

Ceglédi SZC Közgazdasági és Informatikai Technikum

Szakképzés megnevezése: Szoftverfejlesztő és -tesztelő
Szakma azonosító szám: 5 0613 12 03

VIZSGAREMEK **Burger Étterem**

Lendér Sándor
Témavezető

Készítette:
Sipos Ferenc
Szabó Dániel
Majercsik Zsolt

Cegléd 2022

Tartalomjegyzék:

- Adatbázis:
 - [Áttekintés](#)
 - [Adatbázis](#)
 - [Táblák szerkezete](#)
 - [Kapcsolatok](#)
 - [Eseményindítók](#)
 - [Megkötések](#)
- Weboldal:
 - [Áttekintés](#)
 - [Belépés](#)
 - [Foglalás](#)
 - [Regisztráció](#)
 - [Specifikáció](#)
 - [Navbar](#)
 - [Osztályok](#)
 - [Oldal szerkezete](#)
 - [Api meghívása](#)
 - [React](#)
- Asztali alkalmazás:
 - [Áttekintés](#)
 - [Fejlesztői dokumentáció:](#)
 - [API](#)
 - [WPF](#)
 - [Felhasználói dokumentáció:](#)
 - [Jelszó változtatási felület](#)
 - [Felszolgálói felület](#)
 - [Tétel hozzáadása felület](#)
 - [Rendelés felület](#)
 - [Szakács/pultos felület](#)
 - [Admin felület:](#)
 - [Termék](#)
 - [Rendelés](#)
 - [Felhasználó](#)
 - [Foglalás](#)

BEVEZETŐ

A projekt egy hamburgerező / étterem működését modellezi, amely megoldást nyújt egy étterem teljesen digitális, autómatikus működésére.

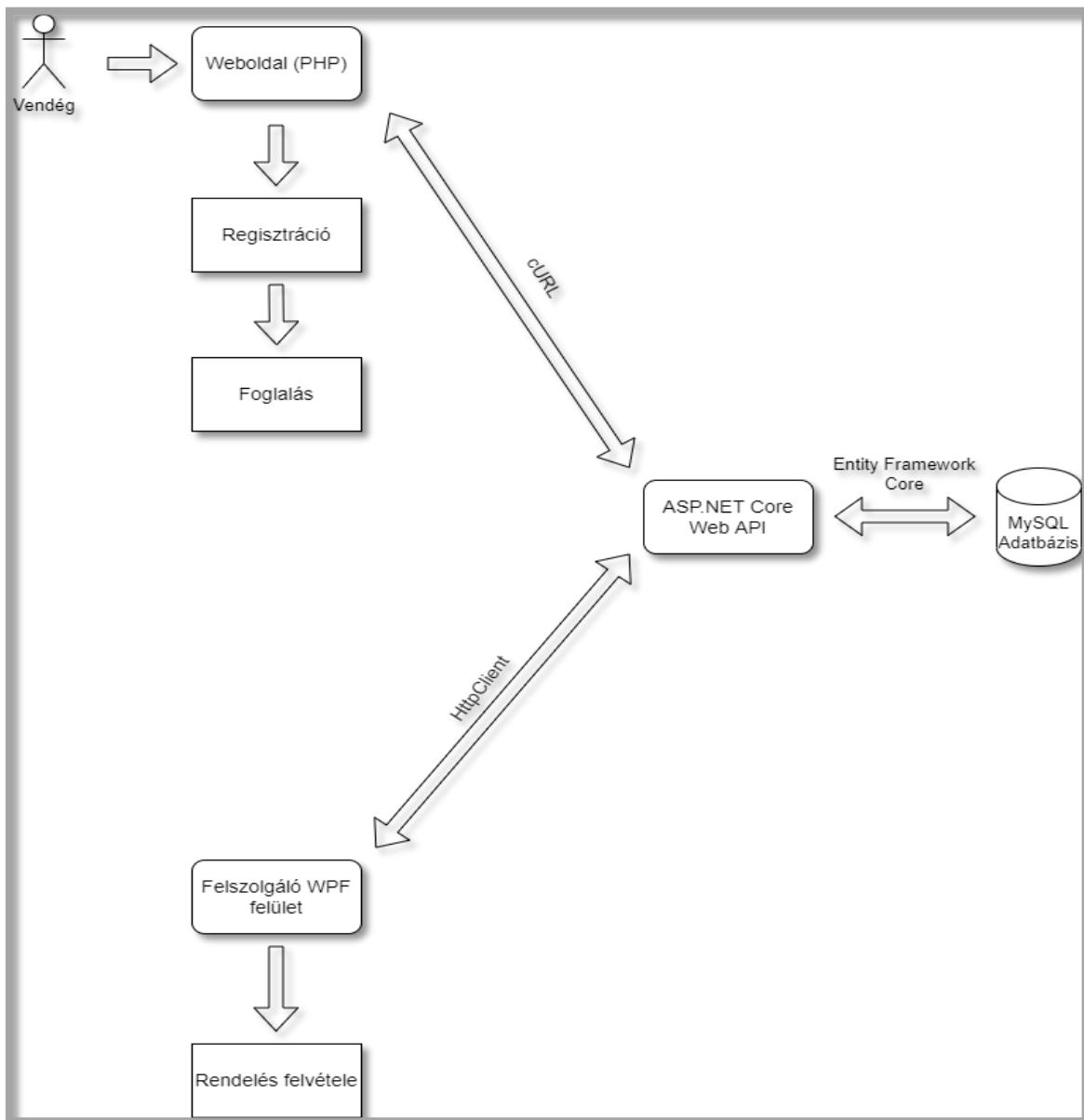
A weboldal lehetővé teszi a vendégek regisztrálását, az aktuális étlap elérését és interneten keresztül történő foglalást biztosít, ezzel kiváltva a telefonálást vagy a személyes megjelenést.

A adatbázis lehetővé teszi a termékek, rendelések, foglások és felhasználók tárolását.

A vastagklienses alkalmazás lehetővé teszi az alkalmazottak / tulajdonos(ok) regisztrálását, külön felületet biztosít a : felszolgálóknak, szakácsoknak, pultosoknak, tulajdonos(ok)-nak.

A weboldal és a vastagklienses alkalmazás egy API-n keresztül kommunikál az adatbázissal.

A projektünk elvi felépítése:



ADATBÁZIS

Az adatbázisokról általában:

Az adatok adatbázisokban való tárolására, kezelésére és lekérésére egy szabványos nyelvet, az **SQL-t** használjuk.

Az „SQL” egy mozaikszó, a Structured Query Language rövidítése, magyarul: struktúrált lekérdező nyelv.

Az SQL 1986-ban az Amerikai Nemzeti Szabványügyi Intézet (ANSI), 1987-ben pedig a Nemzetközi Szabványügyi Szervezet (ISO) szabványává vált.

Segítségével:

- lekérdezéseket tudunk végrehajtani egy adatbázisban;
- adatokat tudunk lekérni egy adatbázisból;
- rekordokat tudunk beszűrni az adatbázisba;
- frissíthetjük az adatbázis rekordjait;
- rekordokat tudunk törölni az adatbázisból;
- új adatbázisokat tudunk létrehozni;
- új táblákat tudunk létrehozni egy adatbázisban;
- tárolt eljárásokat tudunk létrehozni egy adatbázisban;

Bár az SQL ANSI/ISO szabvány, az SQL nyelvnek különböző verziói léteznek.

A szabványoknak való megfelelés érdekében azonban mindenki hasonló módon támogatja legalább a főbb parancsokat (például SELECT, UPDATE, DELETE, INSERT, WHERE).

A legtöbb SQL adatbázis-programnak az SQL szabványon kívül saját, szabadalmaztatott kiterjesztése is van!

A fontosabb SQL-parancsok közül néhány:

SELECT – adatokat nyer ki egy adatbázisból
UPDATE - frissíti az adatokat az adatbázisban
DELETE – törli az adatokat az adatbázisból
INSERT INTO - új adatokat szűr be egy adatbázisba
CREATE DATABASE – új adatbázist hoz létre
CREATE TABLE – új táblázatot hoz létre
ALTER TABLE - módosítja a táblázatot
DROP TABLE – táblázat törlése
CREATE INDEX – indexet hoz létre (kereső kulcs)
DROP INDEX – töröl egy indexet

RDBMS használata:

Az RDBMS a Relational Database Management System rövidítése (relációs adatbázis-kezelő rendszer), mely egy relációs adatbázis karbantartására szolgáló program.

Relációs adatbázisnak nevezzük a relációs adatmodell elvén létrehozott adatok összességét.

Reláció (azaz kapcsolati viszony) kétféleképpen jön létre egy adatbázisban.

Technikai relációról beszélünk akkor, ha a külön-külön tárolt adatokat bizonyos közvetítő adattal kapcsolunk össze. Például, ha személyek neveit és adószámait külön-külön táblában tároljuk és mondjuk egy azonosító számmal kötjük össze ezeket az adatokat.

A másik lehetőség, ha minden névvel együtt tároljuk a hozzájuk tartozó adószámot.

Ez elvi reláció, amelyben nincs közvetítő adat, a reláció csak névleges, technikailag a relációban álló két elem egyszerre érhető el.

Az elméleti relációs adatmodellben a reláció halmaz, ennek megfelelően a reláció minden eleme (sora) egyedi.

A tipikus relációsadatbázis-kezelők három módosítással élnek:

- a relációk jellemzően nem halmazok, hanem „zsákok”,
- nem teszik lehetővé, hogy egy relációban két azonos nevű oszlopa (attribútuma) legyen,
- lehetőség van az ún. NULL (üres, ismeretlen) értékek használatára.

A relációs adatbázisok kialakításának sajátosságaival az adatbázis-tervezés foglalkozik.

Az RDBMS minden modern adatbázis-rendszer alapja, ilyen rendszer például a MySQL, a Microsoft SQL Server, az Oracle és a Microsoft Access.

Az RDBMS SQL lekérdezéseket használ az adatbázisban lévő adatok eléréséhez.

Az RDBMS-ben lévő adatokat, tábláknak nevezett adatbázis-objektumok tárolják.

A tábla kapcsolódó adatbejegyzések gyűjteménye, amely oszlopokból és sorokból áll.

Minden tábla kisebb entitásokra, úgynevezett mezőkre van felosztva. A mező egy olyan oszlop a táblázatban, amelyet úgy terveztek, hogy a tábla minden rekordjáról specifikus információkat tároljon.

A rekord, -más néven sor- minden egyes bejegyzés, amely egy táblázatban létezik. A rekord egy tábla vízszintes entitása.

Az oszlop egy függőleges entitás a táblában, amely a tábla egy adott mezőjéhez kapcsolódó összes információt tartalmazza.

A tanulmányunk során mi a **MySQL** rendszert használtuk, melyet a phpMyAdmin eszközzel menedzseltünk (XAMPP Apache szerveren keresztül).

A MySQL egy széles körben használt relációs adatbázis-kezelő rendszer (RDBMS).

Ingyenes és nyílt forráskódú, ideális kis és nagy alkalmazásokhoz egyaránt.

Gyors, megbízható, méretezhető és könnyen használható. A rendszer többplatformos, kompatibilis az ANSI és SQL szabvánnal.

1995-ben jelent meg, az Oracle Corporation fejleszti, terjeszti és támogatja.

A MySQL nevét a társalapító Monty Widenius lányáról kapta: My.

MySQL-t használ a Facebook, Twitter, Airbnb, Booking.com, Uber, GitHub, YouTube stb.

Tartalomkezelő rendszerek is használják, mint a WordPress, Drupal, Joomla!, Contao stb.

Burger Étterem adatbázisa

A kitalált étteremünk működésének szimulálására kialakítottunk egy komplex adatbázist. A weboldal és az asztali alkalmazás is ezzel az adatbázissal dolgozik a különböző lekérdezések, adatfeltöltések révén.

Az adatbázis egyik része tartalmazza az egyszerű felhasználó, az adminisztrátor és a különböző munkahelyeken (pincér, konyha, pultos) dolgozó felhasználók adatait, jogosultságait.

A másik rész az étterem működtetéséhez szükséges adatokat tartalmazza.

Táblák szerkezete

Alkalmazott rövidítések:

PK: Primary Key – Elsődleges kulcs

FK: Foreign Key – Idegen kulcs

„Felhasználó” tábla

burgeretterem felhasznalo	
#	azon : int(6) unsigned
■	nev : varchar(50)
■	lak : varchar(100)
■	tel : varchar(12)
■	email : varchar(50)
#	jog : int(1)
■	pw : varchar(40)

- „azon”: ügyfél azonosító (szám, automatikusan generálódik, PK)
- „nev”: ügyfelnév (karakterek, ügyfél adja meg)
- „lak”: ügyfél lakhely (karakterek, ügyfél adja meg)
- „tel”: ügyfél telefonszám (karakterek, ügyfél adja meg)
- „email”: ügyfél email cím (karakterek, ügyfél adja meg)
- „jog”: ügyfél jogosultság (automatikusan „felhasználó”-ként generálódik, ha dolgozóról van szó, az admin módosítja a jogosultsági szintet)
- „pw”: jelszó (karakterek, ügyfél adja meg)

„Foglalás” tábla

burgeretterem foglalas	
#	fazon : int(4) unsigned
#	azon : int(6) unsigned
#	szemelydb : int(1)
■	foglalasido : datetime
■	leadva : datetime
#	megjelent : tinyint(1)

- „fazon”: foglalás azonosító (szám, automatikusan generálódik, PK)
- „azon”: ügyfél azonosító (szám, FK)
- „szemelydb”: a foglalás ennyi személyre szól (szám, max. 8, ügyfél adja meg)
- „foglalasido”: erre az időpontra foglalnak (datetime, ügyfél adja meg)
- „leadva”: a foglalás időpontja (datetime, automatikus)
- „megjelent”: a vedék megjelent-e a foglaláson (tinyint, foglaláskor ’0’ generálódik)
0: foglalás, de még nem jelent meg;
1: megjelent a foglaláson;

„Burger” tábla

burgeretterem burger	
#	bazon : int(1)
█	bnev : varchar(50)
#	bar : int(4)
█	bleir : varchar(150)
#	aktiv : tinyint(1)

- „bazon”: hamburger azonosítója (szám, PK)
- „bnev”: hamburger neve (karakterek)
- „bar”: hamburger ára (karakterek)
- „bleir”: hamburger leírása (karakterek)
- „aktiv”: a termék jelenleg elérhető-e a kínálatban (tinyint)
0: nem elérhető;
1: elérhető;

„Köret” tábla

burgeretterem koret	
#	kazon : int(1)
█	knev : varchar(50)
#	kar : int(4)
█	kleir : varchar(100)
#	aktiv : tinyint(1)

- „kazon”: köret azonosítója (szám, PK)
- „knev”: köret neve (karakterek)
- „kar”: köret ára (karakterek)
- „kleir”: köret leírása (karakterek)
- „aktiv”: a termék jelenleg elérhető-e a kínálatban (tinyint)
0: nem elérhető;
1: elérhető;

„Desszert” tábla

burgeretterem desszert	
#	dazon : int(1)
█	dnev : varchar(50)
#	dar : int(4)
█	dleir : varchar(100)
#	aktiv : tinyint(1)

- „dazon”: desszert azonosítója (szám, PK)
- „dnev”: desszert neve (karakterek)
- „dar”: desszert ára (karakterek)
- „kleir”: desszert leírása (karakterek)
- „aktiv”: a termék jelenleg elérhető-e a kínálatban (tinyint)
0: nem elérhető;
1: elérhető;

„Ital” tábla

burgeretterem ital	
#	iazon : int(1)
█	inev : varchar(50)
#	iar : int(4)
█	ileir : varchar(50)
#	aktiv : tinyint(1)

- „iazon”: ital azonosítója (szám, PK)
- „inev”: ital neve (karakterek)
- „iar”: ital ára (karakterek)
- „kleir”: ital leírása (karakterek)
- „aktiv”: a termék jelenleg elérhető-e a kínálatban (tinyint)
0: nem elérhető;
1: elérhető;

„Rendelés” tábla

burgeretterem rendeles	
#	razon : int(4) unsigned
#	fazon : int(4) unsigned
#	asztal : int(1)
#	ido : datetime
#	etelstatus : int(1)
#	italstatus : int(1)

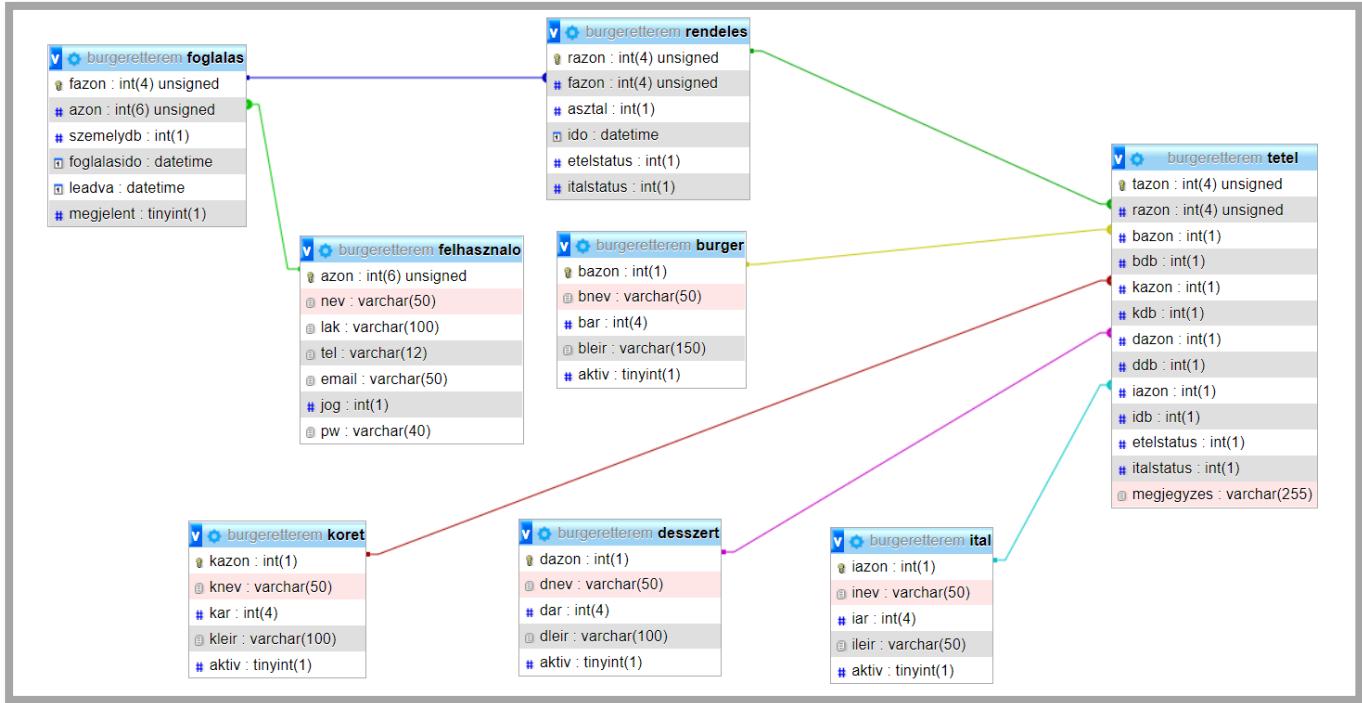
- „razon”: rendelés azonosító (szám, automatikusan generálódik, PK)
- „fazon”: foglalás azonosító (szám, FK)
- „asztal”: az asztal száma, ahonnan a rendelést adják (szám)
- „ido”: a rendelés ideje (datetime, automatikus)
- „etelstatus”: az étel rendelés állapota (szám)
- „italstatus”: az ital rendelés állapota (szám)

„Tétel” tábla

burgeretterem tetel	
#	tazon : int(4) unsigned
#	razon : int(4) unsigned
#	bazon : int(1)
#	bdb : int(1)
#	kazon : int(1)
#	kdb : int(1)
#	dazon : int(1)
#	ddb : int(1)
#	iazon : int(1)
#	idb : int(1)
#	etelstatus : int(1)
#	italstatus : int(1)
#	megjegyzes : varchar(255)

- „tazon”: tétel azonosító (szám, automatikusan generálódik, PK)
- „razon”: rendelés azonosító (szám, FK)
- „bazon”: hamburger azonosító (szám, FK)
- „bdb”: hamburger darabszám (szám)
- „kazon”: köret azonosító (szám, FK)
- „kdb”: köret darabszám (szám)
- „dazon”: desszert azonosító (szám, FK)
- „ddb”: desszert darabszám (szám)
- „iazon”: ital azonosító (szám, FK)
- „idb”: ital darabszám (szám)
- „etelstatusz”: a rendelt ételek státusza (szám)
 - 1: megrendelve, elkészítésre vár;
 - 2: konyhában elkészült, felszolgálásra vár;
 - 3: felszolgálva, fizetésre vár;
 - 4: fizetve;
- „italstatusz”: a rendelt italok státusza (szám)
 - 1: megrendelve, elkészítésre vár;
 - 2: pultban elkészült, felszolgálásra vár;
 - 3: felszolgálva, fizetésre vár;
 - 4: fizetve;
- „megjegyzes”: a rendeléssel kapcsolatos bármilyen megjegyzés, amit az elkészítéskor figyelembe kell venni: pl. „nem kér hagymát”, „sok jéggel kéri” stb. (karakterek)

A táblák kapcsolatainak átnézeti képe



Adatbázisunk tartalmaz eseményindítókat (triggerek) is.

