgáló, szakács, pultos felső menüje látható, lehetséges jelszó változtatása, kijelentkezés.

Jobbra: Admin felső menüje látható, tartalmaza az előbbi menüt és egy felületek menüt is. Ebből lehetséges a többi felület megnyitása.

Jelszó változtatása felület:



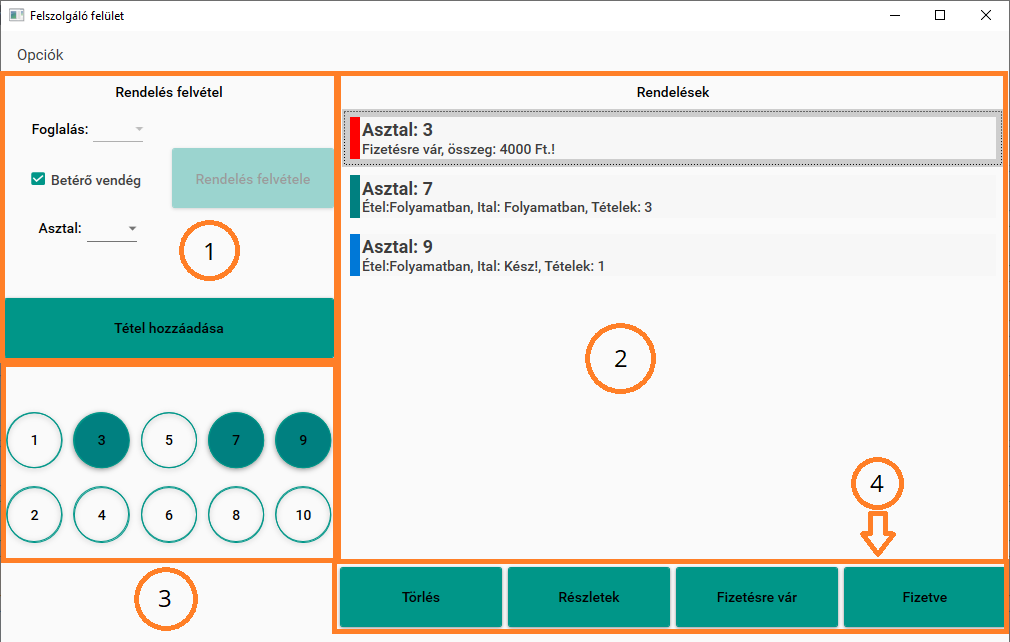
A felület részletei:

1. Aktuális jelszó mező
2. Új jelszó mező
3. Új jelszó újra mező

Jelszó változtatás feltétele hogy a beírt aktuális jelszó megegyezzen a mostani bejelentkezett személy jelszavával.

Ezen felül a két új jelszónak szintén egyeznie kell hogy sikerüljön a jelszó változtatás.

Felszolgáló felület:



A felület részletei:

1. Rendelés Felvétel:
   1. Foglalás választó
   2. Betérő vendég választó
   3. Asztal választó
   4. Rendelés Felvétele gomb
   5. Tétel hozzáadása gomb
2. Rendelések:
   1. Rendelés:
      1. Folyamatban lévő (Kékeszöld jelölő)
      2. Egyik típusú tétel kész (Kék jelölő)
      3. Fizetésre várakozó (Piros jelölő)
3. Asztalok:
   1. Kékeszöld hátterű asztal: Foglalt
   2. Átlátszó hátterű asztal: Szabad
4. Also vezérlő gombok:
   1. Törlés
   2. Részletek
   3. Fizetésre vár
   4. Fizetve

- Egy rendelés státusza:

* Étel: Folyamatban , Ital: Folyamatban:
  + - Ha nincs még kész az összes benne lévő tétel étele / itala
* Étel: Kész , Ital : Folyamatban (vagy fordítva):
  + Ha az összes tétel étele/itala készen van ami a rendeléshez tartozik.
* Felszolgálásra vár! :
  + Ha az összes ital és étel készen van ami a rendeléshez tartozik.
* Fizetésre vár, összeg: \*összeg\* Ft! :
  + Az \*összeg\* az összes benne lévő tétel végösszege összesítve.
  + Ha a rendelés státusza manuálisan “Fizetésre vár”-ra lett változtatva a lenti vezérlővel
* Fizetve!:
  + Ha a rendelés státusza manuálisan “Fizetve”-re lett változtatva a lenti verézlővel. A rendelés a következő frissítésnél eltűnik a listából.

- A felület 5 másodpercenként frissül, ezzel lekérve a rendeléseket:

* Az új rendelések hozzáadódnak a listához
* A törölt / “Fizetve” státuszú rendelések törlésre kerülnek
* A már létező rendelések frissítésre kerülnek

- Minden frissítés után az asztalok választó frissül és egy Event lesz meghívva, ez beszínezi a foglalt/üres asztalokat.

- Betérő vendég választó: ha be van pipálva akkor nem szükséges kiválasztani egy foglalást, amint kiválasztottunk egy asztalt is, hozzáad egy foglalást a vendég felhasználóval, és felvesz rá egy rendelést.

- Rendelés Felvétele gomb: hozzáad egy új rendelést a kiválaszott foglaláshoz (kivéve ha betérő vendég be van pipálva.)

- Tétel hozzáadása gomb: megnyit egy új felületet amivel egy új tételt lehet hozzáadni a kiválaszott rendeléshez.

- Asztalra való kattintás után:

-Ha foglalt: megnyílik egy felület az asztalhoz tartozó tételekkel.

-Ha nem foglalt: információ ablak (az asztalnál nem ülnek).

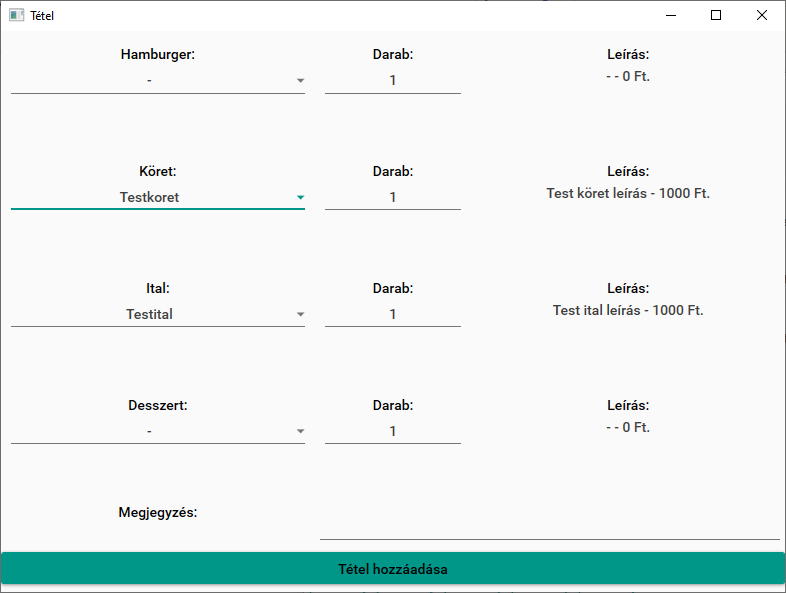
-Törlés gomb: Törli a listából és az adatbázisból a kiválasztott rendelést.

-Részletek gomb: megnyílik egy felület az asztalhoz tartozó tételekkel.

-Fizetésre vár gomb: megváltoztatja a kiválasztott rendelés státuszát “Fizetésre vár”-ra.

-Fizetve gomb: megváltoztatja a kiválasztott rendelés státuszát “Fizetve”-re.

Tétel felület:

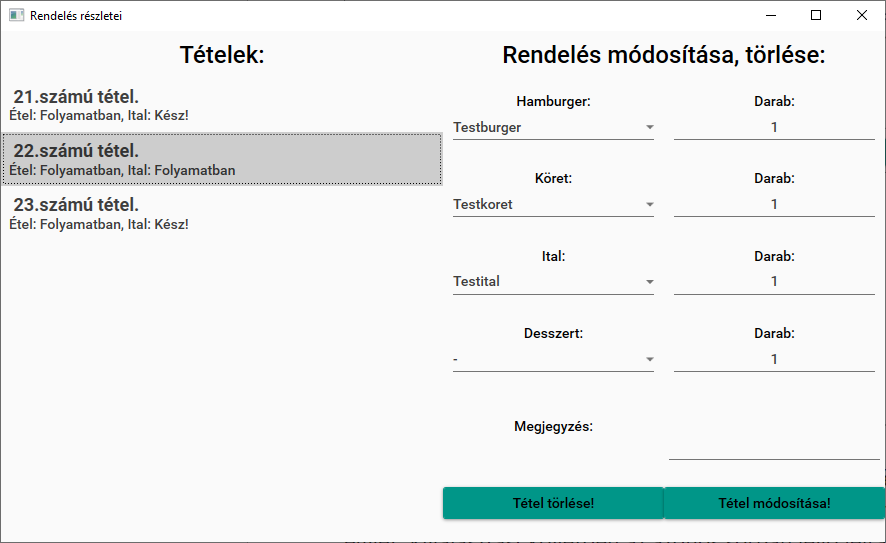


A felület részletei:

1. Hamburger, köret, ital, desszert választó.
2. Darab szövegdobozok
3. Leírás szövegdobozok
4. Megjegyzés szövegdoboz
5. Tétel hozzáadása gomb

A termékek választókba alapból az üres termék lesz kiválasztva, a gomb aktiválódik amint valamelyik termékből kiválasztunk egyet. Kiválasztást követően az azonos sorban lévő leírás doboz frissül. Lehetséges megjegyzés hozzáadása a tételhez. A tétel hozzáadása gomb nyomása után a tétel hozzáadásra kerül a kiválaszott rendeléshez.

Rendelés részletei:



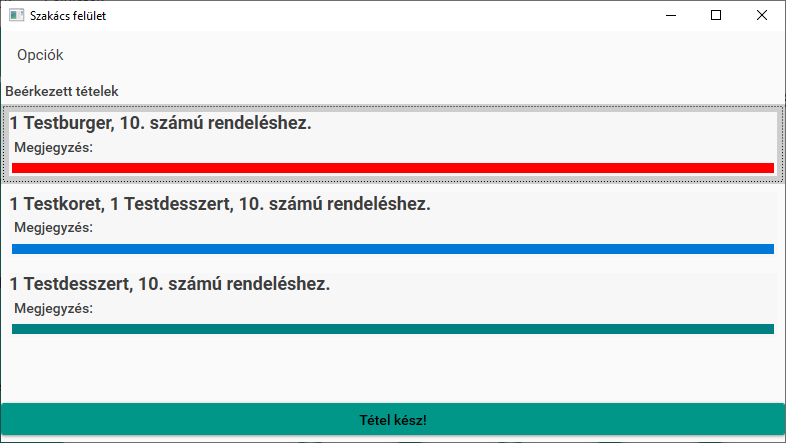
A felület részletei:

1. Tételek lista
2. Rendelés részletei
   1. Hamburger, köret, ital, desszert választó
   2. Darab szövegdobozok
   3. Megjegyzés szövegdoboz
   4. Tétel törlése gomb
   5. Tétel módosítása gomb

A tételek listában látjuk minden egyes tétel státuszát külön-külön.

Egy tétel kiválasztása után a rendelés részletei rész aktívvá válik, és beállításra kerülnek a tétel adatai. Lehetséges a tétel módosítása és a tétel törlése.

Szakács / pultos felület:



A felület részletei:

1. Beérkezett tételek lista
2. “Tétel kész” gomb

A tételek nevében benne vannak a megrendelt termékek, a darabszám és hogy melyik rendeléshez tartoznak (lehetségessé teszi hogy egy rendeléshez tartozó tételeket csináljunk).

Alatta a tételhez tartozó megjegyzés látható (ha van).

A tételek alján egy vízszintes csík található, ami:

* A tétel hozzáadása és < 30mp között Kékeszöld.
* 30 és 60mp között Kék.
* > 60mp Piros.

Figyelmeztetés céljára szolgál, felhívja az 1 percnél tovább bent levő tételekre a figyelmet.

!!! A pultos felület a szakács-al 100%-ba megegyező, azon kívül hogy oda a pultos tételek fognak csak bekerülni !!!