Ezekkel azt érjük el, hogy ha egy táblában bizonyos adatmódosítást hajtunk végre, egy másik táblában automatikusan tudjuk módosítani a szükséges adatot.

Eseményindítóink:

Név	Tábla	Művelet	Idő	Esemény
<input type="checkbox"/> foglalasmegjelent	rendeles	Módosítás Exportálás Eldobás AFTER		UPDATE
<input type="checkbox"/> rendelestetelfriskit	tetel	Módosítás Exportálás Eldobás AFTER		UPDATE
<input type="checkbox"/> rendelestetelfriskitinsert	tetel	Módosítás Exportálás Eldobás AFTER		INSERT
<input type="checkbox"/> urestermekstatus	tetel	Módosítás Exportálás Eldobás BEFORE		INSERT

- „foglalasmegjelent”: amint rendelést vesznek fel egy adott foglalásra, a „foglalas” tábla „megjelent” mezőjében lévő 0-t 1-re állítja;

- „rendelestetelfriskit”: a „rendeles” tábla „etelstatusz” és „italstatusz” mezőit a „tetel” tábla „etelstatusz”, „italstatusz” mezőíhez igazítja, amint a státuszokat módosítják;

- „rendelestetelfrissitinsert”: ua. mint az előző, de az igazításokat akkor hajtja végre, ha új tételt szúrnak be az rendeléshez;
- „urestermekstatusz”: amennyiben nem rendeltek italt (iazon = 1), az ital státuszát egyből 2 (kész)-re állítja, ételnél ez akkor teljesül, ha se burgert, köretet, desszertet nem rendeltek.

Megkötések a táblákhoz:

- A „foglalas” tábla ’azon’ mezőjének referenciája a „felhasznalo” tábla ’azon’ mezője;
- A „rendeles” tábla ’fazon’ mezőjének referenciája a „foglalas” tábla ’fazon’ mezője;
- A „tetel” tábla ’bazon’ mezőjének referenciája a „burger” tábla ’bazon’ mezője;
- A „tetel” tábla ’kazon’ mezőjének referenciája a „koret” tábla ’kazon’ mezője;
- A „tetel” tábla ’dazon’ mezőjének referenciája a „desszert” tábla ’dazon’ mezője;
- A „tetel” tábla ’iazon’ mezőjének referenciája az „ital” tábla ’iazon’ mezője;
- A „tetel” tábla ’razon’ mezőjének referenciája a „rendeles” tábla ’razon’ mezője.

Az adatbázis leírása után következzenek az alkalmazások, melyek ebből az adatbázisból nyerik ki a működésükhez szüksége adatokat.

WEBOLDAL

A weboldal egy fiktív hamburgerező étteremet mutat be. Az oldalakon megtalálhatóak az éppen forgalmazott termékek, felhasználói regisztráció követően tudunk az étteremben asztalt foglalni, a foglalásainkat visszakövetni, esetleg lemondani.

Három fő részből áll:

- A főoldal: bemutatja magát az éttermet.
- Az ételekről és italokról szóló leírásokról, árakról.
- valamint a regisztrációból, asztalfoglalásból.

A fejlesztés alatt törekedtünk arra, hogy a weboldal áttekinthető, felhasználóbarát és egyértelmű legyen mindenki számára

Főoldal



A főoldalon található az étterem leírása, két képpel az étteremről. Illetve az összes oldal tetején szintén az étteremről láthatunk egy képnézegetőt.

Igyekeztünk egyszerű dizájnra törekedni, ezért sötét alapon világos betűszíneket alkalmaztunk.

Itt lehet választani a menüben a hamburgerek, köretek, desszertek, italok között.

Szintén a menüben foglal helyet a foglalás, ahol a regisztrációt és belépést követően, a felhasználónak lehetősége nyílik asztalt foglalni az étteremben.

Hamburgerek

The screenshot shows a dark-themed website for 'Burger Étterem'. At the top, there's a navigation bar with links to 'Főoldal', 'Hamburgerek' (which is the active tab), 'Körtekről', 'Desszertek', 'Italok', and 'Foglalás'. Below the navigation is a header with the text 'Burger Étterem'. The main content area displays two burger options: 'Sajtburger' and 'Pusztaburger'. Each item has a small image, a name, a detailed description in Hungarian, and a price in Ft.

- Sajtburger**
Összetevők: Buci, marhahúspogácsa, jégsaláta, édes hagyma, cheddar sajt, ketchup, mustár.
Ár: 950,-Ft.
- Pusztaburger**
Összetevők: Buci, marhahúspogácsa, csemegeuborka, rántott hagymakarika, gouda sajt, pusztaszósz.
Ár: 1150,-Ft.

A menüben a hamburgerek menürészt kiválasztva juthatunk el a „Hamburgerek” oldalra, ahol bővebb információkhhoz juthatunk az egyes hamburgerekről. Ezekről a hamburgerekről láthatunk képeket és leírást, hogy milyen alapanyagokból készülnek, valamint a hozzájuk tartozó árak is fel vannak tüntetve.

Körtekről

The screenshot shows a dark-themed website for 'Burger Étterem'. The navigation bar is identical to the previous page. The header reads 'Burger Étterem'. The main content area displays two side dish options: 'Csónakburgonya' and 'Édesburgonya'. Each item has a small image, a name, a detailed description in Hungarian, and a price in Ft.

- Csónakburgonya**
Összetevők: Olajban sült csónak formájú burgonyaszálletek.
Ár: 640,-Ft.
- Édesburgonya**
Összetevők: Olajban sült édesburgonya hasábok.
Ár: 800,-Ft.

A menüben kiválasztva a köretekről menüpontot eljuthatunk a „Körtekről” oldalra, ahol a köretekről kaphatunk bővebb leírást. Ezekről a köretekről láthatunk képeket, leírást, hogy milyen alapanyagokból készülnek, valamint a hozzájuk tartozó árak is fel vannak tüntetve.

Desszertek

Burger Étterem

Főoldal Hamburgerek Körtek Desszertek Italok Foglalás

Somlói galuska

Összetevők: Kakao-s vaníliás piskótá tézta, mazsolával, vaníliakrémmel, házi készítésű tejszínhabbal.

Ár: 800,-Ft.

Gesztenyepüré

Összetevők: Friss gesztenyemassza, házi készítésű tejszínhabbal.

Ár: 600,-Ft.

Szintén a menüpontok közül kiválasztva juthatunk el a „Desszertek” oldalra, ahol a desszertekről kaphatunk bővebb tájékoztatást. Ezekről a desszertekről láthatunk képeket és leírást, hogy milyen alapanyagokból készülnek, valamint a hozzájuk tartozó árak is fel vannak tüntetve.

Italok

Burger Étterem

Főoldal Hamburgerek Körtek Desszertek Italok Foglalás

Coca-Cola 0,33l

Összetevők: Cola ízű szénsavas üdítőital

Ár: 350,-Ft.

Coca-Cola Zero 0,33l

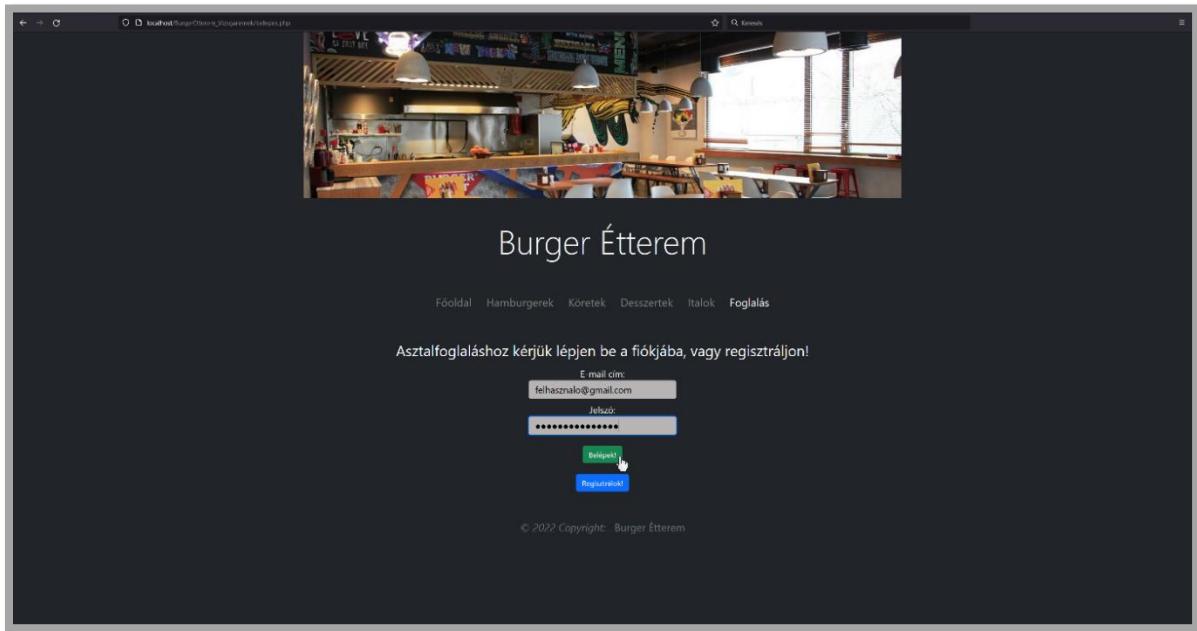
Összetevők: Energiamentes cola ízű szénsavas üdítőital

Ár: 350,-Ft.

Az italokról pedig az „Italok” menüpontban tájékozódhatunk. Ezekről az italokról láthatunk képeket és leírást, valamint a hozzájuk tartozó árak is fel vannak tüntetve.

Végül, a „Foglalás” menüpontra kattintva juthatunk el a felhasználók belépési oldalára, ahol a regisztrált email címmel és jelszóval beléphetünk a felhasználói fiókunkba.

Belépés



Burger Étterem

Főoldal Hamburgerek Körtek Desszertek Italok Foglalás

Asztalfoglaláshoz kérjük lépjön be a fiókjába, vagy regisztráljon!

E-mail cím:
felhasznalo@gmail.com

Jelszó:

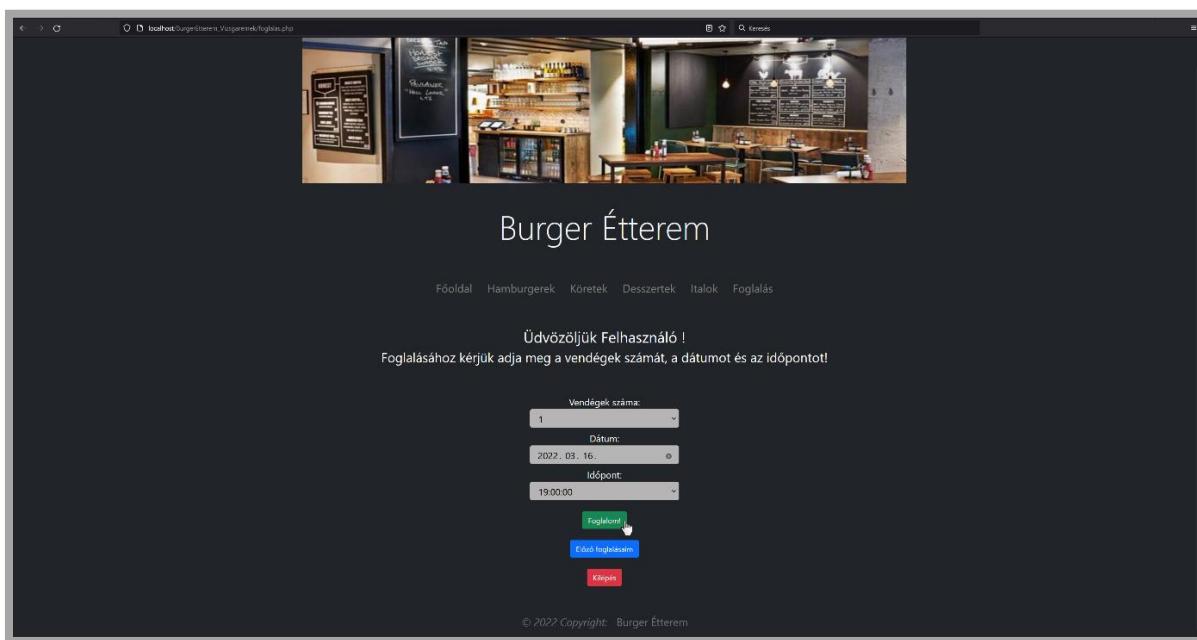
Belépés [Regisztráció](#)

© 2022 Copyright: Burger Étterem

Ha nem megfelelő formátumú email címet adunk meg, az email cím mezőnél hibaüzenetet jelenik meg. Valamint, ha nem regisztrált email címmel és/vagy hibás jelszóval akarunk belépni, akkor is hibaüzenetet kapunk.

A mezők kitöltése után, a „Belépés” gombra kattintva, a felhasználó „Foglalás” oldalára kerülünk.

Foglalás



Burger Étterem

Főoldal Hamburgerek Körtek Desszertek Italok Foglalás

Üdvözöljük Felhasználó!

Foglalásához kérjük adja meg a vendégek számát, a dátumot és az időpontot!

Vendégek száma:
1

Dátum:
2022. 03. 16.

Időpont:
19:00:00

Foglalás [Edző foglalásai](#) [Kiegy.](#)

© 2022 Copyright: Burger Étterem

Az üdvözlés után leírjuk a foglalás menetét: vendégek száma (max 8 lehet), a dátumot, és az órákra lebontott időpontok közül választva, az időpontot is (12:00 - 21:00)

Az „Előző foglalásaim” gombra kattintva, meg tudjuk nézni az előző foglalásainkat.

Előző foglalások

The screenshot shows a restaurant interior with tables and chairs. Below the image, the text "Burger Étterem" is displayed. A navigation bar at the top includes links for Főoldal, Hamburgerek, Körtek, Desszertek, Italok, and Foglalás. The main content area is titled "Kedves Felhasználó!" and contains the message "Éttermünkben az alábbi foglalásai vannak / voltak:". Below this is a table showing three previous bookings:

Foglalásazonosító	Vendégek száma	Foglalás időpontja	Foglalás lezárási ideje	Foglalás törlése
6	1	2022.03.15 19:00:00	2022.03.15 18:53:25	Töröl
7	6	2022.03.16 18:00:00	2022.03.16 07:13:06	Töröl
8	1	2022.03.16 18:00:00	2022.03.16 09:21:35	Töröl

At the bottom of the page, there is a green button labeled "Vissza a foglalásokhoz" and a copyright notice: "© 2022 Copyright: Burger Étterem".

Egyszerre több foglalásunk is lehet. Ha szeretnénk, le tudjuk mondani a foglalásainkat, a „Törlés” gombra kattintással. Azokat a foglalásainkat nem tudjuk törölni, amelyeken már megjelentünk az étteremben!

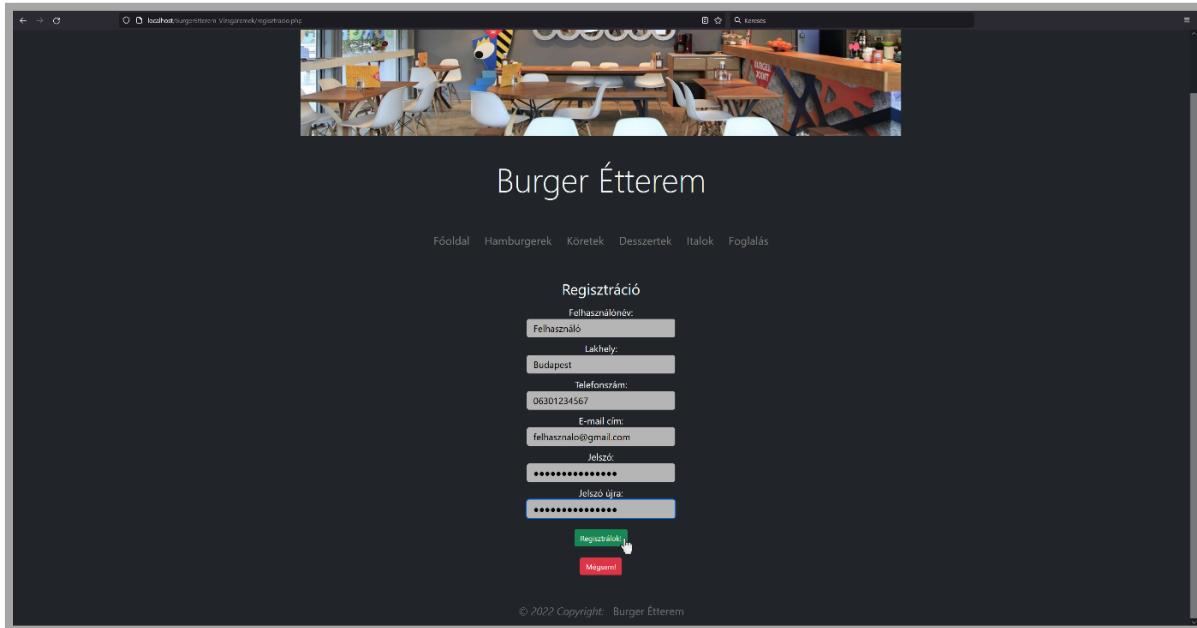
A „Vissza a foglalásokhoz” gombra kattintva térhetünk vissza a „Foglalás” oldalra, majd a „Kilépés” gombbal a főoldalra.

The screenshot shows the same restaurant interior as the previous page. Below the image, the text "Burger Étterem" is displayed. A navigation bar at the top includes links for Főoldal, Hamburgerek, Körtek, Desszertek, Italok, and Foglalás. The main content area contains the message "Asztalfoglaláshoz kérjük lépjön be a fiókjába, vagy regisztráljon!". Below this is a form with fields for "Email cím:" (with placeholder "az Ön email címe") and "Jelszó:" (with placeholder "az Ön jelszava"). There are two buttons: a green "Belépés!" button and a blue "Regisztráció!" button, which is highlighted with a cursor icon.

Ahhoz, hogy asztalt tudjunk foglalni az étterembe, regisztrált felhasználónak kell lennünk.

Ezt a „Foglalás” oldal „Regisztrálok” gombjára kattintva tehetjük meg.

Regisztráció



A „Regisztráció” oldalon, az adatainkat megadva regisztrálhatunk.

A telefonszám mezőnél csak számokat lehet megadni és esetleg a "+" jelet, mint előhívót.

Az email cím mezőnél meg kell adnunk a "@" jelet, hogy elfogadható legyen.

A jelszó mezőben legalább 6 karakterből kell állnia a jelszónak. Tartalmaznia kell nagybetűt, kisbetűt és számot is. A két jelszómezőbe írt karaktereknek természetesen meg kell egyezniük.

Funkcionális specifikáció

Használt technológiák:

- HTML 5
- CSS 3
- Bootstrap 4
- Javascript ES6
- React v.17.0.2
- PHP v.8.1.4
- MySQL v.8.0.28
- REST API (C#) + curl (PHP)

HTML: A HTML egy leírónyelv, amelyet weboldalak készítéséhez fejlesztettek ki, és az oldal vázát, felépítését lehet vele megalkotni. Úgynevezett ”tag”-ekkel lehet, különféle fajtájú és típusú elemeket megadni, amikből felépül az oldal.

CSS: A CSS vagyis lépcsőzetes stíluslapok egy stílusleíró nyelv, amely a HTML és XHTML típusú struktúrált dokumentumok megjelenítését írja le. Lényegében a HTML dokumentumok tartalmának formázásáért és a kinézetért felelős, szétválasztva a tartalmat a megjelenítéstől.

Bootstrap: A Bootstrap egy ingyenes és nyílt forráskódú CSS keretrendszer. Segítségével reszponzív és mobilbarát oldalakat hozhatunk létre. A keretrendszerben osztályok vannak definiálva melyeket hozzáadunk a megfelelő HTML elemekhez, így kialakítva a megjelenést.

Javascript: A Javascript egy objektumorientált kliensoldali szkript nyelv. Segítségével interaktívvá tudjuk tenni a weboldalainkat, jobb felhasználó élményt nyújtva ezzel. Sok weboldalon Javascripttel oldják meg a felhasználói interakciókat.

React: A React egy Javascript könyvtár, felhasználói felületek létrehozásához. Komponensekből épít fel a weboldal szerkezetét. Össze lehet kötni API-val, hogy azon keresztül kommunikáljon HTTPS kérésekkel.

PHP: A PHP egy általános szerveroldali szkript nyelv, amivel szintén dinamikus weboldalakat lehet készíteni. A kiszolgáló a PHP kódot nem küldi el a kliensnek, hanem a kiszolgáló oldalán a PHP értelmező dolgozza fel. Lehet vele adatbázis-lekérdezéseket is végezni. A PHP kódok kimenete a HTML elemekkel együtt kerül az ügyfélhez.

Szinte az összes oldal .php kiterjesztésű, mert a PHP szerveroldali script nyelvvel oldottuk meg az oldalak logikai felépítését. Illetve egyes helyeken Javascript is lett használva.

Mivel az oldalak hasonló felépítésűek, ezért több helyen is megtalálhatóak ugyanazon kódrészletek az egyes oldalakon.

index.php

```
1 <!DOCTYPE html>
2 <html lang="hu">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
9     integrity="sha384-1BmE4kWBg78iYhFldvuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
10  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
11    integrity="sha384-ka7Sk0Gln4gmtz2MIQnikT1wXgYsOg+OMhP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
12  <link rel="stylesheet" href="css/style.css">
13  <title>Burger Étterem</title>
14 </head>
15
16 <body class="container bg-dark">
17
18 <header>
19   <div class="row">
20     <div id="headerimage_carussel"></div>
21   </div>
22
23   <div class="jumbotron jumbotron-fluid bg-dark">
24     <div class="text-center text-light bg-dark p-5 display-2">Burger Étterem</div>
25   </div>
26 </header>
27
28 <?php
29   session_start();
30   include_once("navbar.php");
31 ?>
32
33 <div class="clearfix">
34   <p>A magas minőségű alapanyagokat magunk dolgozzuk fel, szigorú kontroll alatt, és ügy állítottuk össze, hogy a hozzávalók
35     tökéletes harmóniája fokozza a gasztronómiai élményt. <br>
36 </p>
```

1. Az összes *HTML5* felépítésű oldalnak tartalmaznia kell a `<!DOCTYPE html>` tag-et, a `<html>` nyitó és `</html>` záró tag-et, valamint egy `<head></head>` fejrészét és `<body></body>` törzsrészt.

A fejrészben tudjuk megadni az oldalra vonatkozó beállításokat. Mint például a karakterkódolást, az oldal címét és itt tudjuk belinkelni (hozzákapcsolni) az oldalhoz szükséges stíluslapokat, keretrendszeret és Javascriptben írt oldalakat.

2. Az oldal tartalmaz egy `<header>` részt. Ebben található két `<div>` tag, amikben szintén található egy-egy másik `<div>`.

Az első `<div>` tartalmazza azt a `<div>-et`, aminek az id tulajdonságának a `"headerimage_carussel"` lett megadva. Ezt az id-t használja a Reactban megírt `app.js`, aminek köszönhetően változnak folyamatosan, az oldal tetején található éttermi képek.

A második `<div>` tartalmazza azt a `<div>-et`, ami pedig az étterem címét jeleníti meg.

Mindegyik `<div>-re alkalmazva vannak különböző Bootstrap beállítások.`

3. PHP nyitó és záró tag között meghívjuk a `session_start()` függvényt. Erre azért van szükség, mert így az összes oldalon ahol meghívjuk ezt a függvényt, eltárolhatunk és hozzáférhetünk adatokhoz. Pl. a felhasználó nevéhez vagy az azonosítójához.

Az `include_once("navbar.php")`-vel tudjuk meghívni (beilleszteni) a `navbar.php`-ben található kódot, ami a menü megjelenítéséért és működéséért felelős.

```

index.php
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62

```

A magas minőségű alapanyagokat magunk dolgozzuk fel, szigorú kontroll alatt, és úgy állítottuk össze, hogy a hozzávalók tökéletes harmóniája fokozza a gasztronómiai élményt.
 minden burger más, kóstold meg őket, és garantált, hogy több kedvenced is lesz.

Amikor 2013-ban úgy döntöttünk, felpörgetjük, megreformáljuk az utcai étkezést, nem is gondoltuk, hogy ilyen gyorsan még ennél is tovább lépünk.
 A célnak már akkor is az volt, hogy minőségi, ízletes, prémium hamburgert készítsünk; olyat, ami a városban mozgolódva a legnagyobb rendezvényeken is elérhető.

Tudtuk, hogy külföldön van ilyen, és működik, sőt eszméletlenül menő.
 Elhatároztuk, hogy amit eddig itthon csak filmekben láttunk, igenis kell nekünk, és mi szeretnénk lenni a hazai "food truck mozgalom" zászlóvívói.
 Neki is láttunk, hogy Európa-szerte felkutassuk a legizletesebb összetevőket, a legolcsóbb zsemlélt és az igazi marhahúst. Ezeket összegyűrtük, és el is készítettek az első finom hamburgerek.

![éfterem-háttér1](images/hatter/Livepool-St-restaurant14-optimised.jpg)

A nyári fesztiválokon és az esti pörögésen túl fő irány, hogy nagyobb irodaházak, forgalmasabb városi csomópontok mellett állunk meg egy food truckkal, és kinálunk minőségi street food alternatívát a munkahelyi étkezésre.
 A kiállatot mindenkor a helyszínhez igazítjuk, így a burgerek mellett különböző snackeket kinálunk, és szomjaznotok sem kell nálnunk.

2014 novemberéhez köthető az újabb mérföldkő a Burger Étterem életében: megnyitottuk Budapest egyik legfinomabb gyorséttermét, a Seholnemtalálható utca 60. szám alatt, ahol a már megszokott minőségi és prémium hamburgereinket tálcan kínáljuk. Nektek egy olyan barátságos és hangulatos helyen, ahol télen kedvetekre üldögélhettek a barátaitokkal egy-egy

4. `<p>` tag-ek (paragrafus) között található az oldalon látható szöveg. Ahhoz, hogy egy mondat új sorban jelenjen meg, előtte sortörést kell létrehoznunk a `
` taggel. Ezt az előző mondat végén tettük itt meg.

5. A képeket szintén egy `<div>`-ben helyeztük el egy `` tagben, amire újfent Bootstrap beállításokat alkalmaztunk.

```

index.php
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104

```

Mert igy nem kell beutaznod Pestre, ha burgerezni szeretnél.
 Októberben, Budapest szívében, a Szabó utcai Konyha&Co. egységgel bővültünk, ahol a burgerek mellett inycsiklandozó ételkülönlegességekkel és hetről hétre változó ebed menüvel találkozhatsz, melyeket Szabó Dániel, Sipos Ferenc és Majercsik Zsolt séfek állítanak össze.

![éfterem-háttér2](images/hatter/LiverpoolSt_05.jpg)

2017 szeptemberétől az Alien Bevásárlóközpont bejáratánál is ott vagyunk egy food truckkal, vagyis Budán már két helyünk van.
 Továbbá "összekötöttük" a Királyka utcai 4-6-os megállót a II. Ferenc József térrrel; a Királyka 60 mellett a Királyka utca 20. szám alá is bekötözünk!

Polymatosan bővülünk, és járjuk az országot, néha pedig külföldre is kikacsintunk, hogy mindenki megismerhesse finom hamburgereinket.
 Te még nem próbáltad a Burger Éttermet? Akkor legfőbb ideje!

© 2022 Copyright:
Burger Étterem

6

7

6. Az oldal alján <footer> tagben jelenik meg középen az évszám és a copyright, valamint az étterem neve.

7. Ahhoz, hogy a 2. pontban említett Reactos képváltás működhessen, itt adjuk meg a szükséges linkeket hozzá <script> tagek között.

navbar.php

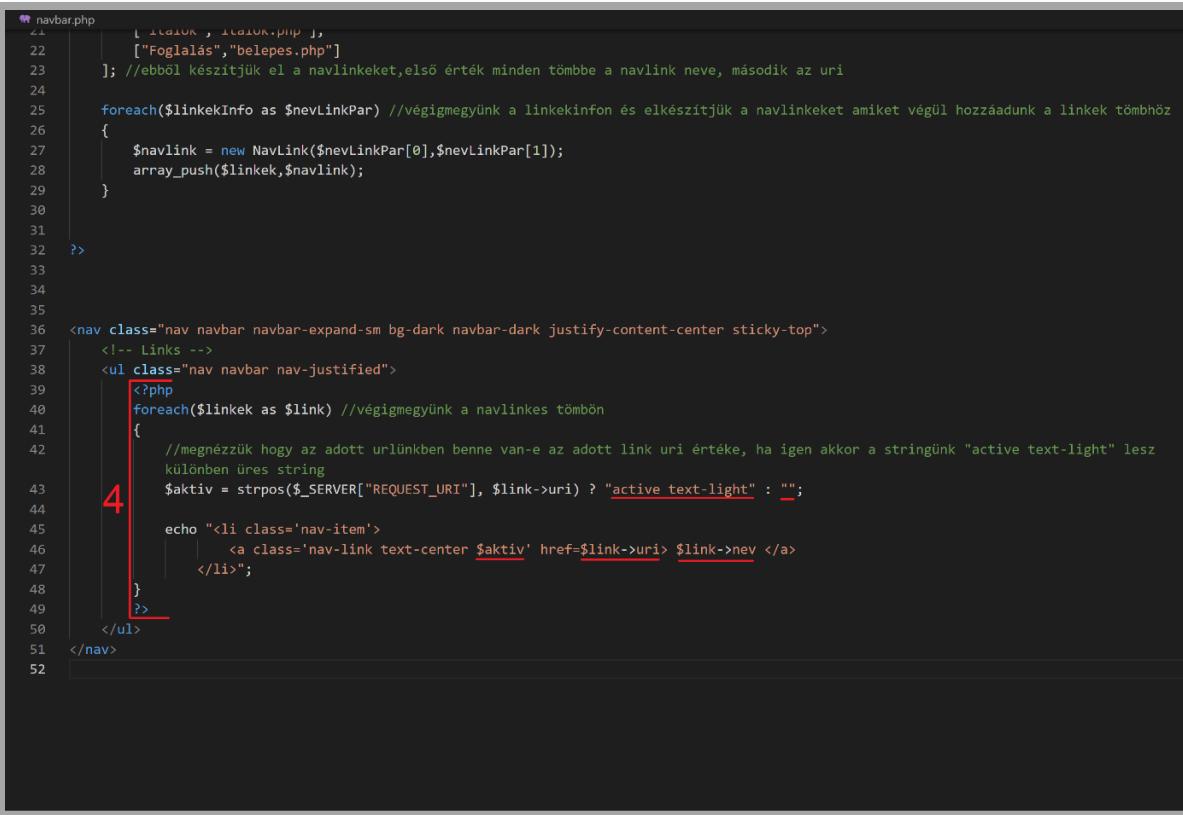
```
❶ navbar.php
1  <?php
2  class NavLink
3  {
4      public $nev;
5      public $uri;
6
7      function __construct($nev,$uri)
8      {
9          $this->nev = $nev;
10         $this->uri = $uri;
11     }
12 }
13
14 $linkek = []; //ebbé lesznek a navlink classú ítemek
15 $linkekInfo =
16 [
17     ["Főoldal","index.php"],
18     ["Hamburgerek","hamburgerek.php"],
19     ["Köretek","koretek.php"],
20     ["Desszertek","desszertek.php"],
21     ["Italok","italok.php"],
22     ["Foglalás","belepes.php"]
23 ]; //ebből készítjük el a navlinkekét,első érték minden tömbbe a navlink neve, második az uri
24
25 foreach($linkekInfo as $nevLinkPar) //végigmegyünk a linkekinfon és elkészítjük a navlinkeket amiket végül hozzáadunk a linkek tömbhöz
26 {
27     $navlink = new NavLink($nevLinkPar[0],$nevLinkPar[1]);
28     array_push($linkek,$navlink);
29 }
30
31
32 ?>
33
34
35
36 <nav class="nav navbar navbar-expand-sm bg-dark navbar-dark justify-content-center sticky-top">
37     <!-- Links -->
38     <ul class="nav navbar nav-justified">
39         <?php
```

1. Létrehozunk egy *NavLink* nevű osztályt két adattaggal és egy konstruktornnal. A konstruktornban beállítjuk, hogy példányosításkor a neki átadott névvel és url-el hozzon létre egy menüelemet (linket).

2. Létrehozunk egy *\$linkek* nevű tömböt. Ez fogja tárolni a menü elemeket.

Majd létrehozunk egy *\$linkekInfo* nevű tömböt is, ami annyi tömböt fog tárolni, ahány menü elem van. A menüelem nevével és a fájlnévvel, ami a link létrehozásához szükséges.

3. Foreach-el bejárjuk a *\$linkekInfo* tömböt *\$nevLinkPar* néven. Készítünk egy *\$navlink* változót, amiben példányosítunk egy *NavLink* osztályt, aminek az első eleme a *\$linkekInfo*-ban tárolt menüelemek első eleme, (vagyis a neve) míg a második eleme a fájlnév lesz.



```

21     [ 'index', 'index.php' ],
22     ["Foglalás","belepes.php"]
23   ]; //ebből készítjük el a navlinkeket,első érték minden tömbbe a navlink neve, második az uri
24
25   foreach($linkekInfo as $nevLinkPar) //végigmegyünk a linkekinfo és elkészítjük a navlinkeket amiket végül hozzáadunk a linkek tömbhöz
26   {
27     $navlink = new NavLink($nevLinkPar[0],$nevLinkPar[1]);
28     array_push($linkek,$navlink);
29   }
30
31
32 ?>
33
34
35
36 <nav class="nav navbar navbar-expand-sm bg-dark navbar-dark justify-content-center sticky-top">
37   <!-- Links -->
38   <ul class="nav navbar nav-justified">
39     <?php
40       foreach($linkek as $link) //végigmegyünk a navlinkes tömbön
41     {
42       //megnézzük hogy az adott urlünkben benne van-e az adott link uri értéke, ha igen akkor a stringünk "active text-light" lesz
43       //kulönben üres string
44       $aktiv = strpos($_SERVER["REQUEST_URI"], $link->uri) ? "active text-light" : "";
45
46       echo "<li class='nav-item'>
47         <a class='nav-link text-center $aktiv' href=$link->uri> $link->nev </a>
48       </li>";
49     }
50   </ul>
51 </nav>
52

```

4. Megjelenítjük a menü elemeit. Ehhez létrehozunk egy `<nav>` tag-ben egy számozatlan listát (`` tag) és ezen belül PHP-val oldjuk meg a listaelemek felépítését.

PHP-n belül `foreach`-el végig megyünk a `$linkek` tömbön. Az `$aktiv` változóban eltároljuk egy ternáris kifejezés kiértékelését. Miszerint, ha a `strpos()` függvény igaz értéket ad, akkor az `"active text-light"` értéket tárolja el. Ha hamis értéket ad, akkor pedig egy üres string-et.

A `strpos()` függvényen belüli első paraméter a teljes elérési utat adja vissza, a második pedig hogy megtalálhatóak-e ebben az elérési útban, a menü elemek fájlnevei (pl. `index.php`).

Majd `echo`-val megjelenítjük a listaelemeket (``) bennük a linkekkel (`<a>`). A linkek osztály tulajdonságainál adjuk meg ezt az `$aktiv` változót, ami által, ha a jelenleg betöltött oldal url-jében megtalálható magának az oldalnak a fájlneve, akkor az a menüelem lesz aktív (`index.php` esetén a Főoldal). Míg a többi oldallal, amelyekkel nincs egyezés, azok inaktívak maradnak.

Ugyanígy a `href` tulajdonságnál is behelyettesítődik a `$link->uri` helyére az oldalt tartalmazó fájl neve (`index.php`), valamint a link tag-jei között a menüelem neve (Főoldal).

Osztályok

Osztályok alkalmazásával tudjuk átláthatóbbá tenni az oldalak egyes részeinek a működését.

A Termék osztály példányosításával létrehozhatjuk a hamburgereket, desszerteket, italokat stb.

Míg a Felhasználó osztállyal az egyes személyeket hozhatjuk létre.

Ezekkel a példányokkal tudunk dolgozni a későbbiek folyamán.

Összesen öt darab osztályunk van:

Termék

```
Termek.php
1  <?php
2  class Termek
3  {
4      public $id,$nev,$ar,$leiras,$aktiv;
5
6      function __construct($id,$nev,$ar,$leiras,$aktiv)
7      {
8          $this->id = $id;
9          $this->nev = $nev;
10         $this->ar = $ar;
11         $this->leiras = $leiras;
12         $this->aktiv = $aktiv;
13     }
14 }
15 ?>
```

Felhasználó

```
Felhasznalo.php
1  <?php
2  class Felhasznalo
3  {
4      public $Email, $Pw;
5  }
6 ?>
```

Felhasználó regisztráció

```
FTP FelhasznReg.php
1  <?php
2  |   class FelhasznReg
3  |   {
4  |       public $Azon, $Nev, $Lak, $Tel, $Email, $Jog, $Pw;
5  |   }
6  ?>
```

Felhasználó ellenőrzés

```
FTP FelhasznEll.php
1  <?php
2  |   class FelhasznEll
3  |   {
4  |       public $Nev, $Email;
5  |   }
6  ?>
```

Foglalás

```
FTP Foglal.php
1  <?php
2  |   class Foglal
3  |   {
4  |       public $Fazon, $Azon, $Szemelydb, $Foglalasido, $Leadva, $Megjelent;
5  |   }
6  ?>
```

Az egyes oldalak szerkezete:

hamburgerek.php

```
hamburgerek.php
1 <?php
2 session_start();
3 include_once "Termek.php";
4 include_once "callApi_2.php";
5
6 $result = CallAPI("GET", "https://localhost:5001/Burgerek/Aktiv"); // CallAPI hívása, csak az aktív elemeket adja vissza
7
8 $decoded = json_decode($result[1]); // a visszakapott választ json formátummá alakítjuk
9 //var_dump($decoded);
10 $hamburgerek = []; //ebben lesznek a Termek classú ítemek
11 //for ($sor = 1; $sor < count($decoded); $sor++) {
12 //    $item = $decoded[$sor];
13 //    $hamburgerek[] = $item;
14 //} //ciklus a dekódolt eredmény végéig
15
16 $hamburgerek = new Termek($hamburgerek);
17 foreach ($hamburgerek as $hamburger) {
18     array_push($hamburgerek, $hamburger);
19 }
20
21 ?>
22
23 <!DOCTYPE html>
24 <html lang="hu">
25
26     <head>
27         <meta charset="UTF-8">
28         <meta http-equiv="X-UA-Compatible" content="IE=edge">
29         <meta name="viewport" content="width=device-width, initial-scale=1.0">
30         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
31             integrity="sha384-1BmE4kvBq78iYhFldvuhfTAU6au8tT94WrlHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
32         <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
33             integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
34         <link rel="stylesheet" href="css/style.css">
35         <title>Burger Étterem</title>
36     </head>
37     <body class="container bg-dark">
```

1. Mindig a `session_start()` függvény meghívásával kell kezdenünk a PHP kódot, hogy minden oldalon tudjuk használni, a SESSION-ben tárolt változókat. (lásd: `index.php` 3.)