Admin felület:



A felület részletei:

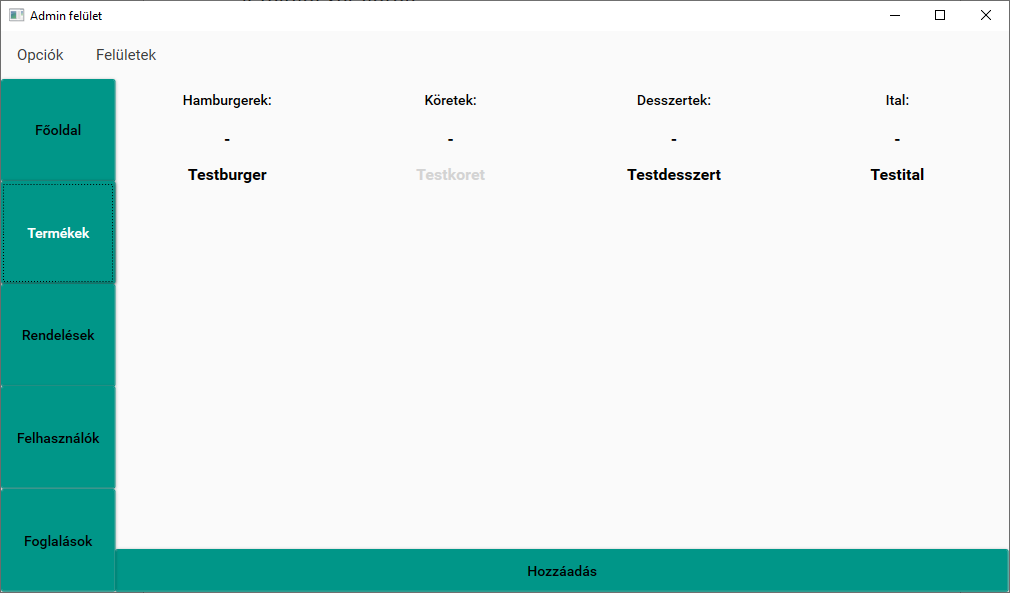
1. Navigációs menü
   1. Főoldal (áttekintés)
   2. Termékek kezelése
   3. Rendelések történet
   4. Felhasználók kezelése
   5. Foglalások történet
2. Az adott menü tartalma(ContentControl)

Főoldal(áttekintés):

* Mai, havi, összes bevétel
* Asztalok

A felület 5 másodpercenként frissül, ez lekéri a bevételeket és az aktív rendeléseket , amikből beszínezzük a foglalt asztalokat.

Termékek (Admin) felület:

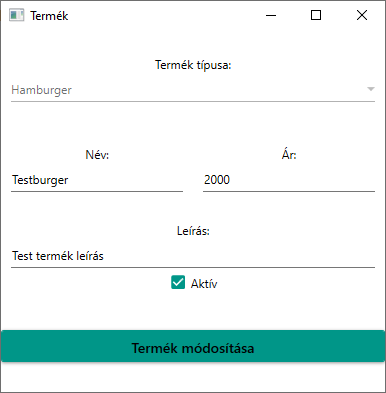


A felület részletei:

1. Termékek lista, kategorizálva
2. Termék hozzáadása gomb

A listában a nem aktív termékek világos fekete / félig átlátszó felirattal jelennek meg, ezek a termékek nem kerülnek árúsításra, a felszolgáló tétel hozzáadása ablakában sem jelennek meg.

Egy termék dupla bal kattintására a következő felület jelenik meg:



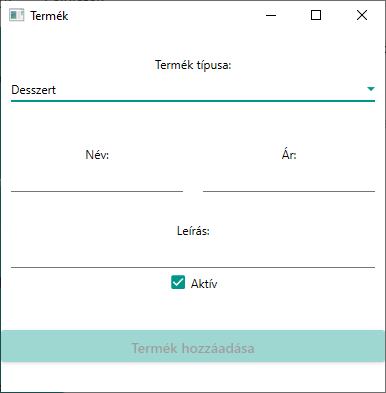
A felület részletei:

1. Termék típusa választó (inaktív)
2. Termék neve
3. Termék ára
4. Termék leírása
5. Termék aktív-e választó
6. Termék módosítása gomb

A felületen lehetséges a termék nevét, árát, leírását (és hogy aktív-e) módosítani.

A termék módosítása gombra kattintás után ha sikeres a módosítás, az ablak eltűnik, és a termék felületünk listája frissül.