Itt include-oljuk be a `Termek` osztályt és a `CallApi`-t is.

2. A `$result` változóba elmentjük a `CallAPI` függvény által kapott hamburgerek aktív elemeit.
3. A vissza kapott `json` objektumú választ PHP objektummá alakítjuk a `json_decode()` függénnyel, és elmentjük a `$decoded` változóba.

Létrehozunk egy üres `$hamburgerek` tömböt és egy `$sor` változót, aminek az értékét nullára állítjuk.

While ciklussal addig megyünk, amíg a `$sor` értéke kisebb a dekódolt objektumok számánál.

A `$burger` változóba lementjük az objektum egy sorát. A `$hamburger` változóban példányosítunk egy `Termek` osztályt, és az egyes elemeihez hozzárendeljük az objektumban kapott elemeket. (név = a hamburger neve, ára = a hamburger ára stb.)

Majd az `array_push()` függénnyel beletesszük a `$hamburgerek` tömbbe és növeljük a `$sor` értékét egyel.

Ezzel feltöljtük a `$hamburgerek` tömböt az egyes hamburgerek adataival.

```

49     <?php
50     |     include_once("navbar.php");
51     ?>
52     <br />
53
54     <div class="tab-content">
55         <?php
56             foreach($hamburgerek as $h) //végigmegyünk a termékeket tartalmazó tömbön és megjelenítjük
57             {
58                 echo
59                 "<div class='container tab-panel'><br>
60                     <div class='row p-2 m-3'>
61                         <div class='col-sm-6 col-m-12'>
62                             <div class='text-center'>
63                                 <img src='images/burgerek/$h->id.jpg' style='width:25rem; height: 18rem;' class='img-fluid rounded'
64                                     alt=$h->nev>
65                             </div>
66
67                             <div class='col-sm-6'>
68                                 <h1 class='text-center text-light p-2 m-3'>
69                                     $h->nev
70                                 </h1>
71                                 <p>
72                                     <span class='text-info'>Összetevők: </span>
73                                     <span class='p-2'>$h->leiras</span>
74                                 </p>
75                                 <p>
76                                     <span class='text-warning'>Ár: </span>
77                                     <span class='p-2'>$h->ar,-Ft.</span>
78                                 </p>
79                             </div>
80                         </div>
81                     </div>";
82             }
83         ?>
84     </div>
85
86     <footer class="page-footer text-center wow fadeIn">
```

4

4. Foreach-el végigmegyünk a hamburgerek példányait tartalmazó tömbön és echo-val kiíratjuk a megjelenítéshez szükséges HTML tartalmat.

A kép forrásához, a hamburger nevéhez, leírásához és árához behelyettesítjük a példányok megfelelő értékeit, megjelenítve így az összes hamburger képét, nevét és leírását.

Ugyanilyen módon épülnek fel a köretek, a desszertek és az italok oldalak is.

regisztracio.php

```
98
99     <br />
100
101    <div class="container">
102        <div class="ujfelhasznalo">
103
104            <form name="regist" method = "POST" action="" onsubmit="return formEllenorzes()">
105                <div>
106                    <h2>Regisztráció<h2>
107                </div>
108                <div><!--Alapból a placeholder felirata jelenik meg az input mezőben, ha már volt beleírva valami, akkor a javításkor
már az jelenik meg! --&gt;
109                    &lt;label class="labela"&gt;Felhasználónév:&lt;/label&gt;&lt;br /&gt;
110                    &lt;input type="text" class="bevitel" name="nev" id="nev" value=&lt;?php if(isset ($_SESSION['Felhasznalonev'])){ echo
111                         ".$_SESSION['Felhasznalonev'];}?&gt;&gt; placeholder="felhasználónév"
112                &lt;/div&gt;
113                &lt;div&gt;
114                    &lt;label class="labela"&gt;Lakhely:&lt;/label&gt;&lt;br /&gt;
115                    &lt;input type="text" class="bevitel" name="lak" id="lak" value=&lt;?php if(isset ($_SESSION['Lakcim'])){ echo "".
116                         $_SESSION['Lakcim'];}?&gt;&gt; placeholder="lakcím"
117                &lt;/div&gt;
118                &lt;div&gt;
119                    &lt;label class="labela"&gt;Telefonszám:&lt;/label&gt;&lt;br /&gt;
120                    &lt;input type="text" class="bevitel" name="tel" id="tel" value=&lt;?php if(isset ($_SESSION['Telefon'])){ echo "".
121                         $_SESSION['Telefon'];}?&gt;&gt; placeholder="pl.: +3680111111"
122                &lt;/div&gt;
123                &lt;div&gt;
124                    &lt;label class="labela"&gt;E-mail cím:&lt;/label&gt;&lt;br /&gt;
125                    &lt;input type="email" class="bevitel" name="email" id="email" value=&lt;?php if(isset ($_SESSION['Email'])){ echo "".
126                         $_SESSION['Email'];}?&gt;&gt; placeholder="pl.: valami@valami.com"
127                &lt;/div&gt;
128                &lt;div&gt;
129                    &lt;label class="labela"&gt;Jelszó:&lt;/label&gt;&lt;br /&gt;
130                    &lt;input type="password" class="bevitel" name="pw1" id="pw1" value=&lt;?php if(isset ($_SESSION['pw1'])){ echo "".
131                         $_SESSION['pw1'];}?&gt;&gt; placeholder="min. 6 karakter; kisbetű, nagybetű, szám"
132                &lt;/div&gt;
133                &lt;div&gt;
134                    &lt;label class="labela"&gt;Jelszó újra:&lt;/label&gt;&lt;br /&gt;
135                    &lt;input type="password" class="bevitel" name="pw2" id="pw2" value=&lt;?php if(isset ($_SESSION['pw2'])){ echo "".</pre>
```

1.2

```
121     <label class="labela">E-mail cím:</label><br />
122     <input type="email" class="bevitel" name="email" id="email" value="=php if(isset($_SESSION['Email'])){ echo "".
123         $_SESSION['Email'];}?" placeholder="pl.: valami@valami.com">
124   </div>
125   <div>
126     <label class="labela">Jelszó:</label><br />
127     <input type="password" class="bevitel" name="pw1" id="pw1" value="=php if(isset($_SESSION['pw1'])){ echo "".
128         $_SESSION['pw1'];}?" placeholder="min. 6 karakter; kisbetű, nagybetű, szám">
129   </div>
130   <div>
131     <label class="labela">Jelszó újra:</label><br />
132     <input type="password" class="bevitel" name="pw2" id="pw2" value="=php if(isset($_SESSION['pw2'])){ echo "".
133         $_SESSION['pw2'];}?" placeholder="jelszó megerősítése">
134   </div>
135   <br />
136   <input type="submit" class="btn btn-success" value="Regisztrálok!">
137   <br /><br />
138   <a class="btn btn-danger" href="index.php">Mégsem!</a>
139   </form>
140
141   <br />
142
143   <br />
144
145   <footer class="page-footer text-center wow fadeIn">
146     <div class="py-3 bg-dark">
147       <span class="footer-copyright text-secondary center" id="copyright">© 2022 Copyright:</span>
148       <span class="text-secondary center">Burger Étterem</span>
149     </div>
150   </footer>
151
152   <script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>
153   <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
154   <script src="app.js"></script>
155
156 </body>
</html>
```

1. A kialakított form *onsubmit* tulajdonsága visszatér a *formEllenorzes()* függvénytellyel, ami leellenőrzi az összes beviteli mezőbe beírt érték bizonyos szempontok szerinti helyességét.

Ha a SESSION változókban van tárolt érték, azokat a hozzájuk rendelt input mezők value paramétere megkapja. Ez akkor történik meg, ha a *formEllenorzes()* függvény *false* értékkel tér vissza

```
2.1
157 // regisztracio.php
158 <script>
159     // A regisztrációs adatok előzetes ellenőrzése
160     function formEllenorzes() {
161         // Névmező kitöltésének ellenőrzése
162         let a = document.forms["regist"]["nev"].value;
163         if (a == "" || a == null) {
164             alert("Név megadása kötelező!");
165             return false;
166         }
167         // Lakcím ellenőrzése
168         let b = document.forms["regist"]["lak"].value;
169         if (b == "" || b == null) {
170             alert("Lakcím megadása kötelező!");
171             return false;
172         }
173         // Telefonszám ellenőrzés
174         let c = document.forms["regist"]["tel"].value;
175         if (c == "" || c == null) {
176             alert("Telefonszám megadása kötelező!");
177             return false;
178         }
179         // Telefonszám hossz ellenőrzés
180         if (c.length < 10) {
181             alert("Telefonszám túl rövid! Kérlek használjon elöhhívó tagot és körzetszámot is!");
182             return false;
183         }
184         // Telefonszám karakterellenőrzés
185         let tel1 = c.match(/[A-z]/g);
186         let tel2 = c.match(/[^"!@#$%^&*]/g);
187         if (tel1 != null || tel2 != null) {
188             alert("Telefonszám csak +' előtagot és számokat tartalmazhat!");
189             return false;
190         }
191         // Email cím ellenőrzés
192         let d = document.forms["regist"]["email"].value;
193         if (d == "" || d == null) {
194             alert("Email cím megadása kötelező!");
195             return false;
196     }
```

```
2.2
195         return false;
196     }
197     // Jelszómezők ellenőrzése
198     // Első mező
199     let e = document.forms["regist"]["pw1"].value;
200     if (e == "" || e == null) {
201         alert("Az első jelszómező nincs kitöltve");
202         return false;
203     }
204     //Második mező
205     let f = document.forms["regist"]["pw2"].value;
206     if(f == "" || f == null) {
207         alert("A második jelszómező nincs kitöltve");
208         return false;
209     }
210     //Jelszavak összehasonlítás
211     if (e != f) {
212         alert("A két jelszómező nem egyezik meg!");
213         return false;
214     }
215     // Jelszó hosszának ellenőrzése
216     if (e.length < 6) {
217         alert("A jelszónak minimum 6 karakter hosszúnak kell lennie!");
218         return false;
219     }
220     // Jelszó karakterellenőrzés
221     let jelszol = e.match(/[a-z]/g);
222     let jelszo2 = e.match(/[A-Z]/g);
223     let jelszo3 = e.match(/[0-9]/g);
224     //alert(jelszol + " " + jelszo2 + " " + jelszo3);
225     if (jelszol == null || jelszo2 == null || jelszo3 == null) {
226         alert("A jelszónak tartalmaznia kell kisbetűt, nagybetűt és számot!");
227         return false;
228     }
229     //alert("JS ellenőrzés vége");
230 }
231 </script>
```

2. Az űrlap elküldése (POST) előtt a *formEllenorzes()* függvény kerül meghívásra. Lefutásakor az alábbi adatokat ellenőrizzük:

- 'nev' mező ki van-e töltve (a tartalom nincs ellenőrizve),
- 'lak' mező ki van-e töltve (a tartalom nincs ellenőrizve),
- 'tel' mező ki van-e töltve (a tartalom ellenőrzése: hossz (min. 10 karakter); a '+' jelen kívül csak számokat tartalmaz-e),
- 'email' mező ki van-e töltve (a tartalmat az input mező típusa miatt (email), a form-ban ellenőrizzük),
- 'pw' mezők ellenőrzése: ki vannak-e töltve; a két mező értéke azonos-e; a beírt jelszó hossza megfelelő-e (min. 6 karakter); csak a meghatározott karaktereket tartalmazzák-e.

Ha bármelyik ellenőrzés miatt *false* értékkel tér vissza a függvény, hibaüzenetet kapunk az adott hibáról és az űrlap elküldése (POST) nem történik meg!

Ellenkező esetben a POST metódus lefut.

```
regisztracio.php
1  <?php
2
3  session_start();
4  include_once "FelhasznEll.php";
5  include_once "FelhasznReg.php";
6  include_once "CallApi_2.php";
7
8  if($_SERVER["REQUEST_METHOD"] == "POST")
9  {
10
11      if(isset($_POST["nev"]) && isset($_POST["lak"]) && isset($_POST["tel"]) &&
12          isset($_POST["email"]) && isset($_POST["pw1"]) && isset($_POST["pw2"]))
13      {
14          $nev = $_POST["nev"];
15          $lak = $_POST["lak"];
16          $tel = $_POST["tel"];
17          $email = $_POST["email"];
18          $pw1 = $_POST["pw1"];
19          $pw2 = $_POST["pw2"];
20          $_SESSION['Felhasznalonev'] = $nev; //Ezeket a Session-öket a hibás form adat miatti visszatöltéskor használjuk a html-ben!
21          $_SESSION['Lakcim'] = $lak;
22          $_SESSION['Telefon'] = $tel;
23          $_SESSION['Email'] = $email;
24          $_SESSION['pw1'] = $pw1;
25          $_SESSION['pw2'] = $pw2;
26
27          $hashpw = md5($pw1);
28
29          $f = new FelhasznEll();
30          $f->Nev = $nev;
31          $f->Email = $email;
32
33          $jsonfelh = json_encode($f);
34          $result = CallAPI("POST", "https://localhost:5001/Felhasznalok/Web", $jsonfelh);
35          $felh = json_decode($result[1]);
36
37          if ($_SESSION['Felhasznalonev'] == $felh->{"Nev"}){
38              echo "<script>alert('A felhasználónév már létezik, adj meg egy másikat!')</script>";
39          }
39          elseif ($_SESSION['Email'] == $felh->{"Email"}){
39      }
```

3. Az oldal betöltésekor indítjuk a *session_start()*-ot, utána pedig be include-oljuk a *FelhasznEll.php*, a *FelhasznReg.php* és a *CallApi_2.php*-t.

4. Ha az űrlap kitöltése megfelelő, indítjuk a POST metódust. Ha léteznek a *nev*-, *lakhely*-, *telefonszám*-, *email*- és *jelszó* mezők, akkor ezeket elmentjük változókba, majd az ezekhez tartozó SESSION-öket is lementjük. Erre itt azért van szükség, hogy ha a később indított CallApi hívás eredményének értékelésekor hibaüzenetet kapunk, a SESSION -ök megfelelő

változóit vissza tudjuk tölteni az űrlapba. Ezzel azt érjük el, hogy csak a hibás mezőt kell javítani.

5. A jelszómezőben eltárolt értéket *md5* típusú technológiával titkosítjuk (hash-eljük). Példányosítunk egy FelhasznEll osztályt, aminek az adattagjai megkapják az eltárolt név és email cím értékeit.

A \$jsonfelh változóban eltároljuk a json encode-olt példányt. A \$result változóban eltároljuk a CallAPI hívást, ami visszaadja a felhasználó nevét és email címét.

Majd a \$felh változóban eltároljuk a json decode-olt példányt, ami megkapja a \$result változó első indexén lévő értékeket.

```
32
33     $jsonfelh = json_encode($fe);
34     $result = CallAPI("POST", "https://localhost:5001/Felhasznalok/Web", $jsonfelh);
35     $felh = json_decode($result[1]);
36
37     if ($_SESSION['Felhasznalonev'] == $felh->{"Nev"]){
38         echo "<script>alert('A felhasználónév már létezik, adj meg egy másikat!')</script>";
39     }
40     elseif ($_SESSION['Email'] == $felh->{"Email"}){
41         echo "<script>alert('Ezzel az email címmel már regisztráltak, adj meg egy másikat!')</script>";
42     }
43     else{
44         $fe = new FelhasznReg();
45         $fe->Azon = 0;
46         $fe->Nev = $nev;
47         $fe->Lak = $lak;
48         $fe->Tel = $tel;
49         $fe->Email = $email;
50         $fe->Jog = 0;
51         $fe->Pw = $hashpw;
52
53         $jsonfelhReg = json_encode($fe);
54         $result2 = CallAPI("PUT", "https://localhost:5001/Felhasznalok", $jsonfelhReg);
55
56         $felh2 = json_decode($result2[1]);
57         if ($felh2 == NULL){
58             session_unset();
59             echo "<script>alert('Köszönjük a regisztrációt!')</script>";
60
61             header("Refresh:0"); // Ne ragadjanak be az adatok!!!!
62             echo "<script>location.href='belepes.php'</script>";
63         }
64     }
65   }
66 }
67 ?>
68
69
70 <!DOCTYPE html>
```

6. Leellenőrizzük, hogy már egy létező felhasználónévvel vagy email címmel akarnak-e regisztrálni, és ha igen, akkor figyelmeztető üzenetet küldünk a felhasználónak.

Különben példányosítunk egy *FelhasznReg()* osztályt, és az adattagjainak átadjuk a hozzájuk kapcsolódó változókba mentett értékeket. Kivéve az *Azon* és *Jog* adattagokat, amik nulla értéket kapnak.

Ezután ugyanúgy encode-oljuk és decode-oljuk a CallAPI hívást, mint a *FelhasznReg()* példányosításnál.

Majd, ha a \$felh2 változó értéke NULL, akkor töröljük az összes SESSION-ben tárolt értéket és megköszönjük a regisztrációt. Lenullázzuk az oldalfrissítést és elnavigálunk a *belépés* oldalra.

belepes.php

```
belepes.php
1 <?php
2 session_start();
3 include_once "Felhasznalo.php";
4 include_once "CallApi_2.php";
5
6 if(isset($_SESSION["Azonosito"]))
7 {
8     header("Location: ./foglalas.php");
9 }
10
11 if($_SERVER["REQUEST_METHOD"] == "POST")
12 {
13     if(isset($_POST["email"]) && !empty($_POST["email"]) &&
14         isset($_POST["pw1"]) && !empty($_POST["pw1"]))
15     {
16         $email = $_POST["email"];
17         $pw1 = $_POST["pw1"];
18         $hashpw = md5($pw1);
19
20         3 $f = new Felhasznalo();
21         $f->Email = $email;
22         $f->Pw = $hashpw;
23
24         $jsonfeh = json_encode($f);
25         var_dump($jsonfeh);
26         echo "<br><br>";
27         $result = CallAPI("POST", "https://localhost:5001/Felhasznalok", $jsonfeh);
28
29         $token = $result[0];
30         $feh = json_decode($result[1]);
31
32         4 if ( $feh != NULL){
33             $_SESSION['Azonosito'] = $feh->{"Azon"};
34             $_SESSION['Felhasznalonev'] = $feh->{"Nev"};
35             $SESSION['Token'] = $token;
36             echo "<script>alert('Köszöntjük weboldalunkon!')</script><br />";
37             echo "<script>location.href = 'foglalas.php'</script>";
38         }
39     }
```

1. Az oldal betöltődésekor elindítjuk a `session_start()` függvényt, include-oljuk a `Felhasznalo.php`-t és a `CallApi_2.php`-t.
2. Ha a SESSION-ben van `Azonosito` (felhasználó azonosító), akkor átirányítjuk a `foglalas.php` oldalra.
3. Ha POST metódussal történik a kérés, akkor megnézzük, hogy léteznek-e és nem üresek az email és jelszó mezők. Majd változókban eltároljuk az értékeiket. A jelszót szintén `hash`-eljük `md5`-tel. Végül példányosítunk egy `Felhasznalo()` osztályt és az adattagjainak átadjuk az email címét és a `hash`-elt jelszót.
4. Ezután `json` formátumúvá encode-oljuk ezt a példányt, meghívjuk rá a `CallAPI` függvényt, és a `$token` változóba elmentjük a kapott tokent, a `$feh` változóba pedig a `json` dekódolt felhasználót.

```

belepes.php
11
12 if($_SERVER["REQUEST_METHOD"] == "POST")
13 {
14     if(isset($_POST["email"]) && !empty($_POST["email"]) &&
15         isset($_POST["pw1"]) && !empty($_POST["pw1"]))
16     {
17         $email = $_POST["email"];
18         $pw1 = $_POST["pw1"];
19         $hashpw = md5($pw1);
20
21         $f = new Felhasznalo();
22         $f->Email = $email;
23         $f->Pw = $hashpw;
24
25         $jsonfeh = json_encode($f);
26         var_dump($jsonfeh);
27         echo "<br><br>";
28         $result = CallAPI("POST", "https://localhost:5001/Felhasznalok", $jsonfeh);
29
30         $token = $result[0];
31         $feh = json_decode($result[1]);
32
33         if ( $feh != NULL){
34             $_SESSION['Azonosito'] = $feh->{"Azon"};
35             $_SESSION['Felhasznalonev'] = $feh->{"Nev"};
36             $_SESSION['Token'] = $token;
37             echo "<script>alert('Köszöntjük weboldalunkon!')</script><br />";
38             echo "<script>location.href = 'foglalas.php'</script>";
39         }
40         else{
41             echo "<script>alert('Hibás email cím, vagy jelszó!')</script>";
42         }
43     }
44 }
45 ?>
46
47 <!DOCTYPE html>
48 <html lang="hu">

```

5

5. Hogy ha a `$feh` nem üres, akkor az `Azonosito` és `Felhasznalonev` SESSION-ökben eltároljuk a `$feh Azon` és `Nev` értékeit, a `$token`-t pedig a SESSION `Token`-ben.

Echo-val kiíratunk egy javascript alertet, amiben üdvözöljük a felhasználót az oldalon.

Szintén ki echo-zunk egy javascript átirányítást a `foglalas.php` oldalra.

Különben, ha a `$feh` üres, akkor egy figyelmeztető üzenetet jelenítünk meg a felhasználónak jelezve, hogy hibás email címet vagy jelszót adhatott meg.

foglalas.php

```
 95     include_once('navvar.php');
96  ?>
97
98 <br />
99
100 <div class="container">
101   <div class="ujfelhasznalo">
102     <form method="POST" action= "">
103
104       <div>
105         <h2> Üdvözöljük <?php echo ". $_SESSION['Felhasznalonev'] ?> !</h2>
106         <h3> Foglalásához kérjük adja meg a vendégek számát, a dátumot és az időpontot!</h3>
107       </div>
108
109       <br /><br />
110
111       <div>
112         <label class="labela">Vendégek száma:</label><br />
113         <select class="bevitel" name="szemelydb">
114           <option>1</option>
115           <option>2</option>
116           <option>3</option>
117           <option>4</option>
118           <option>5</option>
119           <option>6</option>
120           <option>7</option>
121           <option>8</option>
122         </select>
123
124       <div>
125         <label class="labela">Dátum: </label><br />
126         <input type="date" class="bevitel" name="datum">
127       </div>
128
129       <div>
130         <label class="labela">Időpont:</label><br />
131         <select class="bevitel" name="idopont">
132           <option>12:00:00</option>
133           <option>13:00:00</option>
134           <option>14:00:00</option>
135           <option>15:00:00</option>
136           <option>16:00:00</option>
137           <option>17:00:00</option>
138           <option>18:00:00</option>
139           <option>19:00:00</option>
140           <option>20:00:00</option>
141           <option>21:00:00</option>
142         </select>
143
144       <br />
145
146       <input type="submit" class="btn btn-success" value="Foglalom!">
147
148       <br /><br />
149
150       <a class="btn btn-primary" href="foglalasaim.php">Előző foglalásaim</a>
151
152       <br /><br />
153
154       <input type="submit" name="kilepes" class="btn btn-danger" value="Kilépés">
155
156     </form>
157   </div>
158
159 <br/>
160
161 <footer class="page-footer text-center wow fadeIn">
162   <div class="py-3 bg-dark">
```

1.1

```
124       <label class="labela">Dátum: </label><br />
125       <input type="date" class="bevitel" name="datum">
126     </div>
127
128
129     <div>
130       <label class="labela">Időpont:</label><br />
131       <select class="bevitel" name="idopont">
132         <option>12:00:00</option>
133         <option>13:00:00</option>
134         <option>14:00:00</option>
135         <option>15:00:00</option>
136         <option>16:00:00</option>
137         <option>17:00:00</option>
138         <option>18:00:00</option>
139         <option>19:00:00</option>
140         <option>20:00:00</option>
141         <option>21:00:00</option>
142       </select>
143
144     <br />
145
146     <input type="submit" class="btn btn-success" value="Foglalom!">
147
148     <br /><br />
149
150     <a class="btn btn-primary" href="foglalasaim.php">Előző foglalásaim</a>
151
152     <br /><br />
153
154     <input type="submit" name="kilepes" class="btn btn-danger" value="Kilépés">
155
156   </form>
157 </div>
158
159 <br/>
160
161 <footer class="page-footer text-center wow fadeIn">
162   <div class="py-3 bg-dark">
```

1.2

1. A HTML részben két `<div>`-en belül található a `<form>`. Ezen belül szintén `<div>`-ekben találhatóak meg a beviteli mezők és gombok.

A `<h2>`-es tagek között php-vel jelenítjük meg a felhasználó nevét, mikor üdvözöljük az oldalon.

```
foglalas.php
1 <?php
2 session_start();
3 include_once "CallApi_2.php";
4 include_once "Foglal.php";
5
6 if($_SERVER["REQUEST_METHOD"] == "POST")
7 {
8     if(isset($_POST["szemelydb"]) && !empty($_POST["szemelydb"]) &&
9         isset($_POST["datum"]) && !empty($_POST["datum"]) &&
10        isset($_POST["idopont"]) && !empty($_POST["idopont"]))
11     {
12         $token = $_SESSION['Token'];
13         $azon = $_SESSION['Azonosito'];
14         $nev = $_SESSION['Felhasznalonev'];
15         $szemelydb = $_POST["szemelydb"];
16         $datum = $_POST["datum"];
17         $idopont = $_POST["idopont"];
18
19         $foglalas = $idopont.$datum;
20
21         $foglalasido = date(DATE_ATOM, strtotime($foglalas));
22
23         $date = new DateTime('NOW');
24         $leadva = $date->format(DateTime::ATOM);
25
26         $date1 = date_create($datum);
27         $date2 = date("Y-m-d");
28
29         $ido1 = date_timestamp_get($date1);
30         $date2Creat = date_create($date2);
31         $ido2 = date_timestamp_get($date2Creat);
32         $idoKulonbseg = $ido1 - $ido2;
33
34         //echo $idoKulonbseg;
35
36         if($idoKulonbseg < 0){
37             echo "<script>alert('Az Ön által megadott időpont korábbi a mai napnál!
38                         másik időpontot!')</script>";
39             //echo "<script>location.href='foglalas.php'</script>";
39                                         Kérjük adjon meg egy
```

2. Az oldal betöltésekor meghívjuk a `session_start()` függvényt. Include-oljuk a `CallApi_2.php` és a `Foglal.php` fájlokat.

3. Ha a POST-ban kapott `szemelydb`, `datum` és `idopont` mezők léteznek és nem üresek, akkor elmentjük az értékeiket az ugyanilyen nevű változókba. És a SESSION-ben tárolt `Felhasznalonev`, `Azonosito` és `Token` értékeit is elmentjük ugyanúgy változókba.

A `$foglalas` változóban tároljuk el összefűzve, az `$idopont` és `$datum` változók értékeit.

4. A `$foglalasido` változóba elmentjük a `date()` függvény által vissza adott értéket. Az első paramétere a `DATE_ATOM` előredefiniált konstans, ami visszaadja számokkal a dátumot és az időt is (egyfajta formátumként pl. `2022-04-20T15:52:01+00:00`). A második paraméterében pedig meghívjuk a `strtotime()` függvényt, átadva neki paraméterben a `$foglalas` változót, amelyet átalakít Unix időbeléyéggé (`1970-01-01` óta eltelt másodpercek száma).

A `$date` változóban létrehozunk egy `DateTime('NOW')` példányt, ami az aktuális időt fogja minden tárolni.

A `$leadva` változóban eltároljuk a `$date`-ben lévő példány megfelelő formátumú időbeléyegét.

A `$date1`-ben elmentjük az űrlapmezőből kapott és a `date_create()` függvénynek átadott dátumot.

A `$date2`-ben eltároljuk az aktuális dátumot év-hónap-nap formátumban (pl. 2022-04-20).

Az `$ido1`-ben tároljuk a `date_timestamp_get()`-el Unix formátumúvá alakított `$date1` beli dátumot.

A `$date2Create`-ben készítünk egy új dátumot, a `$date2`-ben tárolt string-ből.

Majd az `$ido2` változóban is Unix formátumúvá alakítjuk a `$date2Create`-ben tárolt dátumot.

Ezután az `$idokulonbseg` változóban eltároljuk az `$ido1`-ből kivont `$ido2` különbségét.

```
foglalas.php
32     $idokulonbseg = $ido1 - $ido2;
33
34     //echo $idokulonbseg;
35
36     if($idokulonbseg < 0){
37         echo "<script>alert('Az Ön által megadott időpont korábbi a mai napnál!';
38             másik időpontot!')</script>";
39         //echo "<script>location.href='foglalas.php'</script>";
40     }
41     else{
42         $f = new Foglal();
43         $f->Fazon = 0;
44         $f->Azon = $azon;
45         $f->Szemelydb = $szemelydb;
46         $f->Foglalasido = $foglalasido;
47         $f->Leadva = $leadva;
48         $f->Megjelent = false;
49
50         $jsonfoglal = json_encode($f);
51         $result = CallAPI("POST", "https://localhost:5001/Foglalasok", $jsonfoglal, $token);
52         echo "<script>alert('Köszönjük a foglalást!')</script>";
53         header("Refresh:0"); // Ne ragadjanak be az adatok!!!
54         //echo "<script>location.href='foglalas.php'</script>";
55     }
56
57     //kijelentkezésnél is elküldődik a form, de a post arraybe benne lesz a kilépés gomb is
58     if(isset($_POST["kilepes"]))
59     {
60         session_unset();
61         header("Location: index.php");
62     }
63
64
65 ?>
66
67 <!DOCTYPE html>
68 <html lang="hu">
```

The code editor shows a PHP script named 'foglalas.php'. The code handles date calculations, creates a booking object (\$f), sends a POST request to a local API, and handles a logout action. Two specific sections are highlighted with red boxes and numbered 5 and 6:

- Section 5:** This section highlights the part where the script calculates the difference between two dates (\$ido1 and \$ido2) and checks if it's negative. If it is, it displays an alert indicating the input date is earlier than today. It also includes code to redirect back to the booking page.
- Section 6:** This section highlights the logout logic. It checks if the 'kilepes' (logout) button was pressed via a POST request. If so, it unsets the session and redirects the user to the main index page.

5. Ha az `$idokulonbseg` kisebb nullánál, echo-val megjelenítünk egy javascript alertet, amelyben közöljük a felhasználóval, hogy az általa megadott dátum, korábbi az aktuális dátumnál, és adjon meg mai, illetve későbbi dátumot.

Különben az `$f` változóban példányosítunk egy `Foglal()` osztályt és az adattagjainak megadjuk a korábban hozzájuk kötődő változókat. Kivéve a foglalás azonosítót (`Fazon`), aminek a nulla értéket adjuk meg (ez az adatbázisban 'autoincrement'). A `Megjelent`-et `false`-ra állítunk.

Ezután a `$jsonfoglal`-ban `json_encode()`-oljuk ezt a példányt. A `$result`-ban meghívjuk a `CallAPI()`-t, és átadjuk neki a `$jsonfoglal`-t és a `$token`-t is paraméterben.

Majd megköszönjük a foglalást egy alert üzenettel, és a `header`-nek megadjuk, hogy frissítse az oldalt.

6. Ha a POST-ban kapott *kilepes* gomb létezik, akkor kitöröljük az összes SESSION-ben tárolt változót, és a *header()*-el átirányítunk a főoldalra.

foglalasaim.php

```

foglalasaim.php
1 <?php
2 session_start();
3 include "CallApi_2.php";
4
5 $azon = $_SESSION['Azonosito'];
6 $token = $_SESSION['Token'];
7 // adott felhasználóhoz tartozó foglalások lekérdezése
8 $foglalasaim = CallAPI("GET", "https://localhost:5001/Foglalasok?feh=". $_SESSION['Azonosito'], false, $token);
9
10 $foglalasdecod = json_decode($foglalasaim[1]); // A visszakapott válasz json formátummá alakítjuk, a html-ben használjuk fel!
11 //Foglalás törlése
12 if($_SERVER["REQUEST_METHOD"] == "POST")
13 {
14     $storlesAzon = $_POST["fazon"];
15     $foglalastorles = CallAPI("DELETE", "https://localhost:5001/Foglalasok/$storlesAzon", false, $_SESSION['Token']);
16     header("Refresh:0");
17 }
18
19 ?>
20
21 <!DOCTYPE html>
22 <html lang="hu">
23
24     <head>
25         <meta charset="UTF-8">
26         <meta http-equiv="X-UA-Compatible" content="IE=edge">
27         <meta name="viewport" content="width=device-width, initial-scale=1.0">
28         <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
29             integrity="sha384-1BmE4kBWBq781YhfIdvkunfTAU6au8T94wHFtjDbrCEXSUloBoqyl2qvZ6jiW3" crossorigin="anonymous">
30         <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
31             integrity="sha384-ka7Sk0Gln4gmtz2M1QnikT1wXgYsOg+OMhuPflRH9sENB00LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
32         <link rel="stylesheet" href="css/style.css">
33         <title>Burger Étterem</title>
34     </head>
35
36     <body class="container bg-dark">
37         <header>
|           <div class="row">

```

1. Elindítjuk a *session_start()*-ot, be *include*-oljuk a *CallApi_2.php*-t és elmentjük változókba a SESSION-ben tárolt *Azonosito*-t és *Token*-t.
2. Létrehozzuk a *\$foglalasaim* változót és meghívjuk benne a *CallAPI()*-t, aminek a második paraméterében hozzáfűzzük az URL-hez a SESSION-ben tárolt azonosítót, a végén pedig a tokent. A vissza kapott *json* objektumú választ PHP objektummá alakítjuk a *json_decode()* függvénytel, és elmentjük a *\$foglalasdecod* változóba.
3. Ha POST metódussal történik a kérés, akkor a *\$storlesAzon*-ba elmentjük a kapott foglalás azonosítót (*fazon*). A *\$foglalastorles*-ben meghívjuk a *CallAPI()*-t "DELETE" kéréssel, és a végén átadjuk paraméterben a tokent. A végén pedig frissítjük az oldalt a *header()*-el.

```

foglasaim.php
47     include_once("navbar.php");
48
49
50 <br />
51
52 <div class="container">
53
54     <div class="ujfelhasznalo">
55         <div>
56             <h2>Kedves <?php echo $_SESSION['Felhasznalonev']?>!</h2>
57             <h3>Éttermünkben az alábbi foglalásai vannak / voltak:</h3>
58         </div>
59     </div>
60
61 <br/>
62
63     <div class="ujfelhasznalo">
64
65         <table class="table table-dark table-striped">
66
67             <thead >
68                 <tr>
69                     <th>Foglalás azonosító</th>
70                     <th>Vendégek száma</th>
71                     <th>Foglalás időpontja</th>
72                     <th>Foglalás leadásának ideje</th>
73                     <th>Foglalás törlése</th>
74                 </tr>
75             </thead>
76
77             <tbody>
78                 <?php
79                     $sor = 0;
80                     while($sor < count($foglalasdecod)) //ciklus a dekódolt eredmény végéig
81                     {
82                         //A db-ból kapott időpontok átalakítása a megjelenítéshez
83                         $fogdate=date_create($foglalasdecod[$sor]->{"Foglalasido"});
84                         $leaddate = date_create($foglalasdecod[$sor]->{"Leadva"});
85                         echo
86                         "<tr>
87                             <td>".$foglalasdecod[$sor]->{"Fazon"}."</td>
88                             <td>".$foglalasdecod[$sor]->{"Szemelydb"}."</td>
89                             <td>".date_format($fogdate, "Y.m.d H:i:s")."</td>
90                             <td>".date_format($leaddate, "Y.m.d H:i:s")."</td>
91                             <td>
92                                 <form method='POST' onsubmit='return megerosites()'\>
93                                     <input type='text' name='fazon' id='fazon' style='display: none;' value='".$foglalasdecod[$sor]->{"Fazon"}."'";
94                                     if ($foglalasdecod[$sor]->{"Megjelent"} == false){
95                                         echo"<input type='submit' class='btn btn-danger' value='Törlés'>";
96                                     }
97                                     else{
98                                         echo"<input type='submit' class='btn btn-danger' value='Törlés' disabled>";
99                                     }
100                                     echo"</form>
101                             </td>
102                         </tr>";
103                         $sor++;
104                     }
105                 ?>
106             </tbody>
107         </table>
108     </div>
109
110     <br />

```

4. Itt ugyanúgy egy php-s kiíratás történik, mint korábban. A `<h2>` tagok között kiíratjuk a SESSION-ben tárolt felhasználó nevét.

```

foglasaim.php
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111

```

5. Egy `<div>`-ben létrehozunk egy `<table>` tag-et, azon belül pedig egy `<thead>` és egy `<tbody>` tag-et. A `<tbody>` tagjei között php-val oldjuk meg a sorok és a bennük lévő adatok megjelenítését.

Létrehozunk egy `$sor` változót, aminek a *nulla* értéket adjuk. Egy *while* ciklussal addig megyünk, amíg a `$sor` kisebb a dekódolt adatok számánál. A `$fogdate`-be elmentjük a foglalás idejét (*Foglalasido*), a `$leaddate`-be pedig a leadás idejét (*Leadva*).

Ezután ki *echo*-zunk egy sort (`<tr>`), és a sorban található adatokat (`<td>`). A `<td>` nyitó tagjei után fűzzük a `$foglalasdecod` `$sor`-adik indexén lévő *Fazon* és *Szemelydb* értékeit, és a `date_format()`-nak átadott foglalási és leadási időket. Majd utána fűzzük a lezáró `</td>` és az új `<td>` tag-eket.

A foglalás azonosító, a vendégek száma, a foglalás időpontja és a foglalás leadásának ideje után az utolsó `<td>` tag-ek között létrehozunk egy `<form>`-ot POST metódussal és egy `onsubmit` tulajdonsággal, aminek a paraméterében visszatérünk a `megerosites()` függvénytel. Ezután készítünk egy `'text'` típusú input mezőt, aminek kikapcsoljuk a láthatóságát, és az értékének a foglalás azonosítóját adjuk meg.

Ha a `$foglalasdecod` `$sor`-adik indexének a *Megjelent* értéke hamis, vagyis még nem ment el a foglalt időpontra, akkor megjelenítünk egy `'submit'` típusú input-ot, aminek értékül a *'Törlés'*-t adjuk meg és *bootstrap*-el egy piros gomb szerű hatást rendelünk hozzá. Ezzel létrehozva egy törlés gombot, amivel a felhasználó törölni tudja a leadott foglalását.