Termék hozzáadása felület:



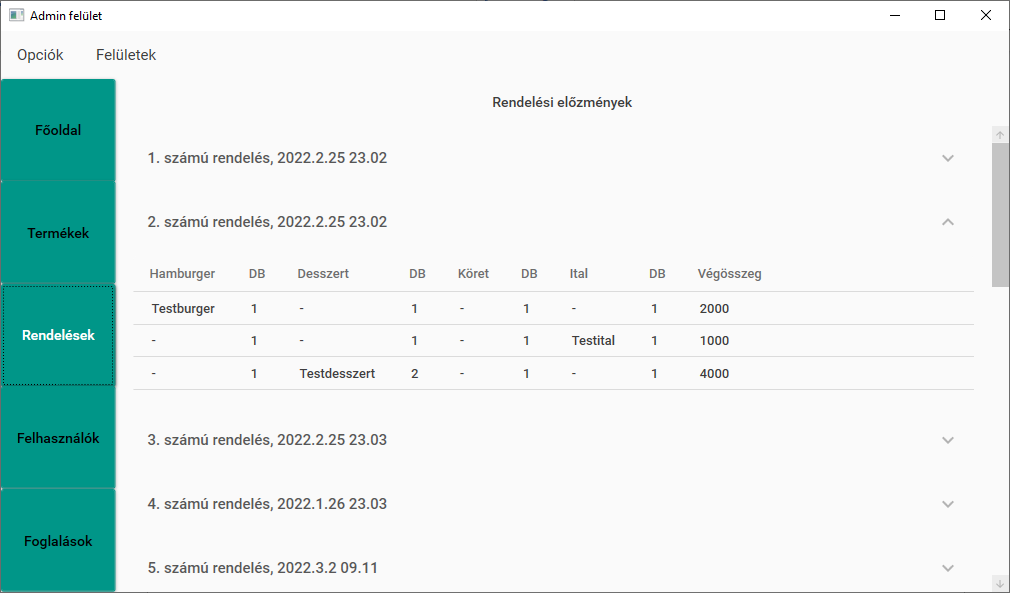
A felület részletei:

1. Termék típusa választó (aktív)
2. Termék neve
3. Termék ára
4. Termék leírása
5. Termék aktív-e választó
6. Termék hozzáadása gomb

Kiválasztható a termék típusa, minden mező megfelelő kitöltése után a Termék hozzáadása gomb kattinthatóvá válik.

A termék hozzáadása gombra kattintás után ha sikeres a hozzáadás, az ablak eltűnik, és a termék felületünk listája frissül.

Rendelések (Admin) felület:

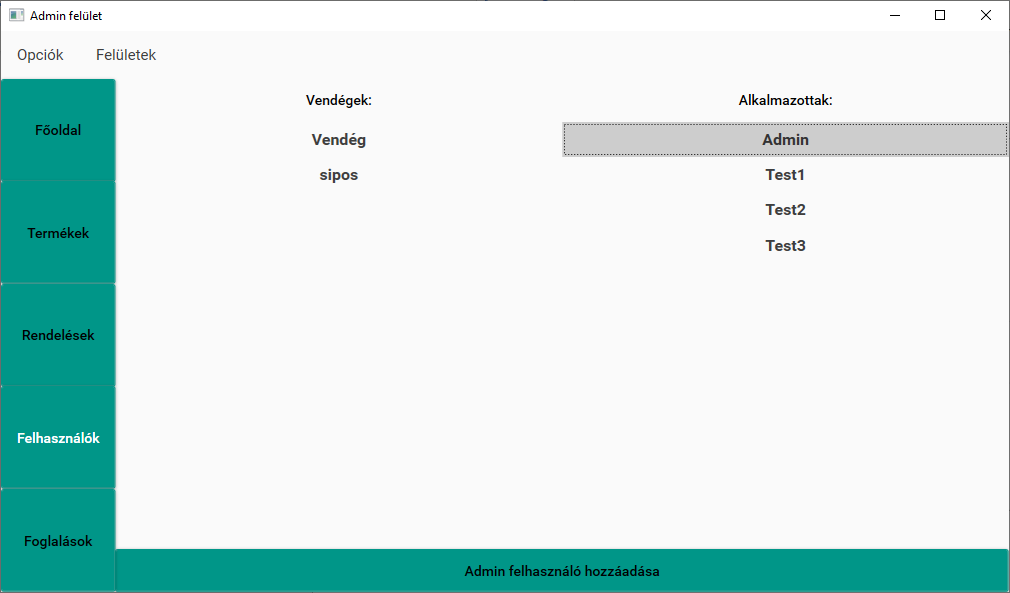
A felület részletei:

1. Rendelési előzmények lista

A listában minden eddigi, már fizetett rendelés található. A nevük a rendelés számuk és utána a dátum amikor leadásra került.