Viszont, ha nem törölte és megjelent a foglalt időpontban, akkor ez az input mező megkapja a `'disabled'` paramétert, amivel kikapcsolttá, nem kattinthatóvá válik a törlés gomb.

```
foglalasaim.php
107     |         </table>
108   |     </div>
109   |
110   |     <br />
111   |     <br />
112
113
114     <div class="ujfelhasznalo">
115       |       <a class="btn btn-success" href="foglalas.php">Vissza a foglalásokhoz</a>
116     </div>
117
118   </div>
119
120   <br />
121
122   <footer class="page-footer text-center wow fadeIn">
123     <div class="py-3 bg-dark">
124       |       <span class="footer-copyright text-secondary center" id="copyright">© 2022 Copyright:</span>
125       |       <span class="text-secondary center">Burger Étterem</span>
126     </div>
127   </footer>
128
129   <script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>
130   <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
131   <script src="app.js"></script>
132 </body>
133 </html>
134
135 <script>
136   function megerosites(){
137     if(confirm("Valóban törölni kívánja a foglalását?")){
138       |       return true;
139     }
140     else{
141       |       return false;
142     }
143   }
144 </script>
145
```

6. <script> tag-ek között javascripttel készítünk egy függvényt *megerosites()* néven.

Egy „*confirm()*” üzenettel megerősítést kérünk a törlés végrehajtására. Ha a *megerosites()* függvény igaz értékkel tér vissza (az ’OK’ gombra kattint a felhasználó), akkor elküldődik a <form>, és az adott foglalás azonosítójú sor törlődik az adatbázisból és a táblázatból is.

Ellenkező esetben (a ’Mégse’ gombra kattint), a műveletet megszakítjuk.

CallApi_2.php

```
CallApi_2.php
1  <?php
2
3  function CallAPI($method, $url, $data = false, $token = null )
4  {
5      $curl = curl_init($url);
6
7      switch ($method)
8      {
9          case "POST":
10             //curl_setopt($curl, CURLOPT_POST, 1);
11             curl_setopt($curl, CURLOPT_CUSTOMREQUEST, 'POST');
12             curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type: application/json'));
13             if ($data){
14                 curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
15                 //echo ("Burger!");
16             }
17             break;
18
19          case "PUT":
20             //curl_setopt($curl, CURLOPT_PUT, 1);
21             curl_setopt($curl, CURLOPT_CUSTOMREQUEST, 'PUT');
22             curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type: application/json'));
23             if ($data){
24                 curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
25                 //echo ("Felv  ve!");
26             }
27             break;
28
29          case "DELETE":
30             curl_setopt($curl, CURLOPT_CUSTOMREQUEST, 'DELETE');
31             curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type: application/json'));
32             break;
33
34          default:
35             if ($data){
36                 $url = sprintf("%s?%s", $url, http_build_query($data));
37             }
38     }
39 }
```

A *CallApi_2.php* fájlban található *CallAPI()* függvény segítségével tudunk kommunikálni az adatbázissal és adatokat létrehozni, frissíteni, valamint törölni. Paramétereiben meg kell adni a metódus típusát, az API url címét, egy *\$adat*-ot ami kezdetben *false* értéket kap és egy *\$token*-t, ami *null* értéket vesz fel.

A *\$curl* változóban elmentjük a *curl_init()* függvényhívást, átadva neki az *\$url*-t paraméternek.

Egy *switch* elágazással megnézzük, hogy milyen metódussal történik a hívás. Majd a metódus típusától függ  en be  ll  tjuk a *curl_setopt()* függvény paramétereit. Ha van adat, akkor erre is be  ll  tjuk a *curl_setopt()* függvény megfelel   paramétereit.

A *default*   ban, ha van adat, akkor az *\$url*-ben meghívjuk a *sprintf()* függ  nyt, megadva paraméterben a form  atumot (a *%s* hely  re behelyettes  t  dik sorrendben, a k  vetkez   param  terekben megadott *string*), az *url*-t és a *http_build_query()*-t átadva neki a *\$data*-t, ami *url*-   enk  dolja a kapott adatokat, vagyis a *json* enk  dolt objektumot.

```
CallApi_2.php
38     }
39
40     if($token != null)
41     {
42         curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type: application/json','Auth:'.$token));
43     }
44
45     //headers = array("Auth:+");
46     curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
47     curl_setopt($curl, CURLOPT_HEADER, 1); //kérnénk a headeröket is
48     //curl_setopt($curl, CURLOPT_HTTPHEADER, $headers)
49     curl_setopt($curl, CURLOPT_SSL_VERIFYPeer, false);
50     //curl_setopt($curl, CURLOPT_HEADER, true);
51
52     $result = curl_exec($curl);
53
54     $header_size = curl_getinfo($curl, CURLINFO_HEADER_SIZE); //megnézzük milyen hosszú a header rész
55     $authIndex = strpos($result,"auth: "); // megkeressük az auth: string pozícióját
56     $header = "";
57
58     if($authIndex) //ha kapunk auth stringet
59     {
60         $header = substr($result,$authIndex+6,36); //ha megvan hozzáadunk 6ot, auth = 4 + : + space, C# generált GUID 36 hosszú
61     }
62     else
63     {
64         $header = substr($result,0,$header_size); //minden header
65     }
66     $body = substr($result, $header_size); //a body meg a maradék
67
68     //var_dump(curl_error($curl));
69     curl_close($curl);
70
71     $resultArray = array($header,$body);
72     return $resultArray;
73 }
74 ?>
75 }
```

Ha van `$token`, akkor a `curl_setopt()`-nak átadjuk ezt a tokent.

Majd további `curl_setopt()` beállítások következnek.

A `$result`-ban meghívjuk a `curl_exec()` függvényt, és átadjuk neki a `$curl`-ben tárolt adatokat, ami végrehajtja azokat.

A `$header_size`-ban a `curl_getinfo()`-val lekérjük, mekkora a `header` mérete.

Az `$authIndex`-ben a `strpos()`-al megkeressük a `$result`-ben tárolt adatok közül az `"auth: "` string helyét.

Létrehozunk egy `$header` változót és egy üres stringet adunk neki értékül.

Ezután megnézzük, hogy az `$authIndex` tartalmazza-e az `"auth: "` stringet, és ha igen, akkor a `$header`-ben eltároljuk a C# által generált `GUID`-ot, ami 36 karakter hosszú. Ha nem tartalmazza, akkor pedig a teljes headert eltároljuk.

A `$body`-ban eltároljuk a `header` utáni részt, vagyis a `body`-t és ami maradt.

Majd a `curl_close()`-al lezárjuk a `$curl` munkamenetet.

A `$resultArray`-ben eltároljuk tömbként a `$header`-t és a `$body`-t, és ezzel a tömbbel térünk vissza a függvényhívás helyén.

app.js

Ez az egyetlen *.js* kiterjesztésű fájl. Ebben van *React*-al megoldva, az oldalak tetején található éttermi képek váltakozása.

```
1 1 let kepek = ["images/hatter/Liverpool_St_crop.jpg", "images/hatter/LiverpoolSt_05_crop.jpg",
2 2 "images/hatter/DSC_0921_Copy_crop.jpg", "images/hatter/burger-foto-3_crop.jpg",
3 3 "images/hatter/burger-foto-2_crop.jpg", "images/hatter/burger-foto-1_crop.jpg"];
4
5 4 let aktualis = 0;
6 ReactDOM.render(React.createElement(App), document.getElementById("headerimage_carussel"));
7
8 function App() {
9   const [kep, setKep] = React.useState(aktualis);
10  React.useEffect(() => {
11    const lapozo = setInterval(() => {
12      if (aktualis < 5) {
13        aktualis++;
14      } else {
15        aktualis = 0;
16      }
17      console.log(aktualis);
18      setKep(aktualis);
19    }, 3000); // clearing interval
20
21    return () => clearInterval(lapozo);
22  });
23  return React.createElement("img", {
24    src: kepek[kep],
25    style: {
26      width: "1300px",
27      height: "360px",
28      marginLeft: "auto",
29      marginRight: "auto"
30    }
31  });
32}
33
```

1. Létrehozunk egy *kepek* tömböt, melyben string-ként tároljuk el a fejlécben lévő képnézegetőben megjelenő képek elérési útjait.

2. Létrehozzuk az *aktualis* változót nulla értékel. A későbbiekben ennek a változónak a segítségével jelöljük ki a *kepek* tömbből, hogy melyik képet jelenítsük meg a képnézegetőben.

A *ReactDOM.render()*-nek megadjuk, hogy hozzon létre egy elemet az *App()* függvény által vissza adott eredményből. És megadjuk az elem helyét, amit a megfelelő HTML tag *id*-je ad meg.

3. Létrehozzuk az *App()* függvényt. Beállítjuk, hogy mindenkorán az *aktualis* változó értékének megfelelő képet jelenítse meg. Majd a képek váltásához meghívjuk a *setInterval()* függvényt és megadjuk neki, hogy amíg az *aktualis* változó értéke kisebb ötnél, addig növelje az értékét, amíg el nem éri az ötöt. Különben újra beállítjuk nullára az értékét. A *setInterval()* második paramétere azt adja meg, hogy milyen időnként történjen a váltás (3000 millisec, vagyis 3 másodpercenként).

Majd visszatérünk a *clearInterval()* függvénnnyel, amivel leállítjuk a képek váltását. Ezt mindenkorán a *React.useEffect()* függvényén belül hoztuk létre.

4. Ezután visszatérünk a *React.createElement()*-el. Megadjuk, hogy *img* tag-et hozzon létre, majd a képhez tartozó tulajdonságokat azok paramétereivel, mint a képek forrása és stílusa.

C# ASZTALI ALKALMAZÁS

Az adatbázis és weboldal leírása után elérkeztünk a projekt vastagklienses és szerveres részéhez.

Az API (Application Programming Interface) ismertetése fog következni az áttekintés után, ez a szoftver lesz a híd a weboldalunk(vékony kliens) / WPF applikáció(vastagkliens) és az adatbázis között.

A vastagklienses program az étterem belső működésének megkönnyítéséhez lett létrehozva.

Az adatbázisban (MySQL) lévő adatok lekérésére és módosítására (RESTful) ASP.NET Core Web API-n keresztül van lehetőség, amely a szerver oldali részen fut. A weboldal cURL-ön keresztül, a vastagklienses program HttpClient-el éri el az API endpoint-jait.

Felhasznált technológiák:

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

Rövid leírás:

.NET:



MySQL: Egy nyílt forráskódú, ingyenes, relációs adatbázis kezelő rendszer, amit az Oracle fejlesztett ki, SQL nyelvet használva. Kezelése egyszerű, jól teljesíti a neki szánt feladatokat.

ASP.NET Core: Egy nyílt forráskódú, ingyenes, Web keretrendszer, az eredeti ASP.NET-nek a továbbfejlesztett változata. Ez mostmár magába foglalja a korábban külön fejlesztett ASP.NET MVC Web alkalmazás keretrendszerét, 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 megkötlese 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 hogy tartalmazzon (levéve a terhet az API-ról), ezáltal egy RESTful API könnyedén ki tud szolgálni egyszerre több millió felhasználót is.

HTTP: Egy kérés-válasz protokoll a kliens-szerver modellben. A kliens egy HTTP kérést küld, amelynek egy Request-line, Header, Body(nem minden esetben) része van. A Request-line tartalmazza a HTTP metódust(GET,POST,PUT,PATCH,DELETE stb..), a kérés útvonalát (általában egy URL), és a HTTP verziót. A header további információt tartalmaz a kérésről, pl: body tartalmának a típusát, CORS típust stb. A kérés tartalma a body-ba fog kerülni, ennek a formátuma a headerben van definiálva.

URL: Egyedi cím, ami egy valamilyen tartalom (weboldal, kép, videó) elérési útvonalát adja meg. Felépítése a következő: protokoll: http,https,ftp,mailto,file,data stb.. ami után egy kettőspont (:) írandó. Tartománynév (pl: google.com) amely elő két perjel (//) írandó. Portsárm (pl: 80, 443 stb.) amely elő kettőspont (:) írandó. Ez gyakran elhagyható, http kéréseknel az alapértelmezett port 80, https-nél 443. Elérési út a célgépen (pl api, api/pelda stb.) amely elő per jel írandó (/). Ezt követheti opcionálisan query paraméter amely „?”-el kezdődik. Ezek általában kulcs-érték párosok, több párost „&” jel-el választunk el egymástól. Pl: <https://pelda.hu/api/test> vagy <https://pelda.hu/api/test?elso=8&masodik=3>

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

MVVM (Model-View-ViewModel): Egy olyan programtervezési minta, amely a felhasználói felületet (View) különíti el teljesen az üzleti és megjelenítési logikától. A felhasználói felület “Databinding”-al kötődik az elkülönített megjelenítési logikához(ViewModel), amely különböző adatszerkezeteket és Modellek 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 hozzájárulnánk.
- XAML alapú könyvtárak/keretrendserek (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 minden 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 könnyebben újra felhasználható minden része, és sokkal könnyebben tesztelhető, mert nem függnek egymástól.
 - Ha a felépítés megfelelően van alkalmazva:
 - o Egy model nem tartalmaz semmilyen referenciát a View vagy ViewModel-re,
 - o Egy ViewModel nem tartalmaz semmilyen referenciát a View-ra,
 - o Egy View csak Databinding-al kapcsolódik a ViewModelhez, és az UI mögötti kód csak és kizárálag 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 keretrendszerbe/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 verziójánál):

- 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. A projecten belül NuGet csomagként van implementálva.

NuGet: Központi csomag tároló, segítségével különböző hasznos eszközöket lehet készíteni és megosztani más fejlesztőkkel. Egy NuGet csomag lényegében egy sima tömörített file, aminek a kiterjesztése “.nupkg”, DLL-eket és egyéb fileokat tartalmaz.

Entity Framework Core: Egy nyílt forráskódú, adathozzáférésre 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.

Amennyiben van már létező adatbázisunk, tudunk készíttetni belőle egy szerkezetet, létrejön egy “DbContext” amely “DbSet<T>”-eket fog tartalmazni. Lehetséges a fordítottja is, írhatunk először kódot, amiből készíttetünk egy adatbázist.

- DbContext: Ez lesz a “híd” az adatbázisunk, és az applikációnk osztályai között. Amikor példányosítjuk, létrejön a kapcsolat az adatbázissal (az aktuálisan legfrissebb adatokkal), a benne lévő DbSet-eken tudunk módosításokat végezni. Ezek a változások csak akkor lesznek végrehajtva az adatbázison amikor elmentjük az adott DbContext változásait.
- DbSet<T>: Egy olyan gyűjtemény, ami egy táblát reprezentál az adatbázisunkból. A típusa az adott táblából készített model. Pl: Autok táblából lesz egy DbSet<Auto> Autok. Úgy tudjuk kezelní mint egy akármilyen más (IEnumerable,IQueryable) gyűjteményt, pl: hozzáadni egy új auto-t, törlni, frissíteni. Az adott DbContext mentésénél végre lesznek hajtva az adatbázisban is a változtatások.

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/modellekben 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], [HttpPost], [HttpPut] stb..) valamelyik kontrollerbe:

- Az API felépít egy új kontrollert
- A kontrolleren belül készül egy új [DbContext](#) (esetünkben burgeretteremContext), ami tartalmazza az adatbázisunk legfrissebb 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)

```
burgeretteremContext _context = new();
```

```
// PUT api/<FelhasznalokController>
[HttpPut]
0 references
public IActionResult Put(Felhasznalo f)
{
    Felhasznalo letezike = _context.Felhasznalos.FirstOrDefault(x => x.Email == f.Email);
    if(letezike is null && f.Jog < 4)
    {
        _context.Felhasznalos.Add(f);
        if (_context.SaveChanges() > 0)
            return StatusCode(201);
    }
    else
    {
        return StatusCode(409, "A felhasználó már létezik!");
    }

    return StatusCode(500);
}
```

- IActionResult-al tér vissza (a többi endpoint is), ami lehet StatusCodeResult vagy ObjectResult.
- Ez az endpoint nem vár autorizációs 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 0(Vendég), 1(Felszolgáló), 2(Szakács), 3(Pultos) és 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 409-es státusszal (Conflict).
- Ha akármi más történik (pl. Mentésnél 0 a visszatért érték, szóval nem változott semmi) akkor visszatérünk egy 500-as státusszal (Internal Server Error).

Admin szükséges (Köret módosítása)

```
// PUT /<Koretek>
[HttpPost]
0 references
public IActionResult Put([FromHeader]string auth, Koret k)
{
    if(auth == AktivTokenek.AdminToken)
    {
        Koret aktk = _context.Korets.Find(k.Kazon);
        _context.Entry(aktk).CurrentValues.SetValues(k);
        if (_context.SaveChanges() > 0)
            return StatusCode(200);
        else
            return StatusCode(500);
    }

    return StatusCode(403);
}
```

- IActionResult-al tér vissza (a többi endpoint is), ami lehet StatusCodeResult vagy ObjectResult.
- Ez az endpoint vár Authorizációs stringet a headerból, 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 200-as státusszal (OK).
- Ha nem több mint 0 akkor visszatérünk 500-as státusszal (Internal Server Error).
- Ha a kapott token nem egyezik meg a generálittal akkor visszatérünk egy 403-as státusszal (Forbidden).

Admin vagy User szükséges (Tétel státusz módosítása)

```
// PUT /<Tetelek>/Status
[HttpPut("Status")]
0 references
public IActionResult PutStatus([FromHeader] string auth, Tetel t, [FromQuery]bool szakacs = false)
{
    if (auth == AktivTokenek.AdminToken || auth == AktivTokenek.UserToken)
    {
        Tetel aktt = _context.Tetels.Find(t.Tazon);
        if (szakacs)
            aktt.Etelstatus = t.Etelstatus;
        else
            aktt.Italstatus = t.Italstatus;

        if (_context.SaveChanges() > 0)
            return StatusCode(200);
        else
            return StatusCode(500);
    }

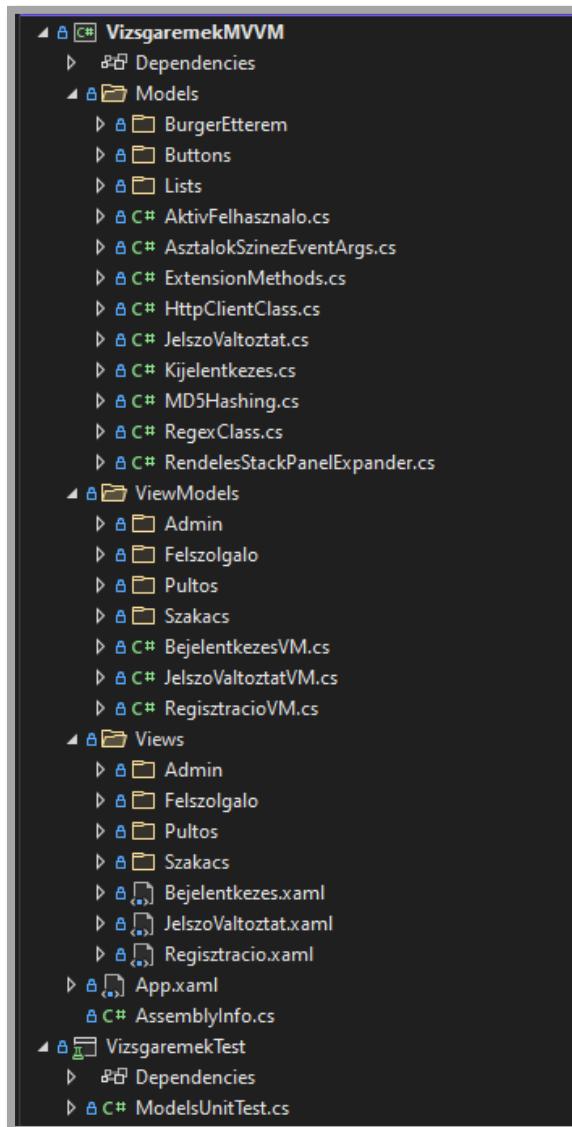
    return StatusCode(403);
}
```

- IActionResult-al tér vissza (a többi endpoint is), ami lehet StatusCodeResult vagy ObjectResult.
- Ez az endpoint vár Authorizációs stringet a headerból, amit leellenőrzünk az API futtatásakor generált GUID-kkel.
- A bodyból egy tételt vár JSON formátumban.
- Queryból (?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 térel azonosítójából).
- Ha queryból a szakács át lett állítva true-ra akkor:
 - A térel ételstátuszát változtatjuk
- Ha nem lett:
 - A térel italstátuszát változtatjuk
- Elmentjük a változtatásokat, ha több mint 0, szóval változott a DB-ben az adat, akkor visszatérünk 200-as státusszal (OK).
- Ha nem több mint 0 akkor visszatérünk 500-as státusszal (Internal Server Error).
- Ha a kapott token nem egyezik meg a generáltakkal akkor visszatérünk egy 403-as 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 megjelenítsi logikát, és Property -ket, amikhez a View Databindol.
- Models:** Ezek tartalmazzák az üzleti logikát, úgymond építőelemei az applikációnak, amiket a ViewModelbe raktunk általában Propertyként hogy Databindolható legyen.