Egy rendelésre kattintás során lenyílik (és lekérdezi) az adott rendeléshez tartozó tételeket, ami táblázat formába jelenik meg (1 sor = 1 tétel).

Felhasználók (Admin) felület:

A felület részletei:

1. Vendégek lista
2. Alkalmazottak lista
3. Admin felhasználó hozzáadása gomb

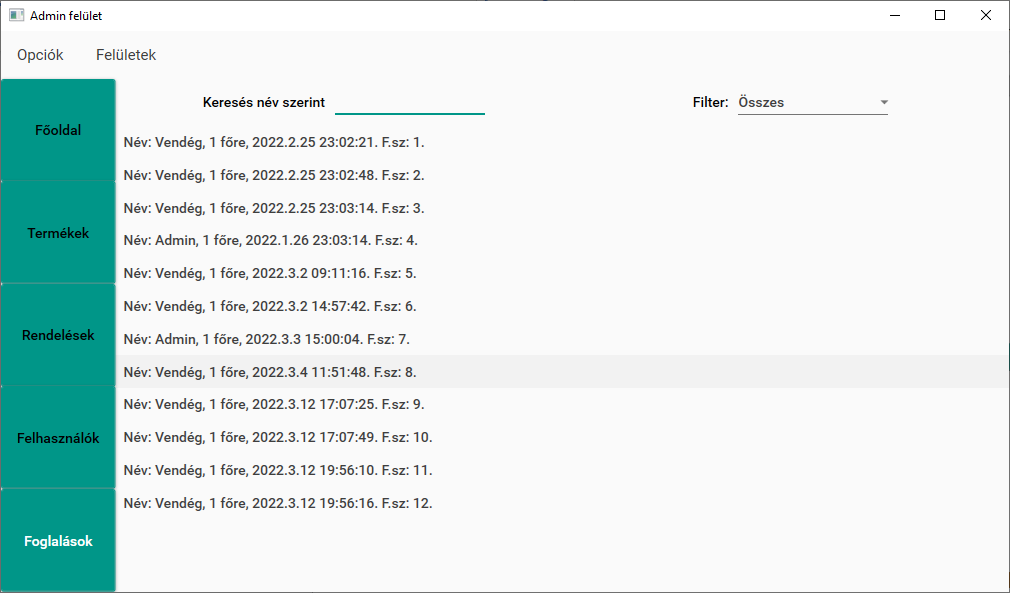
A listákban a regisztrált vendégek és alkalmazottak találhatóak.

Az alkalmazottak listában dupla bal kattintásra egy felhasználó módosítása felület jelenik meg.

Az admin felhasználó hozzáadása gomb kattintására egy regisztráció felület jelenik meg.

Mindkét funkció a fentebb dokumentált Regisztráció ablakot nyitja meg, annyi különbséggel hogy az alkalmazott módosításánál NINCS lehetőség a jelszó változtatására, admin regisztrálásnál a Jog választó inaktív, és “Admin” van kiválasztva benne.

Foglalások (Admin) felület:

A felület részletei:

1. Keresés név szerint mező
2. Filter választó (Összes, Mai nap, Ez a hónap)
3. Foglalások lista

A listában a már megjelent vendégek foglalásai találhatóak, a foglalás neve a vendég nevéből, a foglalás dátumából és a foglalás számból áll.

A keresőbe szöveg beírására a lista csak azokat a foglalásokat fogja tartalmazni ahol a vendég neve a beírt szöveggel kezdődik.

Ezen felül választható szűrés (akár kereséssel együtt), ahol erre a napra, hónapra vagy az összesre tudjuk leszűkíteni a foglalásokat.