Példa az applikáció működésére

View(FelszolgaloUI):

Ablak mögötti kód:

```
2 references
public FelszolgaloUI()
{
    InitializeComponent();
    FelszolgaloUIVM felszolgaloUIVM = new();
    DataContext = felszolgaloUIVM;
```

Az ablak konstruktorában készítünk a ViewModeljéből egy példányt, amit utána beállítunk az ablak DataContext-jének, ezáltal minden megjeleníti logika oda fog kerülni. Az ablak mögötti kód csak UI-val kapcsolatos kódot tartalmaz.

ViewModel(FelszolgaloUIVM):

```
internal class FelszolgaloUIVM : INotifyPropertyChanged
{
```

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.

```
... public interface INotifyPropertyChanged
{
    ... event PropertyChangedEventHandler? PropertyChanged;
```

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.

```
public event PropertyChangedEventHandler? PropertyChanged;
2 references
public void RaisePropertyChanged([CallerMemberName] string? propertyName = null)
{
    var handler = PropertyChanged;
    handler?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

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).

```

private int _kivalasztottAsztal = -1;
7 references
public int KivalasztottAsztal
{
    get => _kivalasztottAsztal;
    set
    {
        _kivalasztottAsztal = value;
        RaisePropertyChanged();
    }
}

```

- 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.

```

<ComboBox ItemsSource="{Binding Asztalok}" SelectedValue="{Binding KivalasztottAsztal}" Width="50px"></ComboBox>

```

- Egy DataBinding lehet:
 - kétirányú: a View tudja változtatni az értéket, ha az érték változik akkor a View is változik.
 - egyirányú (View->Kód): a View tudja változtatni az értéket, a View nem reagál az érték változására.
 - egyirányú a kód felől: a View-ben történt változások nincsenek hatással az értékre, a View reagál az érték változására.
 - egyszeri: a binding létrejötténél frissül a View értéke, ezekután egyik sem reagál a másikban történt változásokra.

Listák databindinghoz

```

10 references
public BindingList<Rendeles> Rendelesek { get; set; } = new();
4 references
public ObservableCollection<Foglala> Foglalasok { get; set; } = new();
5 references
public ObservableCollection<int> Asztalok { get; set; } = new();

```

- **2** fajta lista van használva az applikációban DataBindinghoz.
- A binding lehetséges, mert minden 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

```
public interface ICommand
{
    event EventHandler? CanExecuteChanged;
    bool CanExecute(object? parameter);
    void Execute(object? parameter);
}
```

- minden Gombot, vezérlőt ami kattintásra elvégez valamit egy olyan Modelhez bindolunk amit az ICommand Interface-t implementálja.

```
<Button Grid.Column="1" Command="{Binding RendelesFelveteleButton}">Rendelés felvétel</Button>
```

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

```
16 references
public class ButtonCE : ICommand
{
    Action<object?> execute;
    Func<bool> canExecute;
14 references
    public ButtonCE(Action<object?> execute, Func<bool> canExecute)
    {
        this.execute = execute;
        this.canExecute = canExecute;
    }

2 references
    public bool CanExecute(object? parameter)
    {
        return canExecute();
    }

0 references
    public void Execute(object? parameter)
    {
        execute.Invoke(parameter);
    }

    public event EventHandler? CanExecuteChanged
    {
        add
        {
            CommandManager.RequerySuggested += value;
        }
        remove
        {
            CommandManager.RequerySuggested -= value;
        }
    }
}
```

- 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 funkciót 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.

```
0 references
public ICommand RendelesFelveteleButton => new ButtonCE(RendelesHozzaad, RendelesHozzaadCE);
```

- 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 a 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.

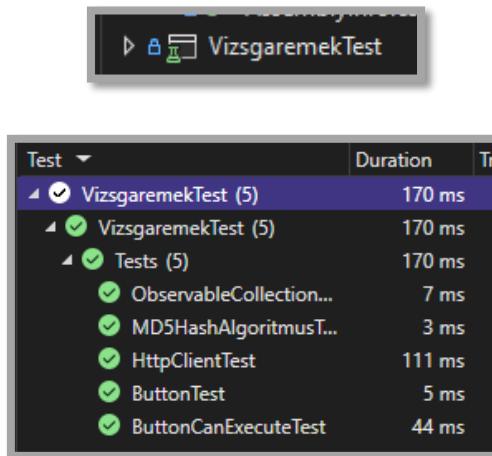
```
1 reference
private async void RendelesHozzaad(object? o)
{
```

```
private bool RendelesHozzaadCE()
{
    if (KivalasztottAsztal is not -1 && (KivalasztottFoglalas is not null || BeteroVendeg))
        return true;
    return false;
}
```

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.

Tesztek:

A solution tartalmaz egy NUnit projektet is amelyben a WPF projekt modelljeire vannak unit teszt-ek készítve.



A teszt osztályunk elkészítésénél létrehozunk egy int és egy bool változót, ezekre később a Button és ButtonCE modelljeink tesztjénél lesz szükség.

```
0 references
public class Tests
{
    private int ertek;
    private bool ceFeltetel;
    [SetUp]
    public void Setup()
    {
        ertek = 0;
        ceFeltetel = false;
    }
}
```

Az első teszeset az MD5 hash algoritmust teszteli, ezt használjuk az alkalmazottak regisztrációjánál a WPF alkalmazásunkban.

```
[Test]
0 | 0 references
public void MD5HashAlgoritmusTest()
{
    string hashedmd5pw = MD5Hashing.hashPW("unittest");
    Assert.AreEqual("16802231b09f155b7a42a5dcaba33a74", hashedmd5pw);
}
```

A második teszteset a HttpClient köré készített osztályunk Json kreálását teszteli, a teszteléshez létre lett hozva egy record típus, amelynek egy példányán teszteljük a Json formátummá alakítást.

```
2 references
internal record HttpTestRecord(int Id, string Nev, int[] szamok);

[Test]
0 | 0 references
public void HttpClientTest()
{
    HttpTestRecord test = new(1, "testing", new int[] { 1, 2, 3 });
    HttpClientClass _http = new();
    var stringContent = _http.contentKreaslas(test);
    string content = stringContent.ReadAsStringAsync().Result;
    Assert.AreEqual("{\"Id\":1,\"Nev\":\"testing\",\"szamok\":[1,2,3]}", content);
}
```

A harmadik teszteset a minden aktív gombunk működését teszteli, létrehozunk belőle egy példányt, paraméterben átadunk egy anonim Action-t amely majd később a gomb Execute eljárásánál lesz végrehajtva.

```
[Test]
0 | 0 references
public void ButtonTest()
{
    Button button = new(() => ertek = 1);
    button.Execute(null);
    Assert.AreEqual(1, ertek);
}
```

A negyedik teszteset a ButtonCE gomb modellünket teszteli amelynek az aktív státusza a CanExecute metódusától függ. Az első anonim Action az Execute eljárása, a második anonim Func a SetUp-ban beállított értékű ceFeltetel változó értékét adja vissza, szóval ettől függ hogy a gombunk aktív-e vagy nem. A SetUp-ban false-ra van állítva az érték, szóval először nem szabad hogy aktív legyen a gomb, miután ezt megváltoztatjuk true-ra a gomb Execute-olható kell hogy legyen.

```
[Test]
0 | 0 references
public void ButtonCanExecuteTest()
{
    ButtonCE button = new(() => ertek = 1, () => ceFeltetel);
    Assert.IsFalse(button.CanExecute(null));
    ceFeltetel = true;
    Assert.IsTrue(button.CanExecute(null));
}
```

Az ötödik teszteset az ObservableCollection gyűjtemény általam megírt kiegészítő metódusát teszteli. Erre azért volt szükség mert alapvetően egy ObservableCollection-t nem lehet rendezni, csak úgy ha egy teljesen új példányt hozunk létre belőle, ami nem lenne jó az UI / DataBinding szempontjából.

```
2 references | 0/1 passing
public static void Sort<T>(this ObservableCollection<T> collection) where T : IComparable
{
    List<T> sorted = collection.OrderBy(x => x).ToList();
    for (int i = 0; i < sorted.Count(); i++)
        collection.Move(collection.IndexOf(sorted[i]), i);
}
```

Ez minden olyan típusú ObservableCollection-re működik ahol a típus az IComparable interface-t örökli. Az ObservableCollection-ból készítünk egy List-et, ami már LINQ-val rendezhető, ezután végigmegyünk egy ciklussal a rendezett listán, és a .Move metódusát használva a kapott ObservableCollection-ben átmozgatjuk az értékeket a megfelelő helyre.

```
[Test]
0 | 0 references
public void ObservableCollectionExtensionMethodTest()
{
    ObservableCollection<int> collection = new() { 128, 2, 338, 1000, 500 };
    collection.Sort();
    Assert.AreEqual(new ObservableCollection<int> { 2, 128, 338, 500, 1000 }, collection);
}
```

Felhasználói dokumentáció

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

Bejelentkezés

The screenshot shows a Windows-style application window titled "Bejelentkezés". Inside, there's a heading "Üdvözlünk!" and "Személyzeti bejelentkezés". Below this, there are two input fields: "Email cím:" and "Jelszó:". A large green button labeled "Bejelentkezés" is centered below the fields. At the bottom left, there's a link "Nincs még felhasználója? Regisztráció".

A felület részei:

- Email mező
- Jelszó mező
- Bejelentkezés gomb
- Regisztráció gomb

A Bejelentkezés gomb aktívvá tételehez mindenki beviteli mezőt ki kell tölteni. A gomb megnyomásával a REST API hívjuk meg, ami sikeres bejelentkezés esetén visszaküldi a bejelentkezett felhasználót.

A Regisztráció gombra kattintás során egy új dialóg ablak nyílik meg.

Regisztráció

The screenshot shows a registration form window titled "Regisztráció". The form contains the following fields:

- Email: [Text input field]
- Lakhely: [Text input field]
- Telefonszám: [Text input field]
- Teljes Név: [Text input field]
- Jelszó: [Text input field]
- Jelszó újra: [Text input field]
- Jogosultság: [Dropdown menu currently set to "Felszolgáló"]

At the bottom right of the form area is a teal-colored button labeled "Regisztráció".

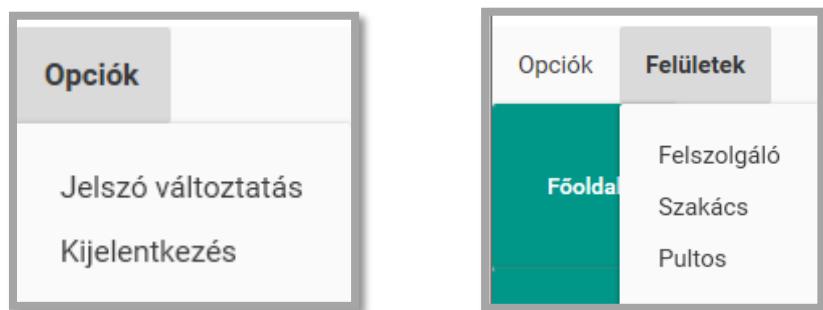
A felület részletei:

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

A regisztráció gomb aktívvá tételehez 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:

The image shows a window titled 'Jelszó Változtatás'. Inside, there are three text input fields: 'Aktuális jelszó:' (Current password), 'Új jelszó:' (New password), and 'Új jelszó újra:' (Re-enter new password). Below these fields is a green button labeled 'Jelszó változtatása' (Change password).

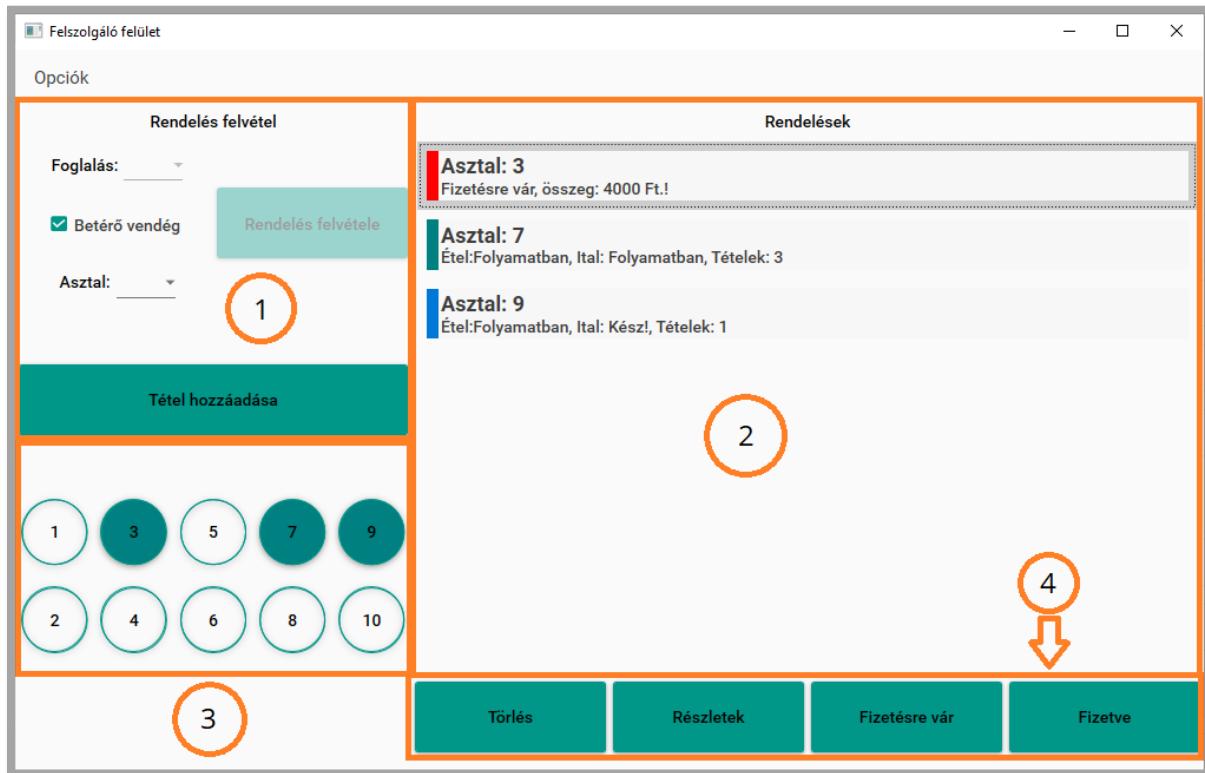
A felület részletei:

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:
 - 1.1.Foglalás választó
 - 1.2.Betérő vendég választó
 - 1.3.Asztal választó
 - 1.4.Rendelés Felvétele gomb
 - 1.5.Tétel hozzáadása gomb
2. Rendelések:
 - 2.1.Rendelés:
 - 2.1.1. Folyamatban lévő (Kékeszöld jelölő)
 - 2.1.2. Egyik típusú tétel kész (Kék jelölő)
 - 2.1.3. Fizetésre várakozó (Piros jelölő)
3. Asztalok:
 - 3.1.Kékeszöld hátterű asztal: Foglalt
 - 3.2.Átlátszó hátterű asztal: Szabad
4. Also vezérlő gombok:
 - 4.1.Törlés
 - 4.2.Részletek
 - 4.3.Fizetésre vár
 - 4.4.Fizetve

- Egy rendelés státusza:

- Étel: Folyamatban, Ital: Folyamatban:
 - - Ha nincs még kész az összes benne lévő térel étele / itala
- Étel: Kész, Ital: Folyamatban (vagy fordítva):
 - Ha az összes térel 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 térel 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 vezérlővel. 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 hívunk meg, 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 kiválasztott foglaláshoz (kivéve, ha betérő vendég be van pipálva.)

- „Tétel hozzáadása” gomb: megnyit egy új felületet amivel egy új térelt lehet hozzáadni a kiválasztott 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: Töri 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:

Hamburger:	Darab:	Leírás:
-	1	-- 0 Ft.
Köret:	Darab:	Leírás:
Testköret	1	Test köret leírás - 1000 Ft.
Ital:	Darab:	Leírás:
Testital	1	Test ital leírás - 1000 Ft.
Desszert:	Darab:	Leírás:
-	1	-- 0 Ft.
Megjegyzés:		
Tétel hozzáadása		

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 térelhez. A térel hozzáadása gomb nyomása után a térel hozzáadásra kerül a kiválasztott rendeléshez.

Rendelés részletei:

Tételek:

21.számú tétel.
Étel: Folyamatban, Ital: Kész!

22.számú tétel.
Étel: Folyamatban, Ital: Folyamatban

23.számú tétel.
Étel: Folyamatban, Ital: Kész!

Rendelés módosítása, törlése:

Hamburger:	Darab:
Testburger	1

Köret:	Darab:
Testkoret	1

Ital:	Darab:
Testital	1

Desszert:	Darab:
-	1

Megjegyzés:

Tétel törlése! **Tétel módosítása!**

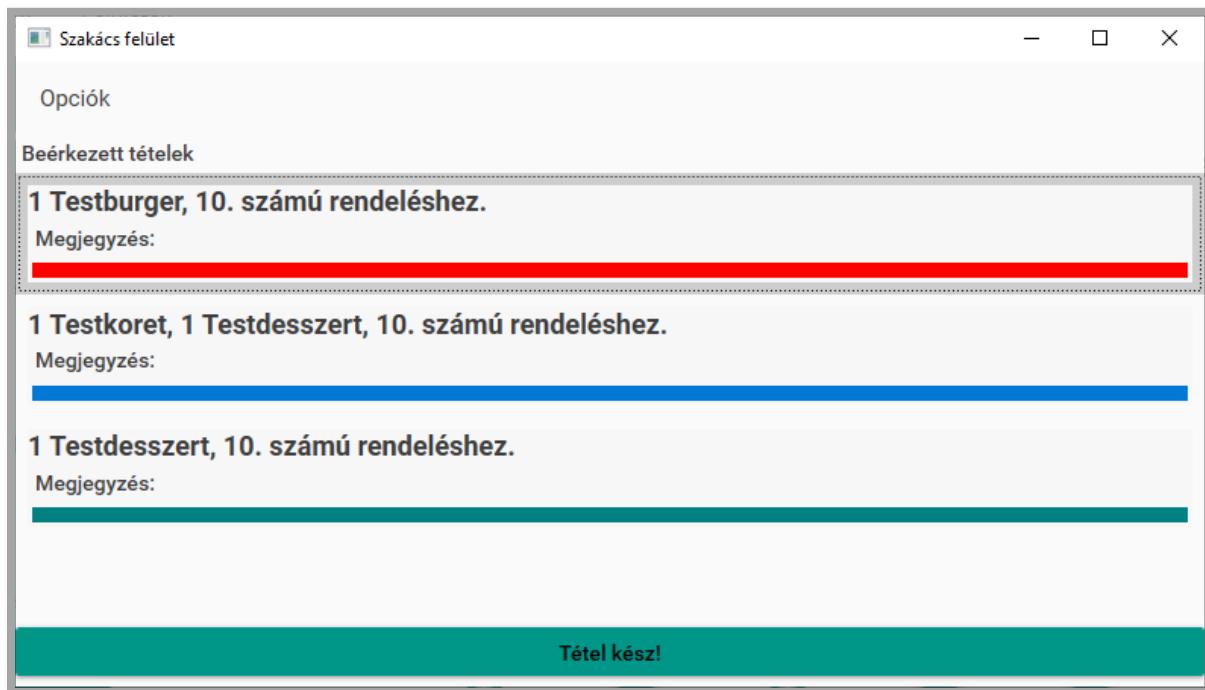
A felület részletei:

1. Tételek lista
2. Rendelés részletei
 - 2.1.Hamburger, köret, ital, desszert választó
 - 2.2.Darab szövegdobozok
 - 2.3.Megjegyzés szövegdoboz
 - 2.4.Tétel törlése gomb
 - 2.5.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 (lehetségesse teszi, hogy egy rendeléshez tartozó tételeket csinálunk).

Alatta a térelhez tartozó megjegyzés látható (ha van).

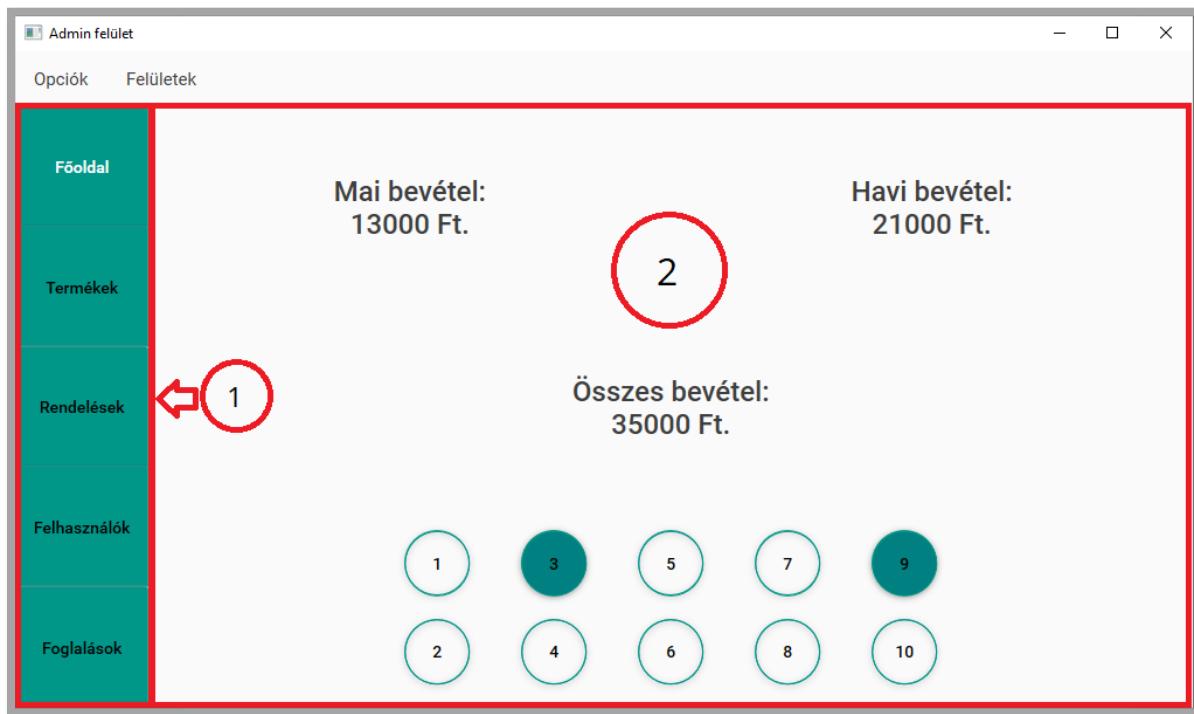
A tételek alján egy vízszintes csík található, ami:

- A térel hozzáadása és <30 mp között Kékeszöld.
- 30 és 60 mp között Kék.
- > 60 mp 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 !!!

Adminisztrátor felület:



A felület részletei:

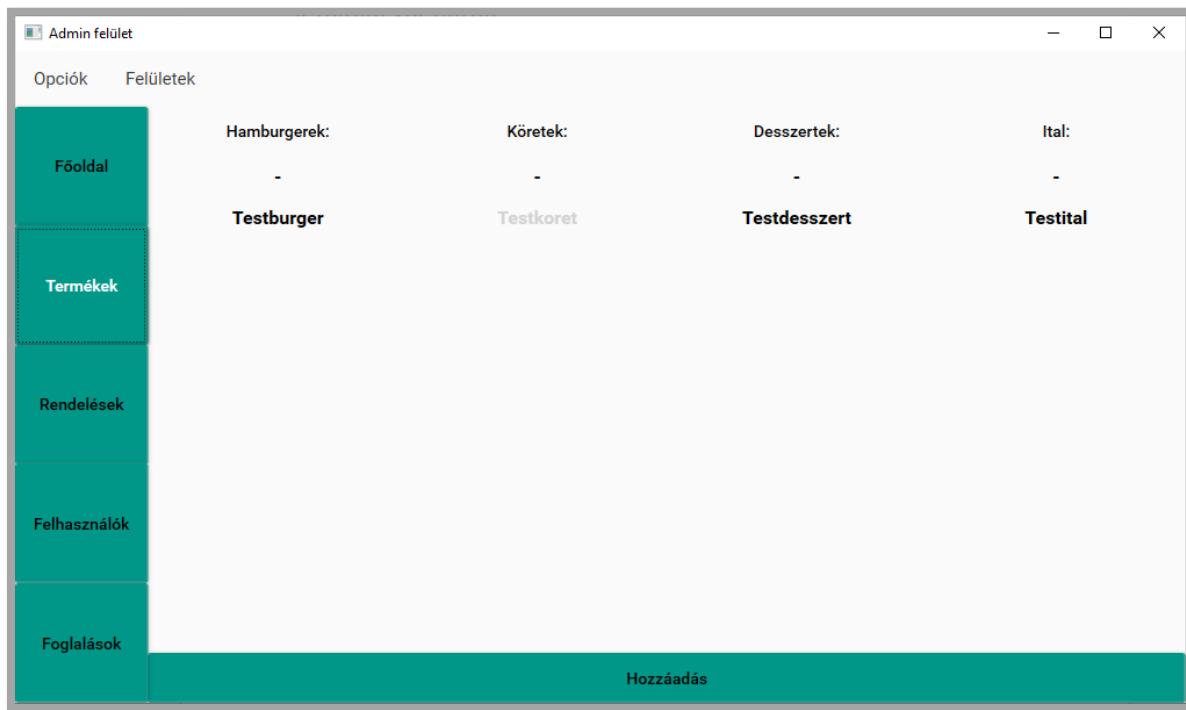
1. Navigációs menü
 - 1.1.Főoldal (áttekintés)
 - 1.2.Termékek kezelése
 - 1.3.Rendelések történet
 - 1.4.Felhasználók kezelése
 - 1.5.Foglalások történet
2. Az adott menü tartalma (ContentControl)

Főoldal(áttekintés):

- Mai, havi, összes bevétel
- Asztalok

A felület 5 másodpercenként frissül, ez lekéri a bevételeket és az aktív rendeléseket, amelyekből 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 árusításra, a felszolgáló téTEL hozzáadása ablakában sem jelennek meg.

Egy termék dupla bal kattintására a következő felület jelenik meg:

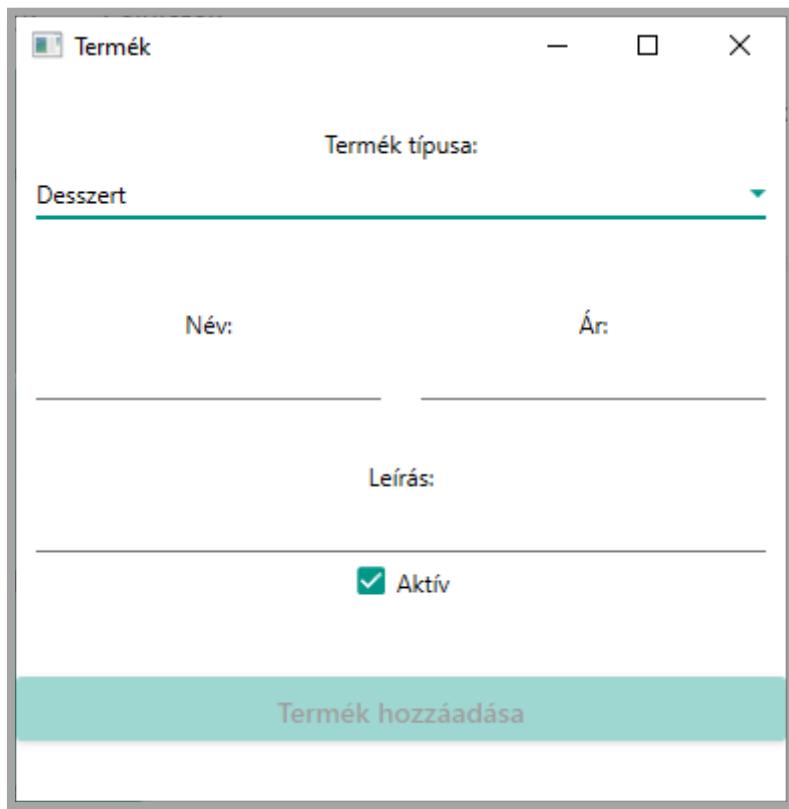
A felület részletei:

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:

The screenshot shows a Windows application window titled "Admin felület". On the left is a vertical navigation menu with buttons for "Főoldal", "Termékek", "Rendelések" (which is highlighted with a dotted border), "Felhasználók", and "Foglalások". At the top right are standard window controls for minimize, maximize, and close. Below the menu, there's a header "Rendelési előzmények" and a sub-header "1. számú rendelés, 2022.2.25 23.02". A table follows, showing an order for "Testburger" with items Hamburger, DB, Desszert, Köret, Ital, and Végösszeg. Below this is another order entry for "Testdesszert". Further down, sections for "2. számú rendelés, 2022.2.25 23.02", "3. számú rendelés, 2022.2.25 23.03", "4. számú rendelés, 2022.1.26 23.03", and "5. számú rendelés, 2022.3.2 09.11" are listed with their respective dates.

A felület részletei:

1. Rendelési előzmények lista

A listában minden eddigி, már fizetett rendelés látható. 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:

The screenshot shows the same "Admin felület" window. The "Felhasználók" button in the sidebar is highlighted with a dotted border. In the main area, there are two columns: "Vendégek:" and "Alkalmazottak:". Under "Vendégek:", there are entries for "Vendég" and "sipos". Under "Alkalmazottak:", there are entries for "Admin", "Test1", "Test2", and "Test3". At the bottom of the screen, a green bar contains the text "Admin felhasználó hozzáadása".

A felület részletei:

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

A listákban a regisztrált vendégek és alkalmazottak találhatóak.

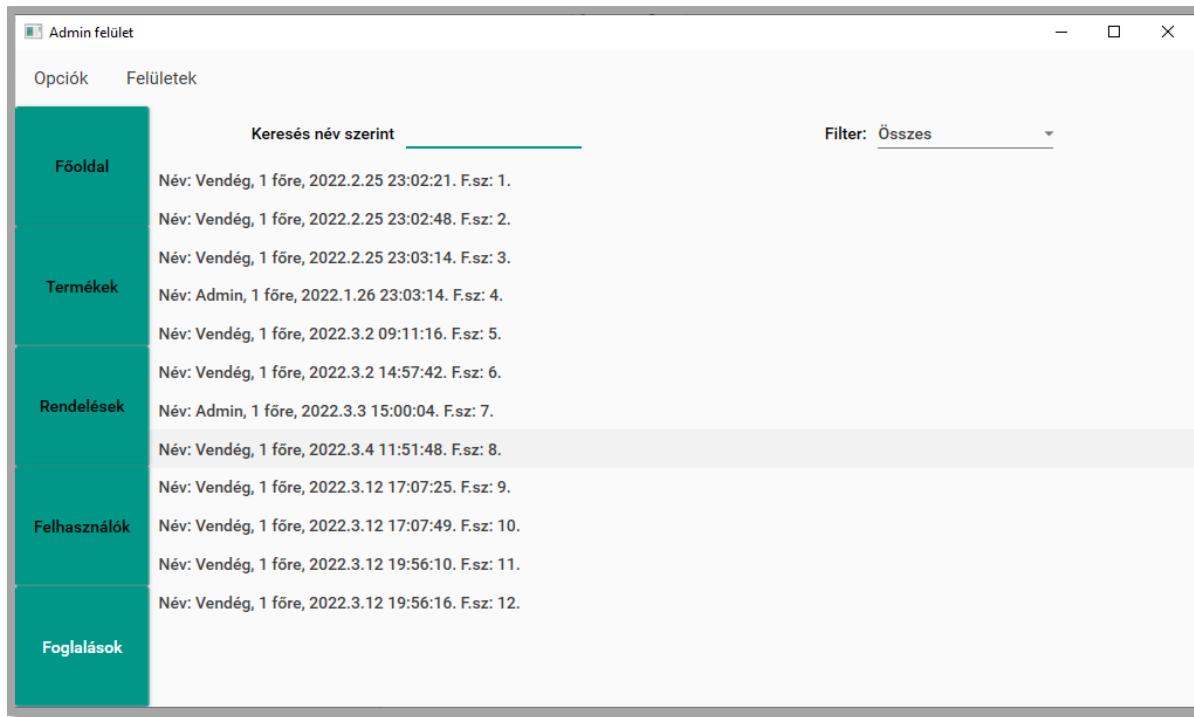
Az alkalmazottak listában dupla bal kattintásra egy felhasználó módosítása felület jelenik meg.

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.

Adminisztrátor regisztrálásnál a „Jog” választó inaktív, és “Admin” van kiválasztva benne.

Foglalások (Admin) felület:



A felület részletei:

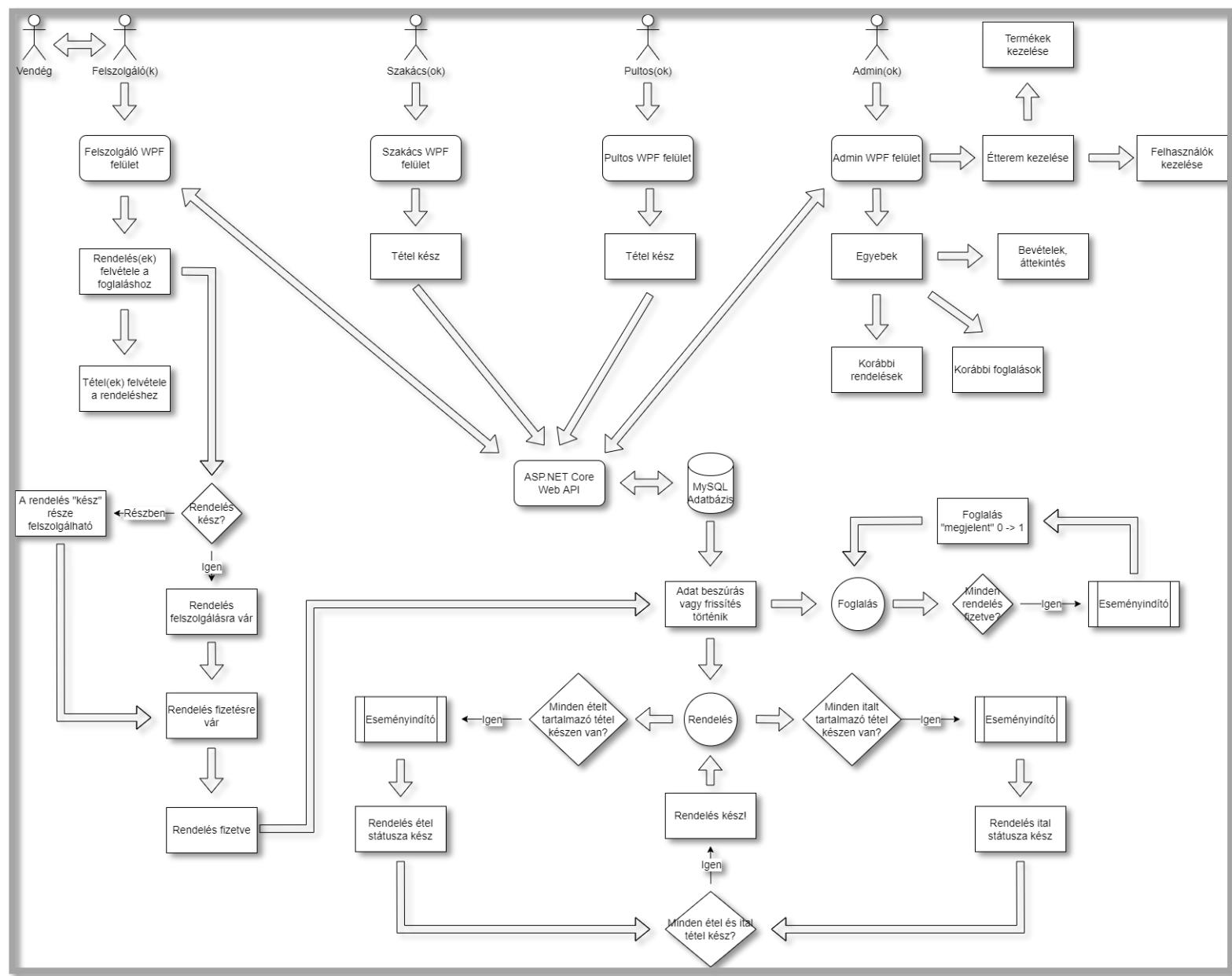
1. Keresés név szerint mező
2. Filter választó (Összes, Mai nap, Ez a hónap)
3. Foglalások lista

A listában a már megjelent vendégek foglalásai találhatóak. A foglalás megnevezése (neve), a vendég nevéből, a foglalás dátumából és a foglalás számából 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.

Az alkalmazás egyszerűsített kapcsolati/logikai felépítés:



FEJLESZTÉSI LEHETŐSÉGEK

Weboldal:

- „Kapcsolat” oldal létrehozása elérhetőségekkel, az étterem helyének térképes megjelenítésével;
- a weboldal ’akadálymentesítése’, fogyatékossággal élő felhasználók részére;
- telefonos applikáció fejlesztése.

Asztali alkalmazás:

- az étterem vezetőségének felépítésétől függően „üzletvezető” jogosultság/felhasználói felület fejlesztése;
- online fizetési és számlázási lehetőség fejlesztése.

Felhasznált irodalom:

- <https://en.wikipedia.org/wiki/MySQL>
- https://bhawk.hu/wp-content/uploads/2019/05/dotnet5_platform.png
- <https://dotnet.microsoft.com/en-us/apps/aspnet>
- <https://www.tutorialsteacher.com/core>
- https://en.wikipedia.org/wiki/Representational_state_transfer
- <https://www.redhat.com/en/topics/api/what-is-a-rest-api>
- https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>
- https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL
- <https://en.wikipedia.org/wiki/URL>
- <https://docs.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-6.0>
- https://en.wikipedia.org/wiki/Windows_Presentation_Foundation
- <https://en.wikipedia.org/wiki/Model%20view%20viewmodel>
- <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm>
- <https://www.tutorialspoint.com/mvvm/index.htm>
- <https://en.wikipedia.org/wiki/.NET>
- <https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-5>
- <https://docs.microsoft.com/en-us/nuget/what-is-nuget>
- <http://materialdesigninxaml.net/>
- <https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>
- <https://docs.microsoft.com/en-us/ef/core/>
- https://hu.wikipedia.org/wiki/Rel%C3%A1ci%C3%A3o%C3%A3s_adatb%C3%A1zis
- <https://www.w3schools.com/mysql/default.asp>
- <https://hu.wikipedia.org/wiki/HTML>
- https://hu.wikipedia.org/wiki/Cascading_Style_Sheets
- <https://developer.mozilla.org/en-US/docs/Web/CSS>
- <https://www.w3.org/Style/CSS/Overview.en.html>
- [https://en.wikipedia.org/wiki/Bootstrap_\(front-end_framework\)](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework))
- https://szit.hu/doku.php?id=oktatas:web:front-end_framework:bootstrap
- <https://hu.wikipedia.org/wiki/JavaScript>
- <https://webiskola.hu/javascript-ismeretek/mi-az-a-javascript-a-js-bemutatasa/>
- <https://hu.reactjs.org/docs/getting-started.html>
- <https://hu.wikipedia.org/wiki/PHP>